ShaDDR: Real-Time Example-Based Geometry and Texture Generation via 3D Shape Detailization and Differentiable Rendering

QIMIN CHEN, Simon Fraser University, Canada
ZHIQIN CHEN, Simon Fraser University, Canada
HANG ZHOU, Simon Fraser University, Canada
HAO ZHANG, Simon Fraser University, Canada

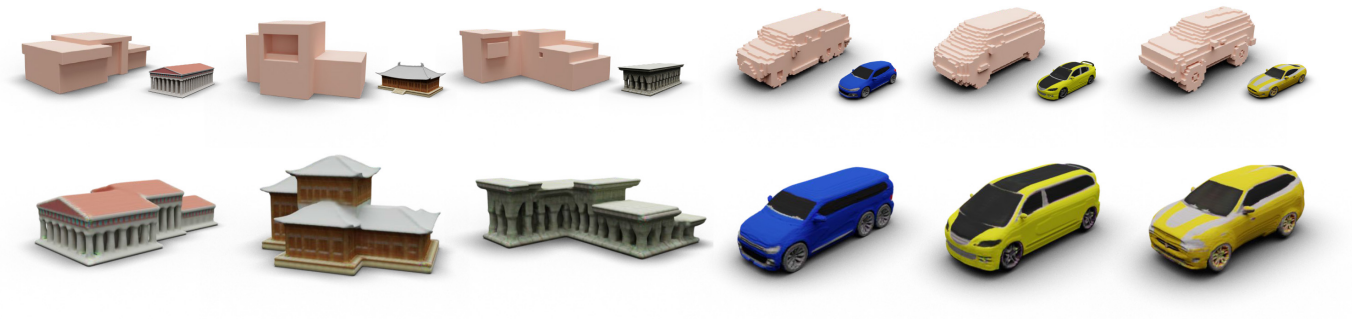


Fig. 1. Given a coarse voxel shape and a textured exemplar shape (top row), our network generates a *geometrically detailized* and *textured* version of the coarse shape (bottom row) in *less than 1 second*, with geometry and texture generations both conditioned on the exemplar. Zoom in to view surface details.

We present ShaDDR, an *example-based* deep generative neural network which produces a high-resolution textured 3D shape through *geometry detailization* and *conditional texture generation* applied to an input coarse voxel shape. Trained on a small set of detailed and textured exemplar shapes, our method learns to detailize the geometry via *multi-resolution* voxel upsampling and generate textures on voxel surfaces via differentiable rendering against exemplar texture images from a few views. The generation is real-time, taking less than 1 second to produce a 3D model with voxel resolutions up to 512^3. The generated shape preserves the overall structure of the input coarse voxel model, while the style of the generated geometric details and textures can be manipulated through learned latent codes. In the experiments, we show that our method can generate higher-resolution shapes with plausible and improved geometric details and clean textures compared to prior works. Furthermore, we showcase the ability of our method to learn geometric details and textures from shapes reconstructed from real-world photos. In addition, we have developed an interactive modeling application to demonstrate the generalizability of our method to various user inputs and the controllability it offers, allowing users to interactively sculpt a coarse voxel shape to define the overall structure of the detailized 3D shape.

1 INTRODUCTION

Deep generative models for 3D shapes have made significant advances in recent years and lately, the progress has been propelled by exciting developments on diffusion and large language models to improve usability via text prompting and generality through zero-shot learning. However, these new advances still do not alleviate fundamental issues such as lack of fine-grained control, whether the generation is from noise or texts, and slow speed, especially when diffusion models are employed. Case in point, state-of-the-art text-prompted, diffusion-based 3D generator, Magic3D [Lin et al. 2023], currently takes about 40 minutes to produce a result. Furthermore, like most results of this kind, e.g., DreamFusion [Poole et al. 2022], Get3D [Gao et al. 2022], and Make-it-3D [Tang et al. 2023], the generated geometry is low-resolution and lacks details.

DECOR-GAN [Chen et al. 2021] is a recent work that takes a more traditional approach to generate higher resolution geometries, through a process called *geometry detailization*. Specifically, their deep generative network *stylizes* an input coarse voxel shape, via voxel *upsampling*, conditioned on an input exemplar shape with geometric details. With such a modeling approach, artists retain some level of fine-grained control as they can freely edit the coarse voxels to define the overall structure of the detailized shape. The detailization is also fast as the network takes one single forward pass with both the generator and discriminator built by 3D CNNs. However, with only a single-resolution upsampling, the generated results still cannot transfer geometric details at finer scales. More critically, like most existing approaches for deep generative modeling of 3D shapes, DECOR-GAN only focuses on generating shape geometries and neglects the equally vital aspect of textures.

In this paper, we present an *example-based* generative network which produces high-resolution *textured* 3D shapes through geometry detailization, akin to DECOR-GAN, and texture generation via differentiable rendering. As show in Figure 1, our network, coined ShaDDR (for SHApe Detailization and Differentiable Rendering), transfers the geometric details and textures from an exemplar textured shape which provides the geometry and texture "styles" or conditions for the generative task. The generation is *real-time*, taking less than 1 second to produce a 256^3-voxel building model.

Technically, our method operates in two separate phases, as illustrated in Figure 2. The geometry detailization phase is built on

Authors' addresses: Qimin Chen, Simon Fraser University, Canada, qca43@sfu.ca; Zhiqin Chen, Simon Fraser University, Canada, zhiqinc@sfu.ca; Hang Zhou, Simon Fraser University, Canada, hza162@sfu.ca; Hao Zhang, Simon Fraser University, Canada, haoz@sfu.ca.

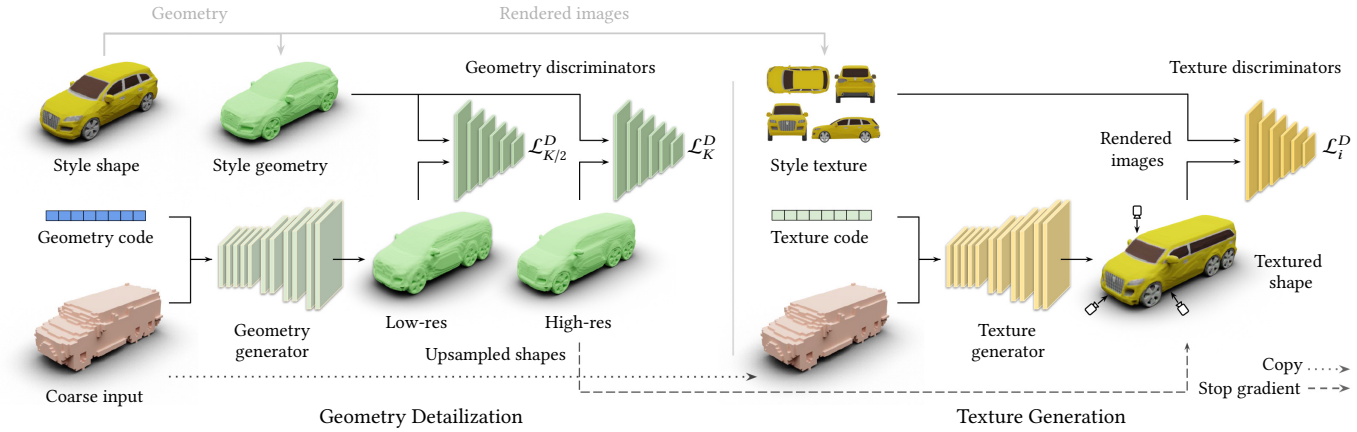
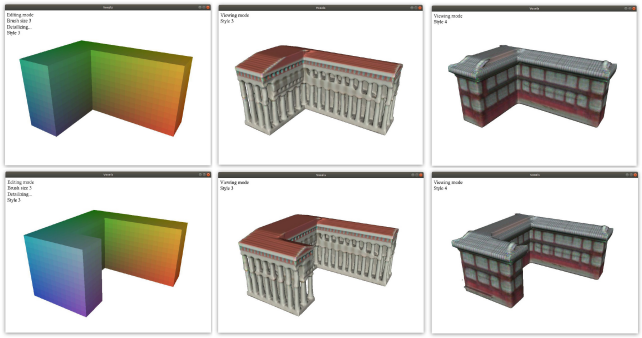


Fig. 2. An overview of our ShaDDR's two-phase solution pipeline and network architecture, for which the input "style shape" provides the exemplars for both detailed geometry and multi-view textures. Conditioned on the geometry code, the geometry generator upsamples a coarse input voxel grid into detailed geometries in multiple (two) resolutions, $(K/2)^3$ and K^3. The geometry discriminators enforce the local patches of the upsampled geometries to be plausible with respect to the target geometry style. The texture generator takes in the texture code and the same coarse voxels and synthesizes 3D volumetric textures for the upsampled geometry. The generated geometry and textures are rendered into 2D images from different views, and the texture discriminators enforce the local patches of the rendered images to be plausible with respect to the target texture style.



(a) Interactive editing (b) ShaDDR result #1 (c) ShaDDR result #2

Fig. 3. A GUI for interactive modeling by ShaDDR. Users can edit coarse voxels and visualize different detailized and textured shapes in real time.

DECOR-GAN, sharing the same input setting and similar network architecture, except that we exploit the intrinsic hierarchical nature of voxel upsampling to generate detailizations at *multiple resolutions* (two in the current implementation). The key advantage achieved, as a direct benefit from additional supervision at intermediate-level voxel resolutions, is higher-quality generation of finer-scale geometric details over DECOR-GAN; see the several zoom-ins in Figure 1 and a comparison presented in Figure 4.

The texture generator takes as input exemplar texture *images* from a small number of views (four or five depending on the object categories), as well as the coarse voxel shape which was fed to the geometry generator. It synthesizes 3D volumetric textures for the highest-resolution upsampled geometry from the first phase. The texture synthesis is learned by employing *image-space* GANs via differentiable rendering against the multi-view exemplar textures. In our current work, the texture exemplars are multi-view projections from a textured 3D shape. In general, such images may be obtained without a pre-defined 3D model; they may be sampled from a neural radiance field (NeRF) [Yu et al. 2021] or taken as photographs of real-world objects (see the first application in Section 4.3).

To our knowledge, ShaDDR represents the first deep generative model that upsamples a coarse voxel model into a fully *textured* and geometrically detailed 3D shape. Compared to DECOR-GAN, our method shows a clear advantage in geometry detailization owing to its multi-resolution generator, while the conditionally generated texture significantly improves the richness and expressiveness of the final shape. At the same time, all the merits of DECOR-GAN are retained, including the kind of fine-grained structural control for artists, real-time generation to facilitate exploratory 3D modeling, and effective reuse of existing digital assets. With the ShaDDR network, a large variety of high-resolution and detailed 3D geometries with textures can be easily created from a small set of exemplars, as demonstrated by extensive experiments in Section 4.

Finally, we show two applications: a) detailed 3D shape generation with textures provided from photographs of real-world objects; 2) an interactive modeling GUI for a user to create detailed models by sculpting coarse voxels; see Figure 3 for a few screenshots and the supplementary video for interactive demos[1].

[1]Note that the model generation time within the GUI is about 2s, which is longer than that of pure network inference (<1s) due to mesh and texture exporting overhead.

2 RELATED WORK

Our work is closely related to 3D generative models and texture synthesis. We also discuss relevant works on few-shot generative models, i.e., models that can generate shapes when given only one or a few high-quality exemplars.

3D Generative Models. Following the introduction of variational autoencoders (VAEs) [Kingma and Welling 2013], generative adversarial networks (GANs) [Goodfellow et al. 2020], autoregressive models [Van den Oord et al. 2016], and normalizing flows [Rezende and Mohamed 2015], various deep generative models for 3D shapes have been developed, utilizing a wide range of shape representations, including point clouds [Achlioptas et al. 2018; Fan et al. 2017; Yin et al. 2019], voxels [Choy et al. 2016; Häne et al. 2017; Wu et al. 2016], deformable meshes [Groueix et al. 2018; Wang et al. 2018; Zhang et al. 2021], part-based structural graphs [Gao et al. 2021, 2019; Li et al. 2017; Mo et al. 2019], and neural implicit functions [Chen and Zhang 2019; Mescheder et al. 2019; Park et al. 2019].

Recent methods [Cheng et al. 2023; Hui et al. 2022; Zeng et al. 2022] based on diffusion probabilistic models [Ho et al. 2020; Sohl-Dickstein et al. 2015] have achieved significant improvements in the geometric details of the generated shapes, yet they solely focus on geometry synthesis and do not generate textures. Get3D [Gao et al. 2022] relies on differentiable rendering, differentiable Marching Tetrahedra [Shen et al. 2021], and 2D GANs, therefore it is able to generate textured meshes. However, it performs unconditional generation and does not provide control over the generated shapes. Other works [Lin et al. 2023; Melas-Kyriazi et al. 2023; Poole et al. 2022] based on differentiable rendering and test-time optimization can synthesize shape with input text or single image conditions. Since these methods need to overfit a NeRF [Mildenhall et al. 2021] model for each different input, they take on the order of many minutes to hours to generate a single shape. Besides, all these methods need to be trained with significant amounts of high-quality 3D shapes or 2D images, which are not always easily available.

