

Deep Geometric Texture Synthesis

AMIR HERTZ*, Tel Aviv University

RANA HANOCKA*, Tel Aviv University

RAJA GIRYES, Tel Aviv University

DANIEL COHEN-OR, Tel Aviv University

Recently, deep generative adversarial networks for image generation have advanced rapidly; yet, only a small amount of research has focused on generative models for irregular structures, particularly meshes. Nonetheless, mesh generation and synthesis remains a fundamental topic in computer graphics. In this work, we propose a novel framework for synthesizing geometric textures. It learns geometric texture statistics from local neighborhoods (*i.e.*, local triangular patches) of a single reference 3D model. It learns deep features on the faces of the input triangulation, which is used to subdivide and generate offsets across multiple scales, without parameterization of the reference or target mesh. Our network displaces mesh vertices in any direction (*i.e.*, in the normal *and* tangential direction), enabling synthesis of geometric textures, which cannot be expressed by a simple 2D displacement map. Learning and synthesizing on local geometric patches enables a genus-oblivious framework, facilitating texture transfer between shapes of different genus.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Shape analysis**.

Additional Key Words and Phrases: Geometric Deep Learning, Shape Synthesis

ACM Reference Format:

Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. 2020. Deep Geometric Texture Synthesis. *ACM Trans. Graph.* 39, 4, Article 108 (July 2020), 11 pages. <https://doi.org/10.1145/3386569.3392471>

1 INTRODUCTION

In recent years, neural networks for geometry processing have emerged rapidly and changed the way we approach geometric problems. Yet, common 3D modeling representations are irregular and unordered, which challenges the straightforward adaptation from image-based techniques. Recent advances enable applying convolutional neural networks (CNNs) on irregular structures, like point clouds and meshes [Li et al. 2018a; Hanocka et al. 2019]. So far, these CNN-based methods have demonstrated promising success for discriminative tasks like classification and segmentation. On the other hand, only a small amount of research has focused on generative models for irregular structures, particularly meshes [Gao et al. 2019].

*Joint first authors.

Authors' addresses: Amir Hertz, Tel Aviv University; Rana Hanocka, Tel Aviv University; Raja Giryes, Tel Aviv University; Daniel Cohen-Or, Tel Aviv University.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2020 Association for Computing Machinery.

0730-0301/2020/7-ART108 \$15.00

<https://doi.org/10.1145/3386569.3392471>

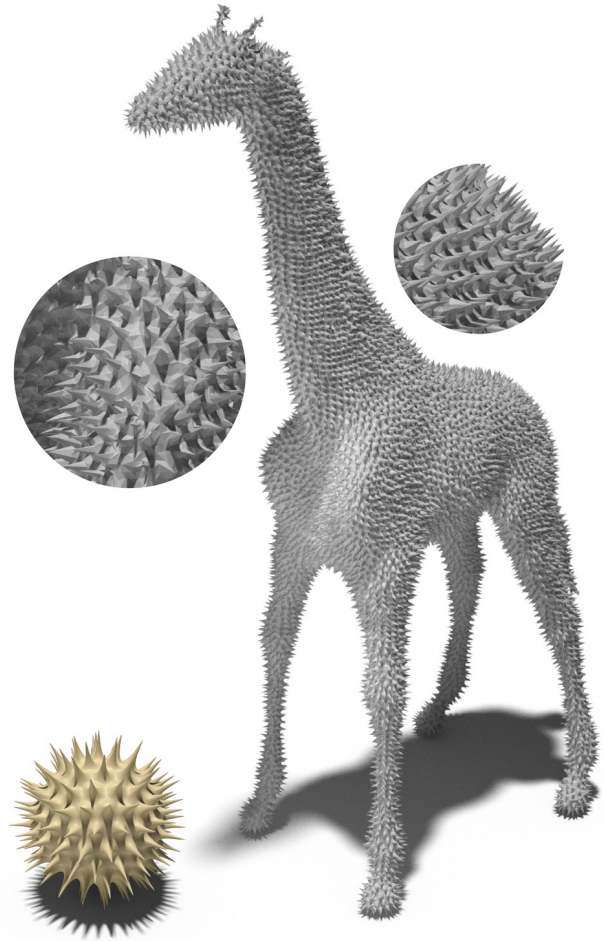


Fig. 1. Learning local geometric textures from a reference mesh (gold) and transferring it to a target mesh (giraffe).

In this work, we take a step forward in developing generative models for meshes. We present a deep neural network that learns the geometric texture of a single 3D reference mesh, and can transfer its texture to any arbitrary target mesh. Our generative framework uses a CNN to learn to model the unknown distribution of geometric textures directly from an input triangulation. Our network learns local neighborhoods (*i.e.*, local triangular patches) from a reference model, which is used to subdivide and generate offsets over the target mesh to match the local statistics of the reference model. For example, see Figure 1, where the geometric spikes of the reference

3D model are learned, and then synthesized on the target surface of the giraffe.

In this work, we calculate deep features directly on the mesh triangles and exploit a unique property of triangular meshes. Every triangle in manifold triangular mesh is adjacent to exactly three faces (Figure 3), which defines a fixed-sized convolutional neighborhood, similar in spirit to MeshCNN [Hanocka et al. 2019]. Our network generates mesh vertex displacements to synthesize local geometries, which are indistinguishable from the local statistics of the reference texture. To facilitate learning the statistics of geometric textures over multiple scales, we process the mesh using a hierarchy. We start with a low-resolution mesh (e.g., an icosahedron), and iteratively subdivide its faces and refine the geometry for each scale in the hierarchy.

Our method of transferring geometric texture from a reference model to a target model has notable properties: (i) it requires no parameterization, of neither the reference nor target surface; (ii) the target surface can have an arbitrary genus, which is not necessarily compatible with the reference surface, and last but not least, (iii) it is generative: reference patches are not copied or mapped, instead, they are learned, and then probabilistically synthesized. Note that geometric textures can be rather complex, as shown in Figure 10, which cannot simply be expressed by 2D displacement maps. Our network is given the freedom to displace mesh vertices in any direction, *i.e.*, not only along the normal direction, but also tangentially.

We demonstrate results of transferring geometric textures from single meshes to a variety of target meshes. We show that the reference mesh can have a different genus than the target mesh. Moreover, we show that our generative probabilistic model synthesizes variations of the reference geometric texture based on different latent codes.



Fig. 2. Our method is agnostic to the genus of both the reference and target meshes. Learning the geometric texture on a cat with a genus of one, and transferring it to the *fertility* statue with a genus of four.

2 RELATED WORK

Generative models have