In contrast, our method only needs a few (8-16) detailed and textured shapes to provide geometry and texture supervision. During test time, our method can upsample a given coarse voxel grid into a detailed shape with textures in less than 1 second. Such a real-time performance enables artists and casual users to create high-quality 3D models interactively with high controllability.

Few-shot Generative Models. Several classic methods from computer graphics are able to synthesize new content by copying local patches from a given source and joining those patches together to form the target object. Image quilting [Efros and Freeman 2001] is able to synthesize texture images of arbitrary sizes given an exemplar texture image. Similarly, [Merrell 2007; Merrell and Manocha 2008, 2010] can synthesize new 3D models given an exemplar 3D shape; mesh quilting [Zhou et al. 2006] can transfer geometric textures from a geometric texture patch to the surface of a coarse shape; and MeshMatch [Chen et al. 2012] can transfer color textures from one textured shape to another textureless shape.

In the deep learning era, the explicit copy-join steps have been replaced by local patch discriminators such as PatchGAN [Isola et al. 2017]. SinGAN [Shaham et al. 2019], as well as earlier work by [Zhou et al. 2018], learns the distribution of patches within a given image in different scales, and is able to generate diverse images that carry the same content and texture as the given image. Similarly, the work by [Wu and Zheng 2022] learns 3D voxel patches and is able to generate diverse 3D shapes of the same style given a single exemplar shape. 3inGAN [Karnewar et al. 2022] and SinGRAV [Wang et al. 2022] are based on NeRF [Mildenhall et al. 2021] and they can generate diverse scenes from an exemplar scene. However, these methods perform unconditional generation and do not provide control over the generated shapes. SketchPatch [Fish et al. 2020] aims for sketch stylization and is able to convert plain solid-lined sketches into diverse sketches with different line styles.

Most closely related to our work, DECOR-GAN [Chen et al. 2021] targets voxel detailization and is designed to generate detailed voxel shapes from input coarse voxels with respect to the style constraint controlled by a latent code. Our work is based on DECOR-GAN. We not only enable DECOR-GAN to generate higher-resolution geometric details by designing hierarchical structures in the neural networks, but also generate detailed textures on the output shapes, making them readily usable in real applications.

Texture Synthesis. Since classic texture synthesis methods focus on generating regular or stochastic textures on images [Cross and Jain 1983; Efros and Freeman 2001; Efros and Leung 1999] or volumetric textures [Kopf et al. 2007], we will mainly discuss recent works that generate semantic-aware textures on surfaces of 3D shapes via deep learning. Various representations have been proposed to represent textures in neural networks. Deformation-based approaches that deform a sphere primitive [Chen et al. 2019b; Henderson et al. 2020; Li et al. 2020; Mohammad Khalid et al. 2022; Monnier et al. 2022; Pavllo et al. 2021, 2020; Zhang et al. 2020] or a set of cuboids [Gao et al. 2021] can directly use the UV mapping defined on the primitives, therefore they only need to generate texture images.

To texture complex shapes, some works adopt texture images with pre-defined UV mapping [Chaudhuri et al. 2021; Richardson et al. 2023; Yin et al. 2021], or learn the UV mapping with a neural network [Chen et al. 2022]. Text2shape [Chen et al. 2019a] adopts the voxel representation and generates RGB color for each output voxel. Texturify [Siddiqui et al. 2022] directly generates textures on the shape surface by introducing face convolutional operators. The work by [Raj et al. 2019] generates multi-view images and projects the images back to the shape to obtain the textures.

Recently, with the introduction of Texture Fields [Oechsle et al. 2019] and NeRF [Mildenhall et al. 2021], a number of works [Chan et al. 2022, 2021; Gao et al. 2022; Michel et al. 2022; Niemeyer and Geiger 2021; Rebain et al. 2022; Schwarz et al. 2020; Skorokhodov et al. 2022] adopt neural fields to represent volumetric textures, possibly with view-dependent colors. Our work is similar to Text2shape [Chen et al. 2019a] in that we generate a grid of colored voxels. However, since our model is supervised via differentiable rendering, only surface voxels will receive gradients during training, similar to Texture Fields [Oechsle et al. 2019]. Our method also provides control over the generated textures via a texture latent code.

3 METHOD

In this section, we introduce our conditional generative model, ShaDDR, that learns to upsample coarse voxels into detailed 3D geometries and corresponding textures. Specifically, given a low-resolution coarse voxel grid of k^3 resolution as the "content shape", and a pair of geometry and texture codes learned from high-resolution detailed "style shapes" during training, our method can generate a novel textured shape up to K^3 resolution that preserves the coarse structure of the content shape, while synthesizing geometric details and textures in a similar style to that of the style shape corresponding to the input geometry and texture codes. To achieve this, we devise a multi-resolution GAN operating on 3D voxels for high-resolution (up to 512^3) geometry detailization and a set of GANs operating on 2D images rendered from the generated shapes for conditional fine texture generation.

3.1 Multi-resolution geometry detailization

When leveraging the original DECOR-GAN [Chen et al. 2021] on super high-resolution voxels, e.g., upsampling 8 times on each dimension, its performance degrades significantly, as shown in Figure 4 (b) where the thin structures are broken and the details are lost. This is likely due to the difficult and ambiguous nature of voxel upsampling with large upsampling factors, which can be mitigated by introducing supervision on intermediate voxel resolutions. Therefore, we employ a multi-resolution GAN by generating an intermediate lower-resolution voxel output and a final high-resolution voxel output, and applying adversarial training on both outputs.

As shown in Figure 2 left, given a coarse content shape represented as occupancy voxels at resolution k^3 and a latent geometry code representing the geometric style of a training style shape, the geometry generator is trained to upsample the content shape into two resolutions, $(K/2)^3$ and K^3. The two upsampled shapes are then fed into two distinct 3D CNN PatchGAN discriminators [Isola et al. 2017]. We use the same embedding module as DECOR-GAN to learn an 8-dimensional latent geometry code for each style shape.

Training losses. Following DECOR-GAN, we denote the set of detailed style shapes as $\mathcal{S}$ and the set of coarse content shapes as C. We assume that a detailed shape $s \in \mathcal{S}$ has both geometry and texture, and the geometry represented as voxels is denoted as s^{geo}. Content shapes $c \in C$ are coarse voxels without textures. The geometry code representing the style of s is denoted as z_s^{geo}. The geometry generator and the geometry discriminator at resolution K^3 are denoted as G_K^{geo} and D_K^{geo}, respectively. Note that the discriminator is designed to be conditioned on each different style in $\mathcal{S}$ so that the style of the output shape can be controlled by z_s^{geo}. We adopt the same design as DECOR-GAN and leave the details in the supplementary. We also adopt the binary generator and discriminator masks proposed in DECOR-GAN to focus the networks' capacity on voxels close to the shape surface. The values in the masks are defined to be 1 near the surface and 0 elsewhere; see supplementary for detailed definitions. The generator masks of shapes s and c at resolution K^3 are denoted as $M_{s\cdot K}^G$ and $M_{c\cdot K}^G$, respectively. Similarly, the discriminator masks are denoted as $M_{s\cdot K}^D$ and $M_{c\cdot K}^D$. We adapt the adversarial loss from DECOR-GAN to our multi-resolution setting and only define discriminator and generator losses on the final resolution of K^3 for simplicity; losses for $(K/2)^3$ resolution can be derived by changing K to $K/2$. The discriminator loss is defined as:

$$\mathcal{L}_K^D = \mathop{\mathbb{E}}_{s\sim\mathcal{S}} \frac{||(D_K^{geo}(s^{geo}) - 1) \circ M_{s\cdot K}^D||_2^2}{||M_{s\cdot K}^D||_1} + \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{S}\\ c\sim C}} \frac{||D_K^{geo}(c_{s\cdot K}^{geo}) \circ M_{c\cdot K}^D||_2^2}{||M_{c\cdot K}^D||_1},$$

$$c_{s\cdot K}^{geo} = G_K^{geo}(c,\, z_s^{geo}) \circ M_{c\cdot K}^G, \tag{1}$$

where $\circ$ denotes element-wise multiplication, and $c_{s\cdot K}^{geo}$ is the upsampled shape of resolution K^3 from input coarse shape c with the style of s. The generator loss is defined as:

$$\mathcal{L}_K^G = \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{S}\\ c\sim C}} \frac{||(D_K^{geo}(c_{s\cdot K}^{geo}) - 1) \circ M_{c\cdot K}^D||_2^2}{||M_{c\cdot K}^D||_1}. \tag{2}$$

Additionally, we adapt the reconstruction loss from DECOR-GAN to the multi-resolution setting: if both the input coarse shape and the geometry code stem from the same detailed style shape, we expect the outputs of the geometry generator to be the ground truth geometry at both the intermediate and the final resolutions.

$$\mathcal{L}_K^{recon} = \mathop{\mathbb{E}}_{s\sim\mathcal{S}} \frac{||G_K^{geo}(s_{\downarrow}^{geo},\, z_s^{geo}) \circ M_{s\cdot K}^G - s^{geo}||_2^2}{|s^{geo}|}, \tag{3}$$

where $|s^{geo}|$ is the volume (height $\times$ width $\times$ depth) of the voxels in s^{geo}, and $s_{\downarrow}^{geo}$ is the coarse content shape downsampled from s^{geo}.

The overall loss for geometry detailization is the sum of the GAN loss and the reconstruction loss at each resolution:

$$\mathcal{L}_{geo} = \mathcal{L}_K^G + \mathcal{L}_{K/2}^G + \mathcal{L}_K^{recon} + \mathcal{L}_{K/2}^{recon}. \tag{4}$$

3.2 Texture generation via differentiable rendering

Similar to our geometry upsampling pipeline, a naïve way to synthesize textures on upsampled shapes is to define them as volumetric textures and directly apply 3D GANs on a grid of RGB values. However, as discussed in our ablation study in Section 4.2, this design does not fully utilize the discriminator's capacity, as the discriminator has to account for colors of voxels in the ambient space, which are ill-defined. Indeed, textures are 2D in nature, and only voxels on the surfaces of the shapes have well-defined colors. Therefore, given the upsampled geometry, we first apply differentiable rendering to project the colors of the surface voxels onto a 2D image, and then employ 2D GANs for learning texture synthesis.

As shown in Figure 2 right, a coarse content shape at resolution k^3 and a latent texture code representing the texture style of a training style shape are given as input to the texture generator, and the generator synthesizes a K^3 color grid of RGB values. We then use the geometry generator to upsample the same content shape with the geometric style of the same style shape into a detailed geometry of resolution K^3. The detailed geometry, combined with the volumetric textures stored in the color grid, are rendered into 2D images from different views (up to five: top, back, front, and sides). The rendered images are then fed into several distinct 2D CNN PatchGAN discriminators corresponding to different views for adversarial training. Similar to geometry detailization, we use an embedding module to learn an 8-dimensional latent texture code for each style shape during training. More details about the differentiable rendering step can be found in the supplementary.

Training losses. We denote the texture code representing the style of s as z_s^{tex}. The color grid of s is denoted as s^{tex}. The texture generator is denoted by G^{tex} and the texture discriminator of view i is denoted by D_i^{tex}. We also reuse the notations defined in Section 3.1. The binary discriminator masks of shape s and c from view i are denoted as $M_{s\cdot i}^D$ and $M_{c\cdot i}^D$, respectively. Similar to the geometry masks, these texture masks are used to focus the networks' capacities on non-empty regions in the rendered images; see supplementary for detailed definitions. The discriminator loss is defined as:

$$\mathcal{L}_i^D = \mathop{\mathbb{E}}_{s\sim\mathcal{S}} \frac{||(D_i^{tex}(R_i(s^{geo}, s^{tex})) - 1) \circ M_{s\cdot i}^D||_2^2}{||M_{s\cdot i}^D||_1} + \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{S}\\ c\sim C}} \frac{||D_i^{tex}(R_i(c_{s\cdot K}^{geo}, c_{s\cdot K}^{tex}) \circ M_{c\cdot i}^D||_2^2}{||M_{c\cdot i}^D||_1},$$

$$c_{s\cdot K}^{tex} = G^{tex}(c, z_s^{tex}), \tag{5}$$

where $c_{s\cdot K}^{tex}$ is the synthesized color grid, and $R_i(\cdot,\cdot)$ is the rendering function at view i given the geometry voxels and the color grid. The generator loss is defined as:

$$\mathcal{L}_i^G = \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{S}\\ c\sim C}} \frac{||(D_i^{tex}(R_i(c_{s\cdot K}^{geo}, c_{s\cdot K}^{tex})) - 1) \circ M_{c\cdot i}^D||_2^2}{||M_{c\cdot i}^D||_1}. \tag{6}$$

For the reconstruction loss, we expect the rendered images of each view to be the ground truth images if the input coarse shape and the texture code stem from the same detailed style shape:

$$\mathcal{L}_i^{recon} = \mathop{\mathbb{E}}_{s\sim\mathcal{S}} \frac{||R_i(s_{s\cdot K}^{geo}, s_{s\cdot K}^{tex}) - R_i(s^{geo}, s^{tex})||_2^2}{|R_i(s^{geo}, s^{tex})|},$$

$$s_{s\cdot K}^{geo} = G_K^{geo}(s_{\downarrow}^{geo}, z_s^{geo}), \quad s_{s\cdot K}^{tex} = G^{tex}(s_{\downarrow}^{geo}, z_s^{tex}), \tag{7}$$

where $|R_i(s^{geo}, s^{tex})|$ is the area (height × width) of the rendered image. $s_{s\cdot K}^{geo}$ and $s_{s\cdot K}^{tex}$ are the upsampled geometry and synthesized texture from downsampled s^{geo}, respectively.

The overall loss for texture generation is the sum of the GAN loss and the reconstruction loss at each view.

$$\mathcal{L}_{tex} = \sum_i (\mathcal{L}_i^G + \mathcal{L}_i^{recon}). \tag{8}$$

3.3 Implementation details

Our generators are designed to upsample the geometry 8 times, i.e., $K/k = 8$. In our experiments, we train individual models for different shape categories. For each category, we train the geometry generator first, and then train the texture generator while fixing the weights of the geometry generator. We assume the training shapes from some categories (*car*, *airplane*, and *chair*) are bilaterally symmetrical, therefore only generating half of the shape. We do not make symmetrical assumption for the *building* category, thus generating the whole shape. Depending on the category, training each model for geometry detailization and texture generation on a single NVIDIA RTX 3090 Ti GPU takes 12-24 hours and 18-36 hours, respectively. The network inference takes less than 1 second per content per style. The network architectures and training details can be found in the supplementary material.

3.4 Data preparation

We follow DECOR-GAN [Chen et al. 2021] to prepare occupancy voxels of 512^3 resolution for each shape, which can be downsampled to K^3 and k^3 resolutions as style shapes and content shapes for training. For detailed style shapes, we apply a Gaussian filter with $\sigma = 1.0$ on the geometry voxels to encourage continuous optimization in GANs. To generate colored voxels for detailed style shapes, we first render the shapes into images from different views, and project the pixel colors in the rendered images back to the surface voxels. We inpaint the colors of non-surface voxels using nearest neighbor to obtain volumetric textures for our ablation study. More details can be found in the supplementary material.

4 RESULTS AND EVALUATION

We conduct a series of experiments on different categories to demonstrate the effectiveness and generalizability of our method, and validate our design decisions in ablation studies.

Dataset. We test our method on four categories: cars, airplanes, chairs from ShapeNet [Chang et al. 2015], and buildings from Houses3K [Peralta et al. 2020]. We use 3,141 cars, 1,743 airplanes, and 2,824 chairs as coarse content shapes, same as DECOR-GAN. We augment the 600 buildings from Houses3K with 90° rotations into 2,400 shapes as content shapes. We split the content shapes by 80%/20% as the training and testing set. We select 8 style shapes for the building category and 16 style shapes for other categories with various topologies and textures. We use 64^3 coarse content shapes for cars and airplanes, and 32^3 for chairs and buildings to further remove the topological details. We extract the surface using Marching Cubes [Lorensen and Cline 1987], and determine the vertex colors of the output mesh by querying the synthesized color grid.

Metrics. We focus the evaluation on texture synthesis, as it is the primary contribution of our work compared to DECOR-GAN. Ideally, the generated textures should be globally plausible and locally similar to the ground truth texture providing the style. To quantitatively measure the quality of the textures, we render images of the generated shapes and the style shapes from different views and evaluate with Fréchet Inception Distance (FID) [Heusel et al. 2017] and Learned Perceptual Image Patch Similarity (LPIPS) [Zhang et al. 2018] metrics. We use FID-all to measure the similarity between the generated textures and all ground truth textures used in our training, and FID-style to measure the similarity between the generated textures and the ground truth texture that provides the style of the generated textures. Similar to FID-style, we make use of LPIPS-style to evaluate the local patch similarity between generated textures and the ground truth textures of each style. Lower FID-all, FID-style, and LPIPS-style indicate higher quality of the generated textures.

4.1 Geometry detailization and texture generation

The visual results of the car, airplane, chair, and building categories can be found in Figures 9, 10, 11, and 12, respectively, with more results in the supplementary. Our method is able to not only produce high-resolution upsampled geometry with fine local details, e.g., the car wheels, but also synthesize high-quality texture colors, e.g., stripes pattern of the car and the windows of the building.

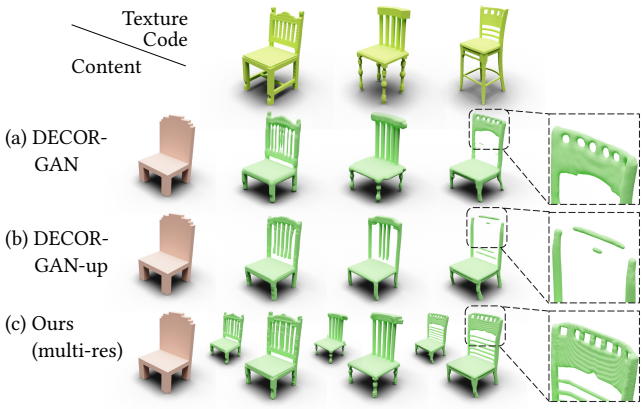


Fig. 4. Comparison of geometry detailization results between DECOR-GAN ($K = 128$), DECOR-GAN-up ($K = 256$), and our multi-resolution setting in ShaDDR ($K = 256$, small figures show the intermediate voxels at 128 resolution). Please zoom in to observe the local geometric details.

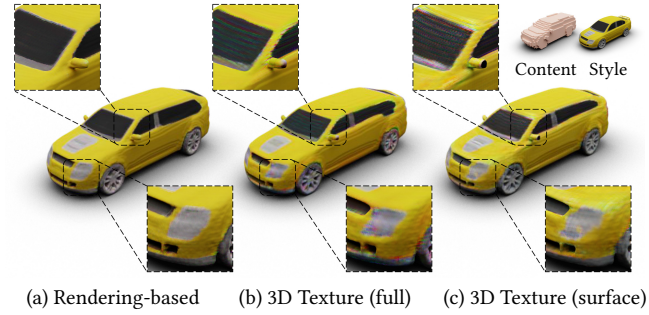


Fig. 5. Comparison of texture generation results between our rendering-based approach and two baselines. Our approach can produce textures that are cleaner and have better alignment with the geometry.

4.2 Ablation study

Geometry detailization w/o vs. w/ multi-resolution setting. We compare our multi-resolution ShaDDR with the original DECOR-GAN [Chen et al. 2021], which generates single-resolution geometries. We lift the DECOR-GAN upsampling resolution, denoted as DECOR-GAN-up, to match ours for a fair comparison. Note that our multi-resolution setting has nearly the same network architectures as DECOR-GAN-up except that ours has one more layer for generating geometry at an *intermediate* resolution.

Figure 4 shows a qualitative comparison between the geometries generated by DECOR-GAN, DECOR-GAN-up, and ours, which reveals a general trend. Note that the original DECOR-GAN [Chen et al. 2021] upsamples the geometry by 4 times, thus unable to provide sufficient local details, e.g., sharp edges shown in Figure 4 (a). DECOR-GAN-up directly lifts the upsampling rate twice, thus the geometry generator has to learn a more complicated upsampling space where the upsampled geometry might be locally distinctive but globally implausible. As a result, it fails to generate complete geometric structures even though the local structures may be able to follow the style shapes, but only partially, as shown in Figure 4 (b). With our multi-resolution setting, both global structures and local details of the upsampled geometry are already better than DECOR-GAN and DECOR-GAN-up at the intermediate resolution, while detailed geometric features, e.g., sharp edges and smooth surfaces, are refined at the final resolution; see Figure 4 (c). More comparison results can be found in the supplementary.

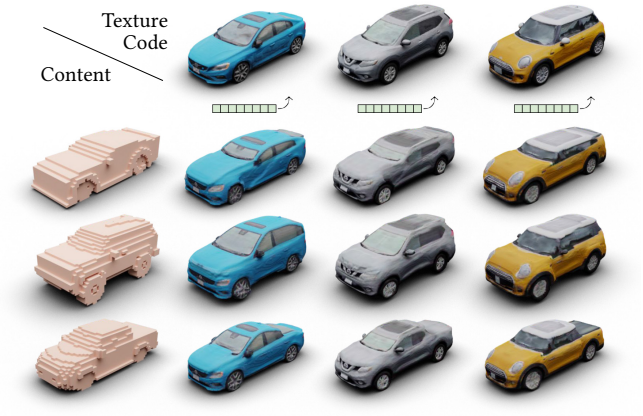


Fig. 6. Results of geometry detailization and texture generation conditioned on cars that are reconstructed from real-world photos. We show the input coarse content voxels on the left and the detailed style shapes with textures on top. The coarse content voxels are 64^3 and the generated shapes are 512^3.

Learning textures via 3D vs. 2D supervision. Since we generate a 3D grid of colors to represent textures, it is possible to treat the color grid as volumetric texture and supervise the learning with a 3D texture discriminator, similar to our 3D geometry discriminator. We denote this setting as *3D texture (full)*. It is worth noting that texture is a surface property and learning colors of voxels inside or outside the shape is intrinsically ill-posed. Therefore, we modify the 3D texture discriminator to only learn the voxel colors near the geometry surface by applying texture discriminator masks whose values are 1 near the surface and 0 elsewhere, similar to $M^D_{s\cdot K}$ and $M^D_{c\cdot K}$. We denote this setting as *3D texture (surface)*. For a fair comparison, we use the same generator architecture for all settings and only replace the 2D convolutions in our texture discriminator with 3D convolutions to be used in those baselines.

A qualitative comparison between our rendering-based approach and the two baselines are shown in Figure 5. Note that even though both 3D texture (full) and 3D texture (surface) can generate reasonable textures, with a closer look at the local regions, e.g. windshield, window frame, and headlight, textures synthesized by our rendering-based method are cleaner and sharper compared to the two baselines whose generated textures are stained and uneven. We also provide a quantitative comparison in Table 1. Rendering-based approach outperforms the two baselines in all metrics except FID-all in chair category. Additionally, we observe that metrics are usually higher in the building category than those in other categories due to the considerable geometric variations of the content shapes.

Table 1. Quantitative evaluation of different texture generation approaches.

	FID-all ↓				FID-style ↓				LPIPS-style ↓			
	Car	Airplane	Chair	Building	Car	Airplane	Chair	Building	Car	Airplane	Chair	Building
3D texture (full)	51.094	19.438	46.434	127.834	104.384	58.199	106.707	192.059	0.121	0.111	0.274	0.396
3D texture (surface)	51.519	23.438	**45.727**	128.227	105.278	61.410	104.850	194.275	0.115	0.113	0.276	0.398
Rendering-based	**41.573**	**13.726**	45.962	**124.618**	**88.325**	**43.418**	**104.735**	**189.071**	**0.104**	**0.101**	**0.270**	**0.386**

(a) Multi-view images (b) Reconstruction (c) Post-processing

Fig. 7. We take multiple photos of a real car and reconstruct a textured mesh using NeuS [Wang et al. 2021]. The mesh is post-processed to remove the ground and close holes.

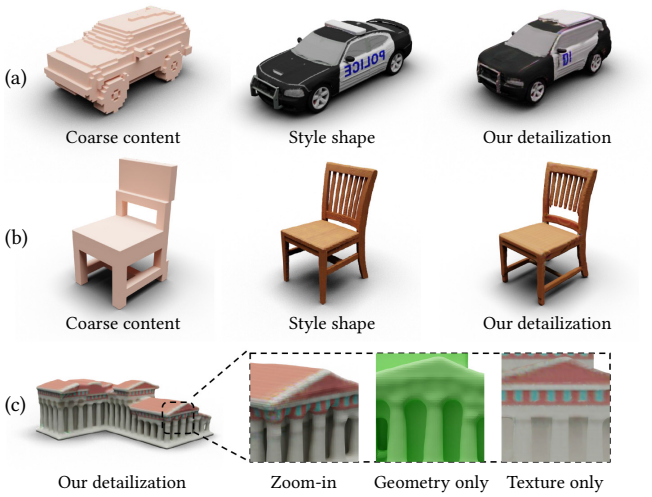


Fig. 8. Limitations. See Figure 1 for the geometry and texture styles of (c).

4.3 Application

Styles from real photos. We show that our method is able to generate geometry and realistic textures by learning from style shapes reconstructed from real-world images. We capture multi-view images from real cars in the wild, as shown in Figure 7 (a). Then we use NeuS [Wang et al. 2021] to reconstruct both the geometry and the texture, as shown in Figure 7 (b). Finally, we post-process the reconstructed mesh by manually removing the ground and closing holes, as shown in Figure 7 (c). We then follow Section 3.4 to prepare the style shapes for training. Figure 16 shows the detailization results.

Generalizability and interactive editing. We show that our method is capable of handling various coarse input shapes that are manually designed by users without further training or fine-tuning. Thanks to the inductive bias of 3D convolutions, i.e., convolutional operations are local and translation equivariant, our texture generator has strong generalizability to new shapes and is able to handle areas that are fully invisible in rendered views during training. Moreover, our model only needs a single forward pass to detailize a new shape, which takes less than a second, thus allowing our method to be integrated into interactive modeling tools. We have created an interactive modeling interface where users can edit a coarse content voxel, choose a style, and visualize the detailed textured shape in real time. Figure 3 shows the interactive editing process and the detailization results. We also provide a video recording to show the entire editing process in the supplementary.

5 CONCLUSIONS

We present ShaDDR, an example-based deep generative network that produces a high-resolution textured 3D shape through geometry detailization and conditional texture generation. ShaDDR is capable of upsampling a coarse voxel model into a fully textured and geometrically detailed 3D shape, whose results show improved geometric details and clean textures compared to prior works. Extensive experiments demonstrate the capability of ShaDDR to generate novel detailed textured shapes, whether trained on synthetic or real data. The interactive modeling interface powered by ShaDDR allows users to edit the coarse content voxel model and visualize the detailed textured shape in real time.

Limitations. As our texture generation is designed to only enforce the plausibility of local patches, the generated textures may lack global structure awareness. For instance, the detailized car in Figure 8 (a) does not possess the same blue letters as those in the style shape, which can be explained by considering that both blue letters and plain white are plausible textures presented in the style shape. Similarly, our geometry detailization relies solely on generating plausible local patches, which may lead to global inconsistencies in the output shapes. An example is shown in Figure 8 (b), where a portion of the back is carved out in the detailized chair to respect the structure of the content shape, leaving inconsistent edges. In addition, our method does not have an explicit mechanism to enforce the alignment between the generated textures and the underlying geometry, thus it may lead to geometry-texture misalignment, as in Figure 8 (c). However, this phenomenon is surprisingly rare in our experiments, which may be attributed to our texture discriminators ensuring multi-view plausibility of the synthesized textures.

Future works. Aside from addressing the limitations above, we are interested in learning both geometry detailization and texture generation without any 3D supervision, possibly with differentiable volumetric rendering. Introducing local control is another interesting direction, e.g., allowing users to specify arbitrary regions of the coarse content shape and detailize them into designated styles.

REFERENCES

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. 2018. Learning representations and generative models for 3D point clouds. In *ICLR*. 40–49.

Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. 2022. Efficient geometry-aware 3D generative adversarial networks. In *CVPR*. 16123–16133.

Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. 2021. pi-GAN: Periodic implicit generative adversarial networks for 3D-aware image synthesis. In *CVPR*. 5799–5809.

Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. 2015. ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012* (2015).

Bindita Chaudhuri, Nikolaos Sarafianos, Linda Shapiro, and Tony Tung. 2021. Semi-supervised synthesis of high-resolution editable textures for 3D humans. In *CVPR*. 7991–8000.

Kevin Chen, Christopher B Choy, Manolis Savva, Angel X Chang, Thomas Funkhouser, and Silvio Savarese. 2019a. Text2Shape: Generating shapes from natural language by learning joint embeddings. In *ACCV*. 100–116.

Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. 2019b. Learning to predict 3D objects with an interpolation-based differentiable renderer. In *NeurIPS*.

Xiaobai Chen, Tom Funkhouser, Dan B Goldman, and Eli Shechtman. 2012. Non-parametric texture transfer using meshmatch. *Technical Report* (2012).

Zhiqin Chen, Vladimir G. Kim, Matthew Fisher, Noam Aigerman, Hao Zhang, and Siddhartha Chaudhuri. 2021. DECOR-GAN: 3D Shape detailization by conditional refinement. In *CVPR*. 15740–15749.

Zhiqin Chen, Kangxue Yin, and Sanja Fidler. 2022. AUV-Net: Learning aligned UV maps for texture transfer and synthesis. In *CVPR*. 1465–1474.

Zhiqin Chen and Hao Zhang. 2019. Learning implicit fields for generative shape modeling. In *CVPR*. 5939–5948.

Yen-Chi Cheng, Hsin-Ying Lee, Sergey Tuyakov, Alex Schwing, and Liangyan Gui. 2023. SDFusion: Multimodal 3D shape completion, reconstruction, and generation. In *CVPR*.

Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 2016. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *ECCV*. 628–644.

George R Cross and Anil K Jain. 1983. Markov random field texture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1 (1983), 25–39.

Alexei A Efros and William T Freeman. 2001. Image quilting for texture synthesis and transfer. In *Proceedings of Annual Conference on Computer Graphics and Interactive Techniques*. 341–346.

Alexei A Efros and Thomas K Leung. 1999. Texture synthesis by non-parametric sampling. In *ICCV*, Vol. 2. 1033–1038.

Haoqiang Fan, Hao Su, and Leonidas J Guibas. 2017. A point set generation network for 3D object reconstruction from a single image. In *CVPR*. 605–613.

Noa Fish, Lilach Perry, Amit Bermano, and Daniel Cohen-Or. 2020. SketchPatch: Sketch stylization via seamless patch-level synthesis. *ACM Transactions on Graphics* 39, 6 (2020), 1–14.

Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. 2022. GET3D: A generative model of high quality 3D textured shapes learned from images. In *NeurIPS*.

Lin Gao, Tong Wu, Yu-Jie Yuan, Ming-Xian Lin, Yu-Kun Lai, and Hao Zhang. 2021. TM-NET: Deep generative networks for textured meshes. *ACM Transactions on Graphics* 40, 6 (2021), 1–15.

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

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. 2018. A papier-mâché approach to learning 3D surface generation. In *CVPR*. 216–224.

Christian Häne, Shubham Tulsiani, and Jitendra Malik. 2017. Hierarchical surface prediction for 3D object reconstruction. In *3DV*. 412–420.

Paul Henderson, Vagia Tsiminaki, and Christoph H Lampert. 2020. Leveraging 2D data to learn textured 3D mesh generation. In *CVPR*. 7498–7507.

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

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *NeurIPS* 33 (2020), 6840–6851.

Ka-Hei Hui, Ruihui Li, Jingyu Hu, and Chi-Wing Fu. 2022. Neural wavelet-domain diffusion for 3D shape generation. In *SIGGRAPH Asia Conference Papers*. 1–9.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. In *CVPR*. 1125–1134.

Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. 2022. 3inGAN: Learning a 3D Generative Model from Images of a Self-similar Scene. In *3DV*.

Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. 2007. Solid texture synthesis from 2D exemplars. In *ACM SIGGRAPH 2007 papers*. 2–es.

Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. 2017. GRASS: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics* 36, 4 (2017), 1–14.

Xueting Li, Sifei Liu, Kihwan Kim, Shalini De Mello, Varun Jampani, Ming-Hsuan Yang, and Jan Kautz. 2020. Self-supervised single-view 3D reconstruction via semantic consistency. In *ECCV*. 677–693.

Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. 2023. Magic3D: High-Resolution Text-to-3D Content Creation. In *CVPR*.

William E. Lorensen and Harvey E. Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *SIGGRAPH*.

Luke Melas-Kyriazi, Christian Rupprecht, Iro Laina, and Andrea Vedaldi. 2023. RealFusion: 360 reconstruction of any object from a single image. In *CVPR*.

Paul Merrell. 2007. Example-based model synthesis. In *Proceedings of Interactive 3D Graphics and Games*. 105–112.

Paul Merrell and Dinesh Manocha. 2008. Continuous model synthesis. In *ACM SIGGRAPH Asia 2008 papers*. 1–7.

Paul Merrell and Dinesh Manocha. 2010. Model synthesis: A general procedural modeling algorithm. *IEEE Transactions on Visualization and Computer Graphics* 17, 6 (2010), 715–728.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy networks: Learning 3D reconstruction in function space. In *CVPR*. 4460–4470.

Oscar Michel, Roi Bar-On, Richard Liu, Sagie Benaim, and Rana Hanocka. 2022. Text2Mesh: Text-driven neural stylization for meshes. In *CVPR*. 13492–13502.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. NeRF: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106.

Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas J Guibas. 2019. StructureNet: Hierarchical graph networks for 3D shape generation. *ACM Transactions on Graphics* 38, 6 (2019), 1–19.

Nasir Mohammad Khalid, Tianhao Xie, Eugene Belilovsky, and Tiberiu Popa. 2022. CLIP-Mesh: Generating textured meshes from text using pretrained image-text models. In *SIGGRAPH Asia Conference Papers*. 1–8.

Tom Monnier, Matthew Fisher, Alexei A. Efros, and Mathieu Aubry. 2022. Share With Thy Neighbors: Single-view reconstruction by cross-instance consistency. In *ECCV*.

Michael Niemeyer and Andreas Geiger. 2021. GIRAFFE: Representing scenes as compositional generative neural feature fields. In *CVPR*. 11453–11464.

Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. 2019. Texture Fields: Learning texture representations in function space. In *ICCV*. 4531–4540.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*. 165–174.

Dario Pavllo, Jonas Kohler, Thomas Hofmann, and Aurelien Lucchi. 2021. Learning generative models of textured 3D meshes from real-world images. In *ICCV*. 13879–13889.

Dario Pavllo, Graham Spinks, Thomas Hofmann, Marie-Francine Moens, and Aurelien Lucchi. 2020. Convolutional generation of textured 3D meshes. *NeurIPS* 33 (2020), 870–882.

Daryl Peralta, Joel Casimiro, Aldrin Michael Nilles, Justine Aletta Aguilar, Rowel Atienza, and Rhandley Cajote. 2020. Next-best view policy for 3D reconstruction. In *ECCV Workshop*. 558–573.

Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. 2022. DreamFusion: Text-to-3D using 2D diffusion. *arXiv preprint arXiv:2209.14988* (2022).

Amit Raj, Cusuh Ham, Connelly Barnes, Vladimir Kim, Jingwan Lu, and James Hays. 2019. Learning to generate textures on 3D meshes. In *CVPR Workshops*. 32–38.

Daniel Rebain, Mark Matthews, Kwang Moo Yi, Dmitry Lagun, and Andrea Tagliasacchi. 2022. LOLNeRF: Learn from one look. In *CVPR*. 1558–1567.

Danilo Rezende and Shakir Mohamed. 2015. Variational inference with normalizing flows. In *ICLR*. 1530–1538.

Elad Richardson, Gal Metzer, Yuval Alaluf, Raja Giryes, and Daniel Cohen-Or. 2023. TEXTure: Text-guided texturing of 3D shapes. In *SIGGRAPH*.

Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. 2020. GRAF: Generative radiance fields for 3D-aware image synthesis. *NeurIPS* 33 (2020), 20154–20166.

Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. 2019. SinGAN: Learning a generative model from a single natural image. In *ICCV*. 4570–4580.

Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. 2021. Deep Marching Tetrahedra: a hybrid representation for high-resolution 3D shape synthesis. In *NeurIPS*.

Yawar Siddiqui, Justus Thies, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. 2022. Texturify: Generating Textures on 3D Shape Surfaces. In *ECCV*. 72–88.

Ivan Skorokhodov, Sergey Tulyakov, Yiqun Wang, and Peter Wonka. 2022. EpiGRAF: Rethinking training of 3D GANs. In *NeurIPS*.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*. PMLR, 2256–2265.

Junshu Tang, Tengfei Wang, Bo Zhang, Ting Zhang, Ran Yi, Lizhuang Ma, and Dong Chen. 2023. Make-It-3D: High-Fidelity 3D Creation from A Single Image with Diffusion Prior. arXiv:2303.14184 [cs.CV]

Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. 2016. Conditional image generation with PixelCNN decoders. *NeurIPS* 29 (2016).

Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. 2018. Pixel2Mesh: Generating 3D mesh models from single RGB images. In *ECCV*. 52–67.

Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. NeuS: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In *NeurIPS*.

Yujie Wang, Xuelin Chen, and Baoquan Chen. 2022. SinGRAV: Learning a generative radiance volume from a single natural scene. *arXiv preprint arXiv:2210.01202* (2022).

Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. 2016. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. *NeurIPS* 29 (2016).

Rundi Wu and Changxi Zheng. 2022. Learning to generate 3D shapes from a single example. *ACM Transactions on Graphics* 41, 6 (2022), 1–19.

Kangxue Yin, Zhiqin Chen, Hui Huang, Daniel Cohen-Or, and Hao Zhang. 2019. LOGAN: Unpaired shape transform in latent overcomplete space. *ACM Transactions on Graphics* 38, 6 (2019), 1–13.

Kangxue Yin, Jun Gao, Maria Shugrina, Sameh Khamis, and Sanja Fidler. 2021. 3DStyleNet: Creating 3D shapes with geometric and texture style variations. In *ICCV*. 12456–12465.

Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. 2021. pixelNeRF: Neural Radiance Fields from One or Few Images. In *CVPR*. 4578–4587.

Xiaohui Zeng, Arash Vahdat, Francis Williams, Zan Gojcic, Or Litany, Sanja Fidler, and Karsten Kreis. 2022. LION: Latent point diffusion models for 3D shape generation. In *NeurIPS*.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*. 586–595.

Yuxuan Zhang, Wenzheng Chen, Huan Ling, Jun Gao, Yinan Zhang, Antonio Torralba, and Sanja Fidler. 2020. Image GANs meet differentiable rendering for inverse graphics and interpretable 3D neural rendering. In *ICLR*.

Yuxuan Zhang, Wenzheng Chen, Huan Ling, Jun Gao, Yinan Zhang, Antonio Torralba, and Sanja Fidler. 2021. Image GANs meet Differentiable Rendering for Inverse Graphics and Interpretable 3D Neural Rendering. In *International Conference on Learning Representations*.

Kun Zhou, Xin Huang, Xi Wang, Yiying Tong, Mathieu Desbrun, Baining Guo, and Heung-Yeung Shum. 2006. Mesh quilting for geometric texture synthesis. In *ACM SIGGRAPH 2006 Papers*. 690–697.

Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. 2018. Non-stationary texture synthesis by adversarial expansion. *ACM Transactions on Graphics* 37, 4 (2018), 1–13.

Fig. 9. Results of geometry detailization and texture generation on the car category. We show the input coarse content voxels on the left and the detailed style shapes with textures on top. The coarse content voxels are 64^3 and the generated shapes are 512^3.

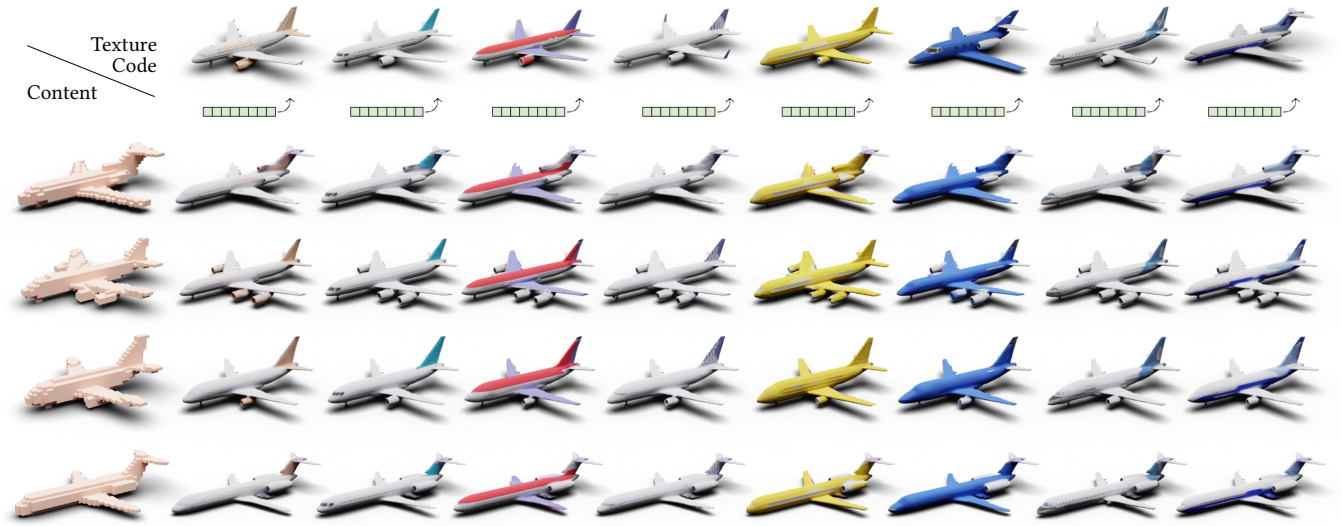


Fig. 10. Results of geometry detailization and texture generation on the airplane category. We show the input coarse content voxels on the left and the detailed style shapes with textures on top. The coarse content voxels are 64^3 and the generated shapes are 512^3.

Fig. 11. Results of geometry detailization and texture generation on the chair category. We show the input coarse content voxels on the left and the detailed style shapes with textures on top. The coarse content voxels are 32^3 and the generated shapes are 256^3.

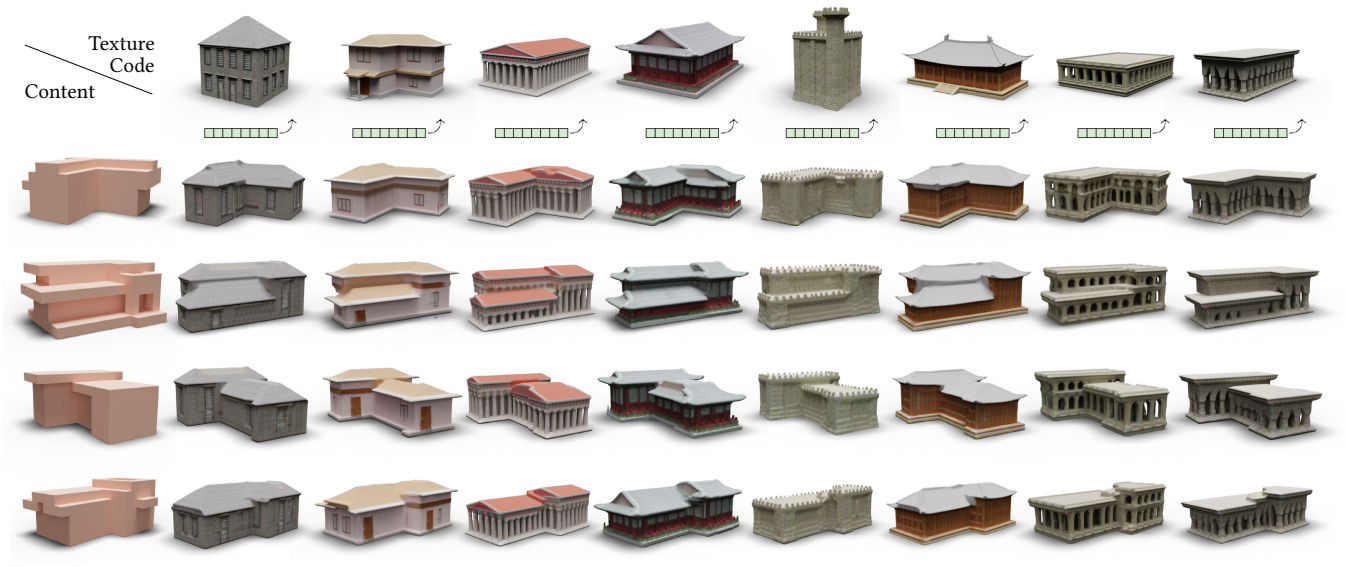


Fig. 12. Results of geometry detailization and texture generation on the building category. We show the input coarse content voxels on the left and the detailed style shapes with textures on top. The coarse content voxels are 32^3 and the generated shapes are 256^3.

In this supplementary material, we provide details regarding network architectures, loss functions, implementation settings and data preparation. We also provide additional qualitative results.

6 NETWORK ARCHITECTURE

We provide detailed network architecture in Figure 13.

Geometry generator. The geometry generator consists of a backbone network and an upsampling network. The backbone network consists of 5 layers of 3D convolution followed by leaky ReLU. The output of each convolution layer is concatenated with an 8-dimensional trainable module representing the geometry style. The upsampling network consists of 3 upsampling layers. Each upsampling layer is followed by a 3D convolution and doubles the input resolution. The output of the first two upsampling layers is concatenated with the same 8-dimensional trainable module as well. Note that the second upsampling layer is followed by an output layer to output the upsampled shape at intermediate resolution.

Texture generator. Similar to the geometry generator, the texture generator consists of a backbone network and an upsampling network. The backbone network consists of 5 layers of 3D convolution followed by leaky ReLU. The output of each convolution layer is concatenated with an 8-dimensional *pre-trained* module representing the geometry style learned during geometry detailization. The upsampling network consists of 3 upsampling layers. Each upsampling layer is followed by a 3D convolution and doubles the input resolution. The output of the first two upsampling layers is concatenated with an 8-dimensional trainable module representing the texture style.

Geometry discriminator. The geometry discriminator consists of 5 layers of *3D* convolution followed by leaky ReLU and a layer of 3D convolution followed by Sigmoid. The receptive fields of the geometry discriminator are adjusted according to the category. We use receptive field $36 \times 36 \times 36$ for car and plane and $18 \times 18 \times 18$ for chair and building.

Texture discriminator. The geometry discriminator consists of 5 layers of *2D* convolution followed by leaky ReLU and a layer of 3D convolution followed by Sigmoid. The receptive fields of the texture discriminator are adjusted according to the category. We use receptive field 36×36 for car and plane and 18×18 for chair and building.

7 LOSS FUNCTION

Like DECOR-GAN, We provide a detailed explanation of annotations defined in Equations (1), (2), (3), (4), (5), (6), (7), (8).

G_K^{geo}. The geometry generator that outputs the geometry voxel of resolution K^3.

D_K^{geo}. The geometry discriminator that determines the patches of the geometry voxel at the resolution K^3. More specifically, ${}^{global}D_K^{geo}$ denotes the global branch of the geometry discriminator at the resolution K^3 and ${}^{style}D_K^{geo}$ denotes the style branch of the geometry discriminator at the resolution K^3.

G^{tex}. The texture generator that outputs the volumetric texture of resolution K^3.

D_i^{tex}. The texture discriminator that determines the patches of the rendered image at view i. More specifically, ${}^{global}D_i^{tex}$ denotes the global branch of the texture discriminator at view i and ${}^{style}D_i^{tex}$ denotes the style branch of the texture discriminator at view i.

$M_{c\cdot K}^G$. The generator mask for content shape. This mask ensures that the empty voxel in the input coarse content shape remains empty in the output upsampled shape and allows the generator to focus on generating voxels in the valid region. It is computed by dilating the coarse input content and upsampling it by 8 times.

$M_{s\cdot K}^G$. Similar to $M_{c\cdot K}^G$. It is computed by first dilating the downsampled style shape and upsampling it by 8 times.

$M_{c\cdot K}^D$. The discriminator mask for upsampled shape. This mask ensures that the occupied voxel in the input coarse content shape should lead to the creation of upsampled voxel in its corresponding area. Empty voxels in the region of interest should be punished by the discriminator. It is computed by upsampling the coarse content shape to match the dimension of the discriminator output. It is defined in 2D space for texture generation.

$M_{s\cdot K}^D$. Similar to $M_{c\cdot K}^D$. It is computed by upsampling the downsampled style shape to match the dimension of the discriminator output. It is defined in 2D space for texture generation.

$c_{s\cdot K}^{geo}$. The output of the geometry generator. This is the upsampled shape to be discriminated by the geometry discriminator. It is also used for rendering texture images.

$c_{s\cdot K}^{tex}$. The output of the texture generator. This is the generated texture voxel to be used for rendering texture images.

$R_i(c_{s\cdot K}^{geo}, c_{s\cdot K}^{tex})$. The rendering function for view i. The rendering function takes upsampled geometry voxel and generated texture voxel as inputs. The 2D masks and depth maps of different views are computed and the rendered images of each view are gathered from the texture voxel. Take the front view for an example, it first calculates the mask (maximum value) and depth (index of the maximum value) by computing the argmax along the first axis of the upsampled geometry voxel, the depth value is then used to gather the color value from the generated texture voxel along the first axis. The mask is then multiplied to mask out the non-region of interest.

Detailed loss functions. We use the same idea as DECOR-GAN to prevent model collapse by splitting the discriminator into $N+1$ branches at the output layer, where N is the number of detailed shapes and an additional 1 branch for the global discriminator accounting for all styles. Note that since both style branches and global branch use the exact same equation to compute loss, we omit the details in the main paper for simplicity and clarity. Here we describe the detailed version of loss functions (1), (2), (5) and (6) of the main paper.

For geometry generation, the discriminator loss consists of the global branch's loss ${}^{global}D_K^{geo}$ and the style branch's loss ${}^{style}D_K^{geo}$ at the resolution of $(K/2)^3$ and K^3, we define losses at the resolution of K^3 here, losses at the resolution of $(K/2)^3$ can be easily derived by changing the subscript:

$$\mathcal{L}_D^{geo} = \mathcal{L}_{D\cdot K}^{global} + \mathcal{L}_{D\cdot K}^{style} + \mathcal{L}_{D\cdot K/2}^{global} + \mathcal{L}_{D\cdot K/2}^{style} \tag{9}$$

where

$$\mathcal{L}^{global}_{D\cdot K} = \mathop{\mathbb{E}}_{s\sim\mathcal{S}} \frac{||(^{global}D^{geo}_K(s^{geo}) - 1) \circ M^D_{s\cdot K}||_2^2}{||M^D_{s\cdot K}||_1} + \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{S}\\ c\sim C}} \frac{||^{global}D^{geo}_K(c^{geo}_{s\cdot K}) \circ M^D_{c\cdot K}||_2^2}{||M^D_{c\cdot K}||_1},$$

$$c^{geo}_{s\cdot K} = G^{geo}_K(c,\, z^{geo}_s) \circ M^G_{c\cdot K} \tag{10}$$

$$\mathcal{L}^{style}_{D\cdot K} = \mathop{\mathbb{E}}_{s\sim\mathcal{S}} \frac{||(^{style}D^{geo}_K(s^{geo}) - 1) \circ M^D_{s\cdot K}||_2^2}{||M^D_{s\cdot K}||_1} + \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{S}\\ c\sim C}} \frac{||^{style}D^{geo}_K(c^{geo}_{s\cdot K}) \circ M^D_{c\cdot K}||_2^2}{||M^D_{c\cdot K}||_1} \tag{11}$$

and $\circ$ denotes element-wise multiplication, and $c^{geo}_{s\cdot K}$ is the upsampled shape of resolution K^3 from input coarse shape c with the style of s. The generator loss is defined as:

$$\mathcal{L}^{geo}_G = (\mathcal{L}^{global}_{G\cdot K} + \alpha\cdot\mathcal{L}^{style}_{G\cdot K}) + \gamma\cdot(\mathcal{L}^{global}_{G\cdot K/2} + \alpha\cdot\mathcal{L}^{style}_{G\cdot K/2}) \tag{12}$$

where

$$\mathcal{L}^{global}_{G\cdot K} = \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{S}\\ c\sim C}} \frac{||(^{global}D^{geo}_K(c^{geo}_{s\cdot K}) - 1) \circ M^D_{c\cdot K}||_2^2}{||M^D_{c\cdot K}||_1} \tag{13}$$

$$\mathcal{L}^{style}_{G\cdot K} = \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{S}\\ c\sim C}} \frac{||(^{style}D^{geo}_K(c^{geo}_{s\cdot K}) - 1) \circ M^D_{c\cdot K}||_2^2}{||M^D_{c\cdot K}||_1} \tag{14}$$

With the reconstruction loss described in the Section 3.1, the overall generator loss for geometry detailization is:

$$\mathcal{L}_{geo} = \mathcal{L}^{geo}_G + \beta\cdot\mathcal{L}^{recon}_K + \beta\cdot\mathcal{L}^{recon}_{K/2} \tag{15}$$

For texture generation, the discriminator loss consists of the global branch's loss $^{global}D^{tex}_i$ and the style branch's loss $^{style}D^{tex}_i$.

$$\mathcal{L}^{tex}_D = \sum_i (\mathcal{L}^{global}_{D\cdot i} + \mathcal{L}^{style}_{D\cdot i}) \tag{16}$$

where

$$\mathcal{L}^{global}_{D\cdot i} = \mathop{\mathbb{E}}_{s\sim\mathcal{S}} \frac{||(^{global}D^{tex}_i(R_i(s^{geo},\, s^{tex})) - 1) \circ M^D_{s\cdot i}||_2^2}{||M^D_{s\cdot i}||_1} + \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{S}\\ c\sim C}} \frac{||^{global}D^{tex}_i(R_i(c^{geo}_{s\cdot K},\, c^{tex}_{s\cdot K}) \circ M^D_{c\cdot i}||_2^2}{||M^D_{c\cdot i}||_1},$$

$$c^{tex}_{s\cdot K} = G^{tex}(c,\, z^{tex}_s) \tag{17}$$

$$\mathcal{L}^{style}_{D\cdot i} = \mathop{\mathbb{E}}_{s\sim\mathcal{S}} \frac{||(^{style}D^{tex}_i(R_i(s^{geo},\, s^{tex})) - 1) \circ M^D_{s\cdot i}||_2^2}{||M^D_{s\cdot i}||_1} + \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{S}\\ c\sim C}} \frac{||^{style}D^{tex}_i(R_i(c^{geo}_{s\cdot K},\, c^{tex}_{s\cdot K}) \circ M^D_{c\cdot i}||_2^2}{||M^D_{c\cdot i}||_1} \tag{18}$$

where $c^{tex}_{s\cdot K}$ is the synthesized color grid, and $R_i(\cdot,\cdot)$ is the rendering function at view i given the geometry voxels and the color grid. The generator loss is defined as:

$$\mathcal{L}^{tex}_{G\cdot i} = \mathcal{L}^{global}_{G\cdot i} + \gamma_1\cdot\mathcal{L}^{style}_{G\cdot i} \tag{19}$$

where

$$\mathcal{L}^{global}_{G\cdot i} = \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{S}\\ c\sim C}} \frac{||(^{global}D^{tex}_i(R_i(c^{geo}_{s\cdot K},\, c^{tex}_{s\cdot K})) - 1) \circ M^D_{c\cdot i}||_2^2}{||M^D_{c\cdot i}||_1} \tag{20}$$

$$\mathcal{L}^{style}_{G\cdot i} = \mathop{\mathbb{E}}_{\substack{s\sim\mathcal{S}\\ c\sim C}} \frac{||(^{style}D^{tex}_i(R_i(c^{geo}_{s\cdot K},\, c^{tex}_{s\cdot K})) - 1) \circ M^D_{c\cdot i}||_2^2}{||M^D_{c\cdot i}||_1} \tag{21}$$

With the reconstruction loss described in the section 3.2, the overall generator loss for texture generation is:

$$\mathcal{L}_{tex} = \sum_i (\mathcal{L}^{tex}_{G\cdot i} + \gamma_2\cdot\mathcal{L}^{recon}_i) \tag{22}$$

Empirically we found that using the same value of γ_1 and γ_2 as geometry detailization is enough to obtain good results.

8 IMPLEMENTATION DETAILS

We provide data preparation details and hyper-parameters used in all experiments.

Volumetric textures generation. In order to generate volumetric texture voxels for detailed style shapes, we first render images of resolution $K\times K\times 4$ from different views of the detailed style mesh, we then take the geometry voxel, approximate the normal direction of each occupied voxel grid, and record surface voxels. We determine the color of each surface voxel by finding the angle between the surface normal and horizontal or vertical plane and retrieving the pixel value from the corresponding view, e.g. if the angle between the surface normal of a surface voxel and the horizontal plane is less than $\pi/4$, the corresponding pixel value of the rendered image from right view is painted into that voxel.

Hyper-parameters. We set $\alpha = 1.0$ for chair, and $\alpha = 0.5$ for car, airplane and building. For training texture of the side view, we set $\beta = 5.0$ for car and building, $\beta = 10.0$ for airplane, and $\beta = 1.0$ for chair. We set $\beta = 1.0$ for the rest of the views. We set $\gamma = 0.5$ in Equation (4) of this supplementary material. We set the batch size to 1 and the learning rate to 0.0001 for all experiments. We train individual models for different categories. We train both the geometry detailization and the texture generation for 20 epochs on a single Nvidia GeForce RTX 3090 Ti. Depending on the category, training each model on the geometry detailization and the texture generation takes 12-24 hours and 18-36 hours, respectively.

Cropping. To handle the large memory footprint and speed up training, we crop each shape and discard the unoccupied voxel according to its dilated bounding box. For each detailed style shape, we crop the geometry voxel and the texture voxel.

9 APPLICATION

We create a GUI application where users can edit a given coarse content voxel template, including adding or removing voxel cell, choose a style shape and visualize the detailed textured shape in real time. The whole pipeline works as follow:

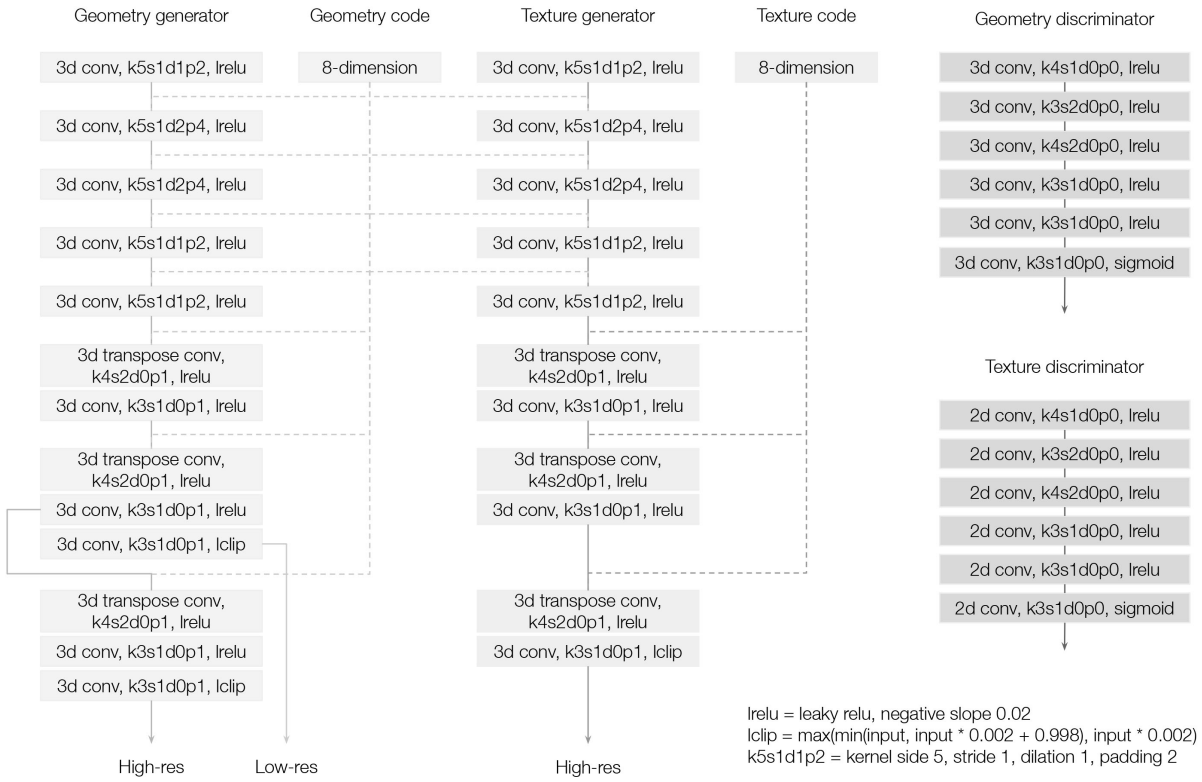


Fig. 13. Detailed network architecture of the geommetry generator, the texture generator, the geometry discriminator and the texture discriminator.

(1) User edits the coarse content voxel.
(2) Once user finishes editing, the edited coarse content voxel is passed to the pre-trained model for geometry detailization and texture generation.
(3) Marching Cubes is performed on the upsampled geometry to extract the surface and texture colors are gathered and assigned to each vertex.
(4) The detailed textured shape is visualized on canvas.

We provide instructions on how to use the modeling interface:

- Left-click on the computer mouse on the coarse voxel to add voxel(s).
- Left-click on the empty space to rotate the camera view.
- Right-click on the computer mouse on the coarse voxel to remove voxel(s).
- Right-click on the empty space to move the coarse voxel.
- Middle-click and scroll the scroll wheel on the computer mouse to zoom in/out.
- "Q", "W", "E", "R" for brush sizes "1", "3", "5", "7".
- Num pad 1-8 for choosing different styles.
- Space for switching between editing and viewing mode.

10 ADDITIONAL QUALITATIVE RESULTS

We apply a Gaussian filter with $\sigma = 1.0$ on the upsampled geometry. We use Blender for rendering all the qualitative results shown in the main paper and this supplemental material.

Fig. 14. Results of geometry detailization and texture generation on car category. Style shapes are shown in the first row and the input coarse voxels are shown in the first column. The input resolution is 64^3 and the output resolution is 512^3.

Fig. 15. Results of geometry detailization and texture generation on car category. Style shapes are shown in the first row and the input coarse voxels are shown in the first column. The input resolution is 64^3 and the output resolution is 512^3.

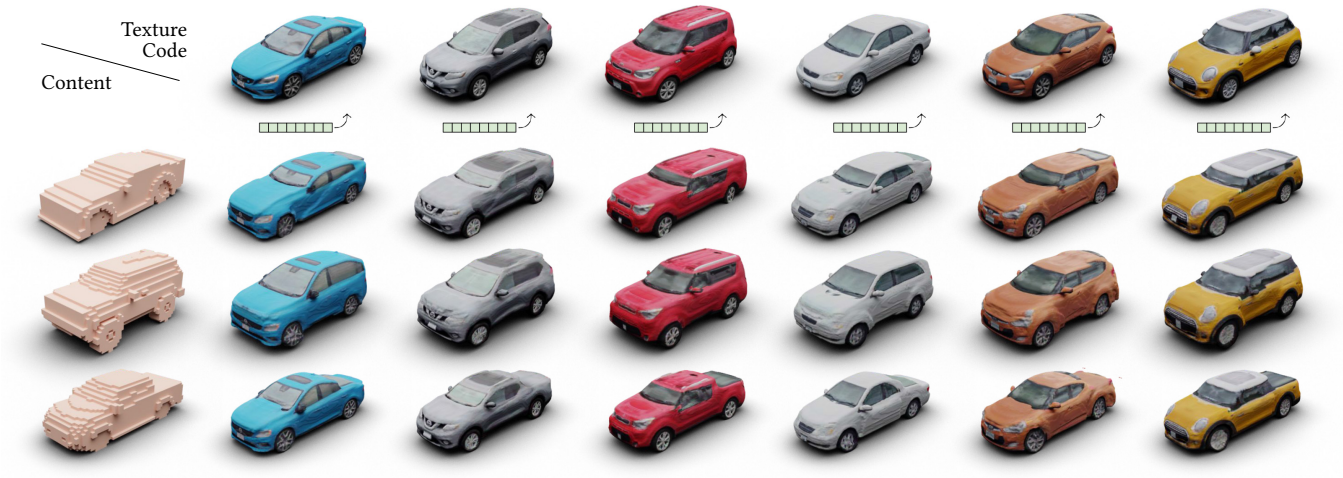


Fig. 16. Results of geometry detailization and texture generation conditioned on cars that are reconstructed from real-world images. We show the input coarse content voxels on the left and the detailed style shapes with textures on top. The coarse content voxels are 64^3 and the generated shapes are 512^3.

Fig. 17. Results of geometry detailization on car category. Style geometries are shown in the first row and the input coarse voxels are shown in the first column. Comparisons of DECOR-GAN-up and our multi-resolution are shown in the first and second row of each coarse voxel, respectively. The input resolution is 64^3 and the output resolution is 512^3. Please zoom in to observe the details.

Fig. 18. Results of geometry detailization and texture generation on airplane category. Detailed shapes are shown in the first row and the input coarse voxels are shown in the first column. The input resolution is 64^3 and the output resolution is 512^3.

Fig. 19. Results of geometry detailization and texture generation on airplane category. Style shapes are shown in the first row and the input coarse voxels are shown in the first column. The input resolution is 64^3 and the output resolution is 512^3.

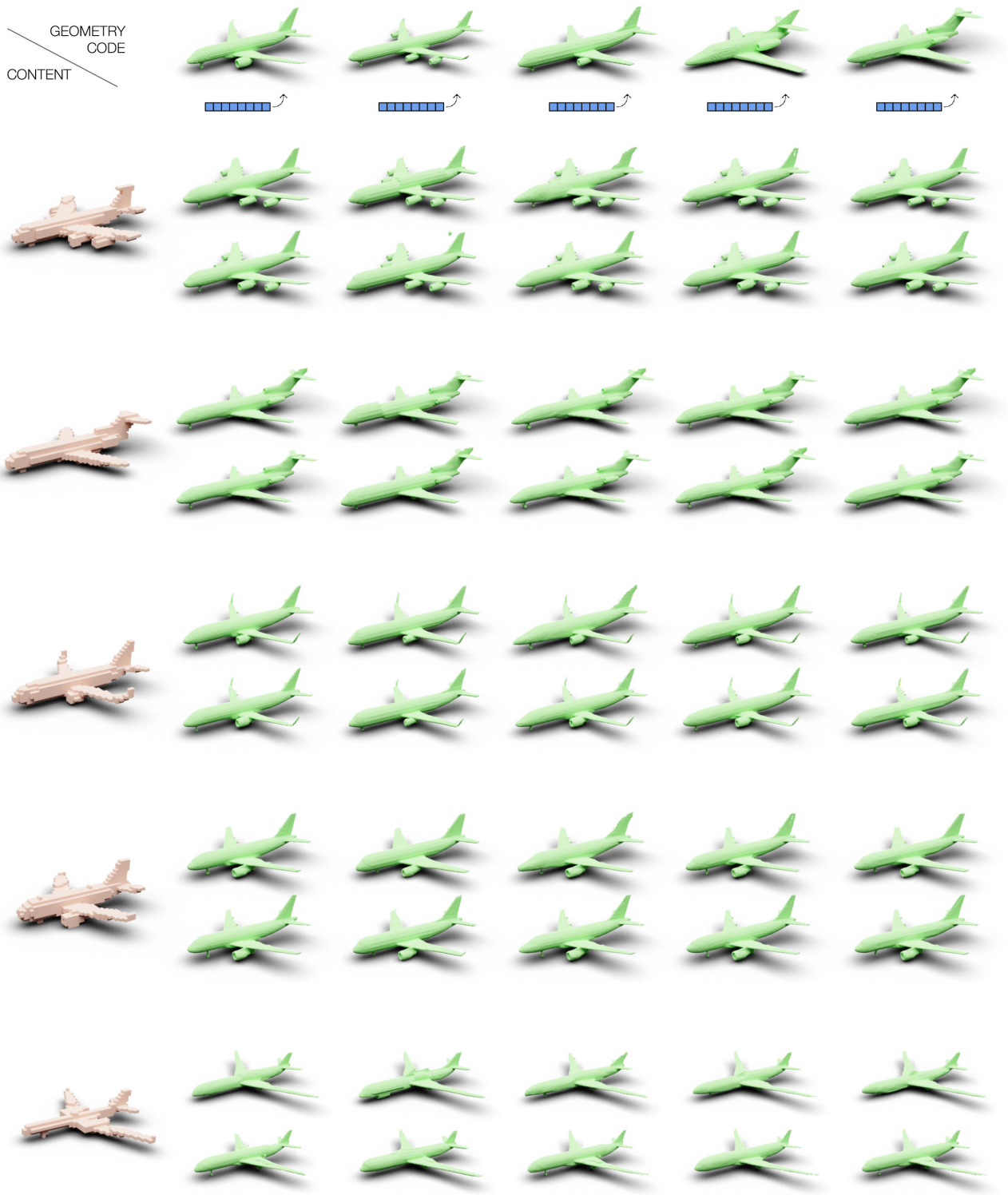


Fig. 20. Results of geometry detailization on airplane category. Style geometries are shown in the first row and the input coarse voxels are shown in the first column. Comparisons of DECOR-GAN-up and our multi-resolution are shown in the first and second row of each coarse voxel, respectively. The input resolution is 64^3 and the output resolution is 512^3. Please zoom in to observe the details.

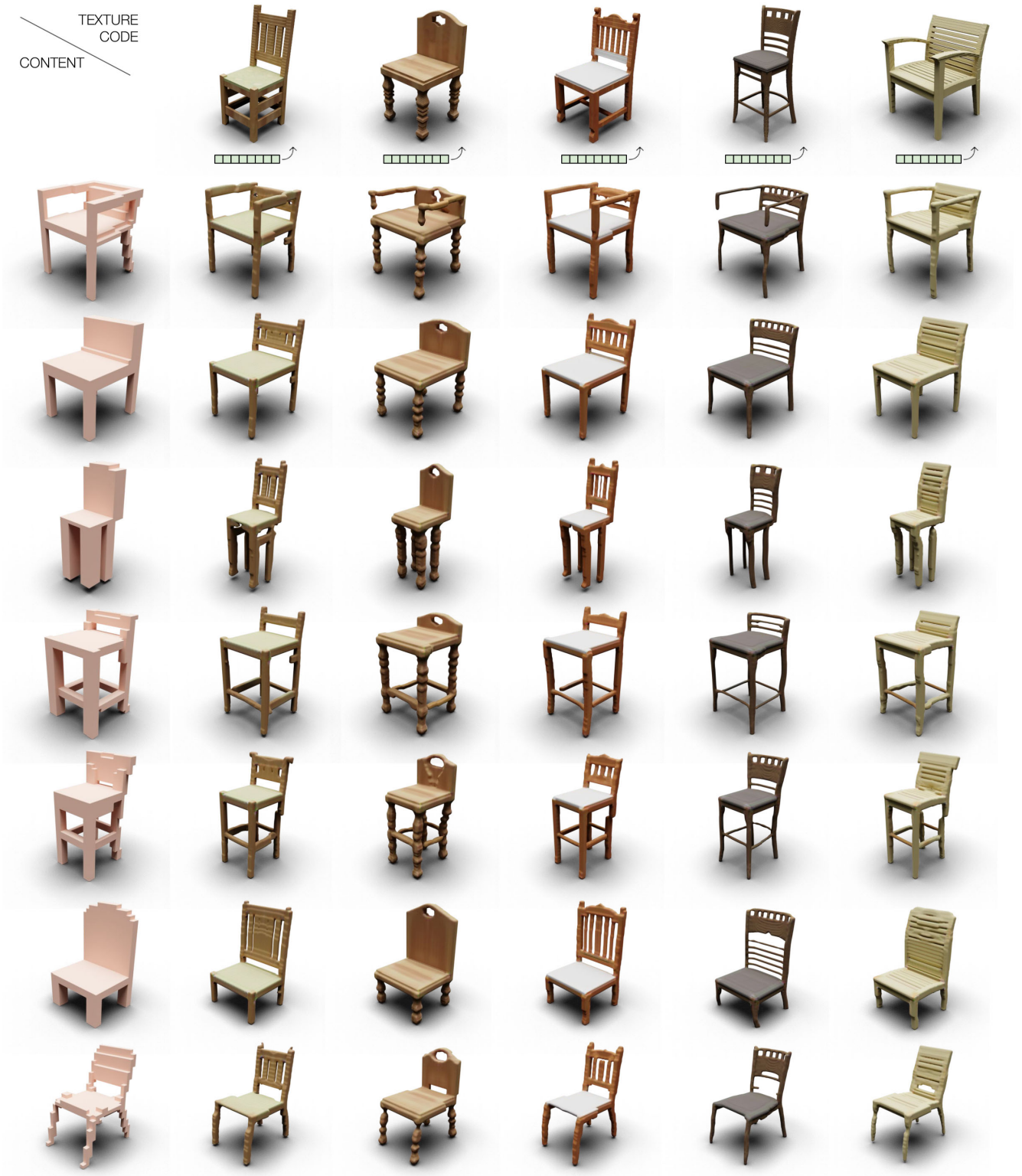


Fig. 21. Results of geometry detailization and texture generation on chair category. Style shapes are shown in the first row and the input coarse voxels are shown in the first column. The input resolution is 32^3 and the output resolution is 256^3.

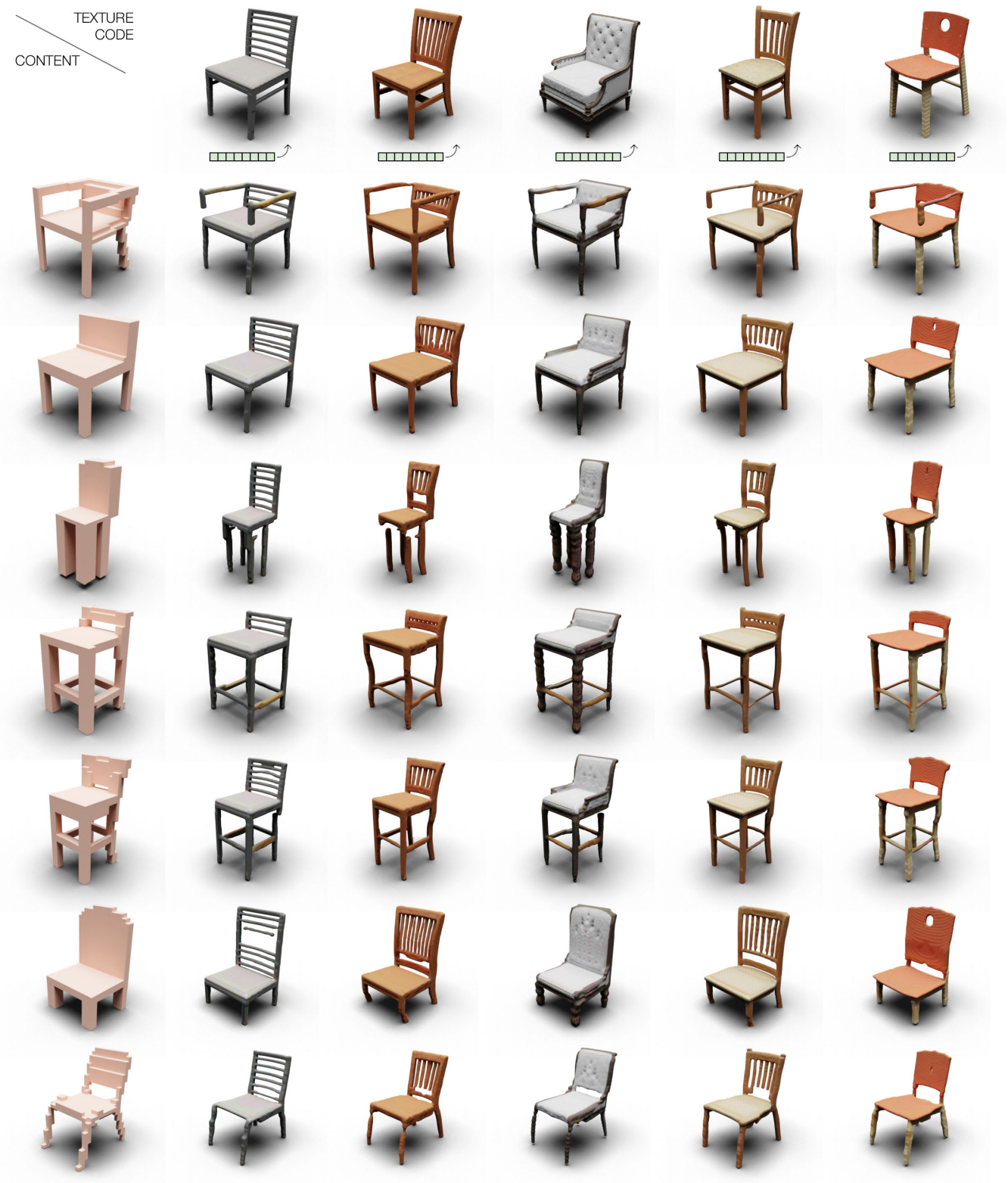


Fig. 22. Results of geometry detailization and texture generation on chair category. Style shapes are shown in the first row and the input coarse voxels are shown in the first column. The input resolution is 32^3 and the output resolution is 256^3.

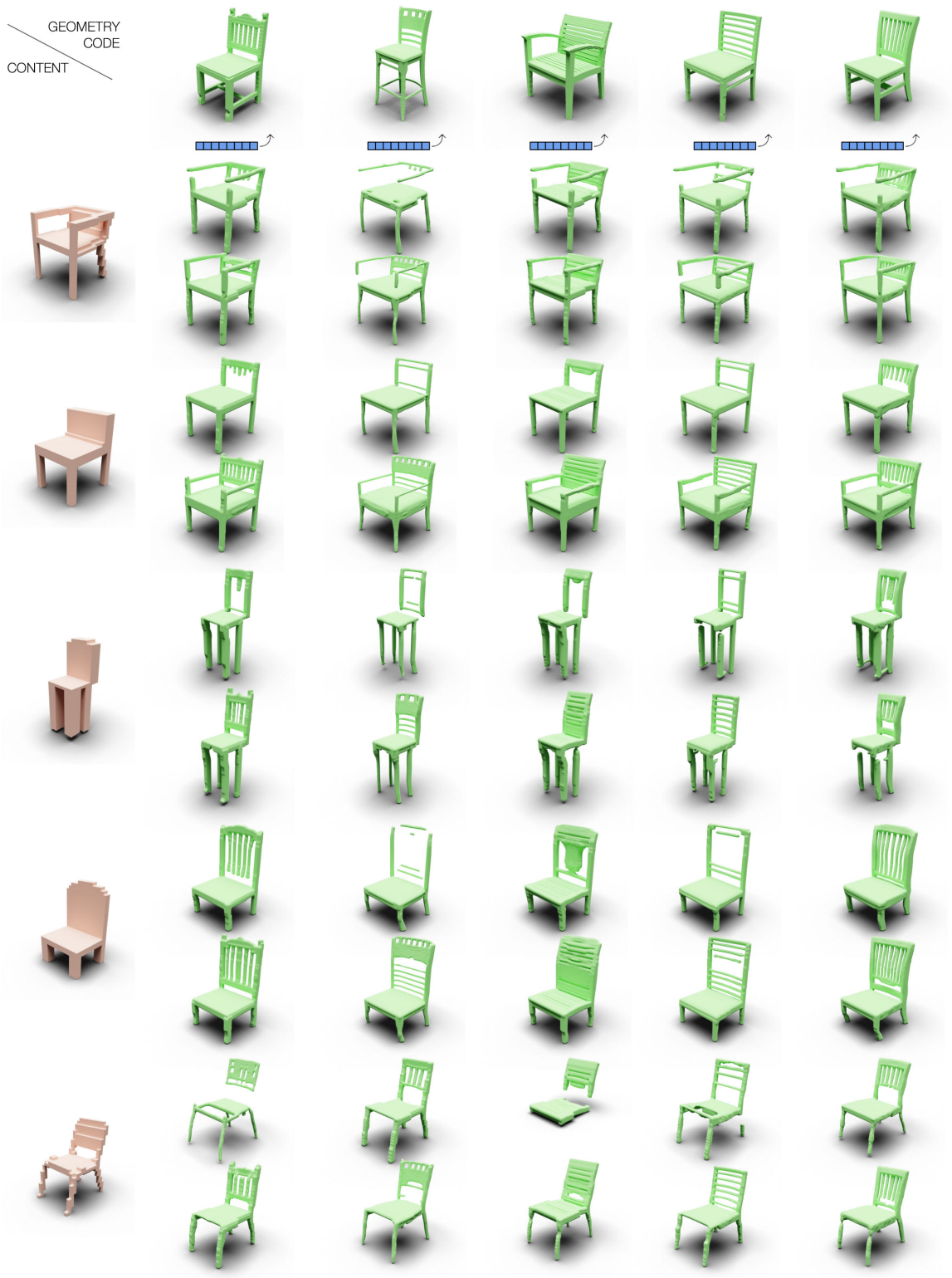


Fig. 23. Results of geometry detailization on chair category. Style geometries are shown in the first row and the input coarse voxels are shown in the first column. Comparisons of DECOR-GAN-up and our multi-resolution are shown in the first and second row of each coarse voxel, respectively. The input resolution is 32^3 and the output resolution is 256^3. Please zoom in to observe the details.

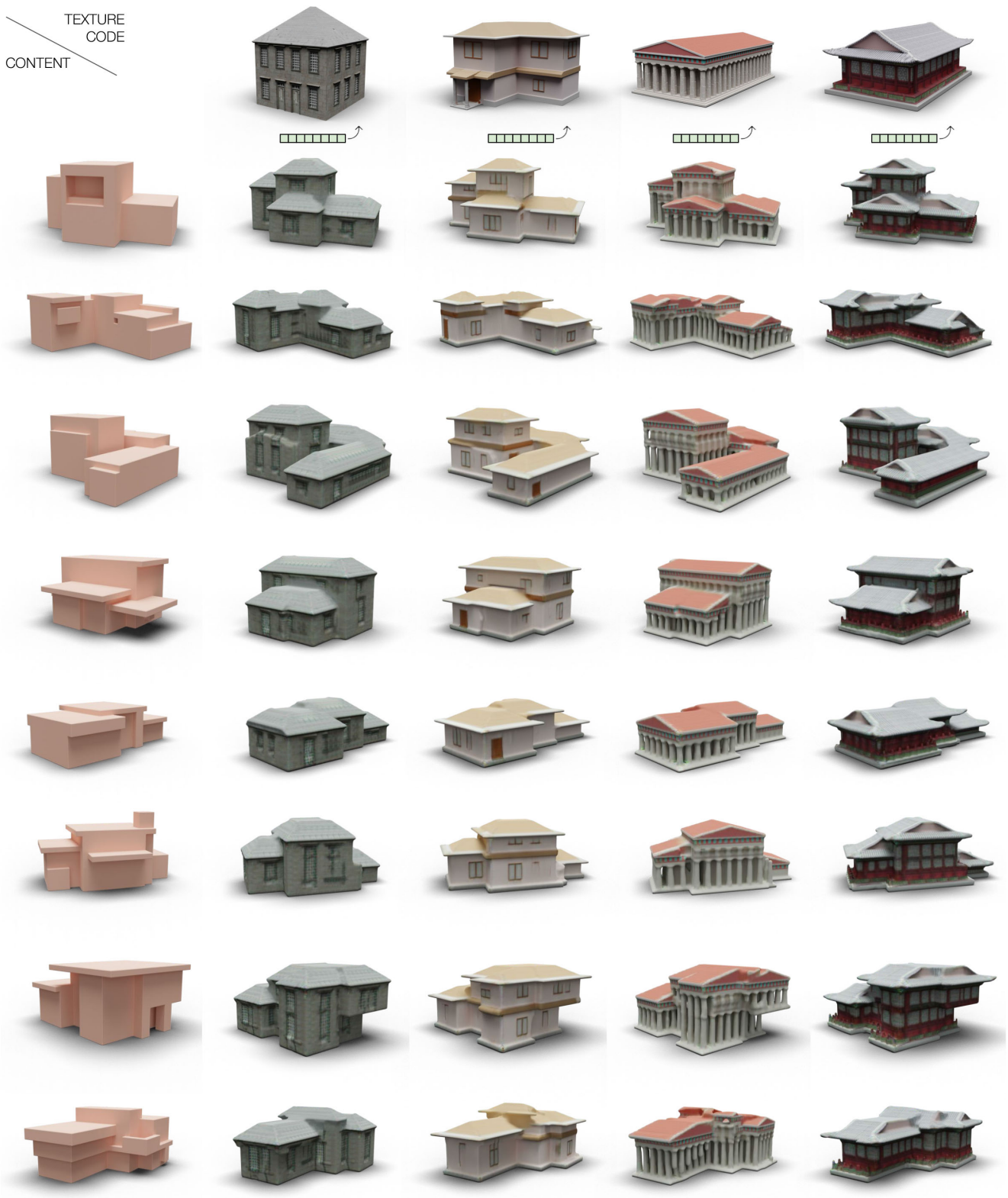


Fig. 24. Results of geometry detailization and texture generation on building category. Style shapes are shown in the first row and the input coarse voxels are shown in the first column. The input resolution is 32^3 and the output resolution is 256^3.

Fig. 25. Results of geometry detailization and texture generation on building category. Style shapes are shown in the first row and the input coarse voxels are shown in the first column. The input resolution is 32^3 and the output resolution is 256^3.

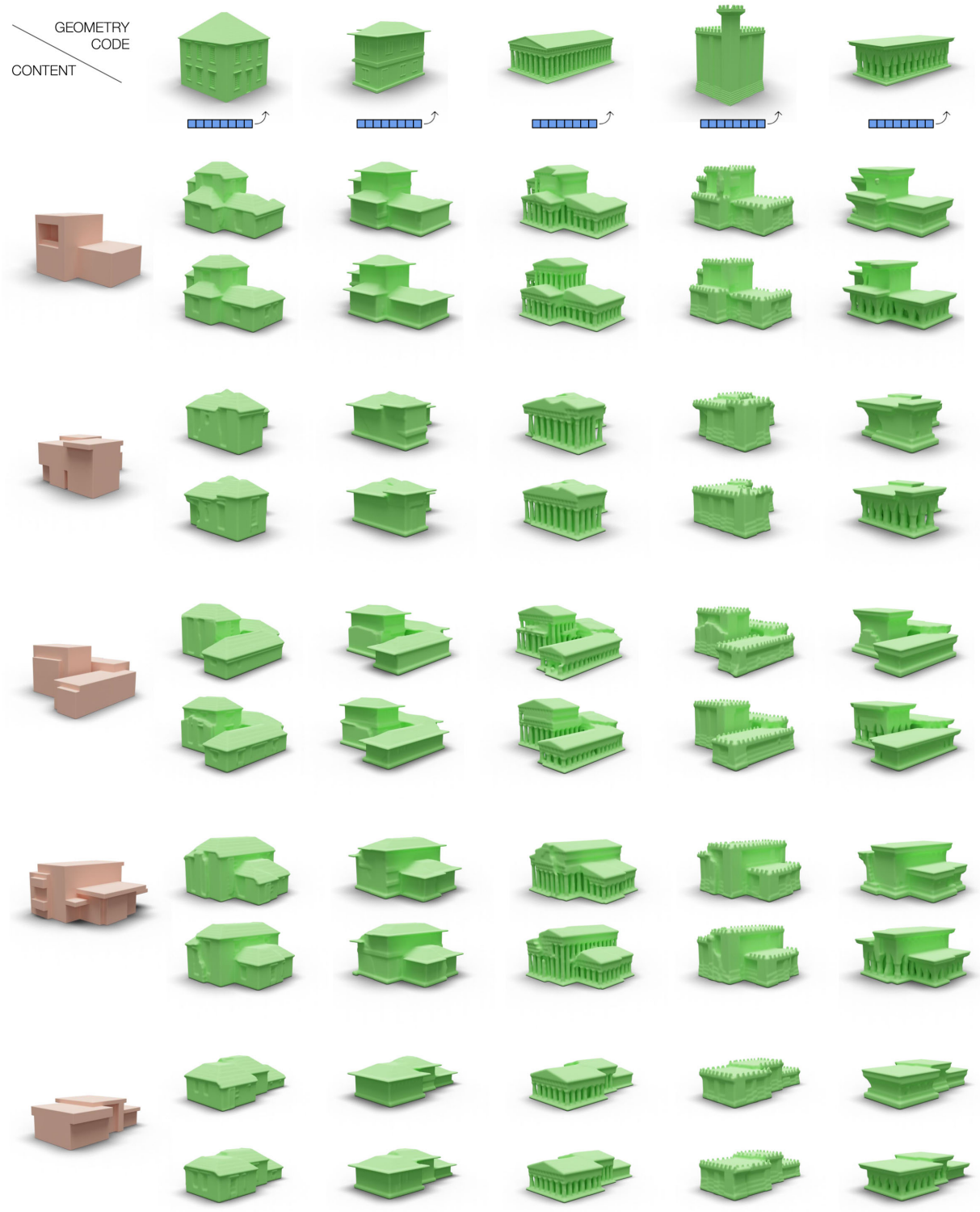


Fig. 26. Results of geometry detailization on building category. Style geometries are shown in the first row and the input coarse voxels are shown in the first column. Comparisons of DECOR-GAN-up and our multi-resolution are shown in the first and second row of each coarse voxel, respectively. The input resolution is 32^3 and the output resolution is 256^3. Please zoom in to observe the details.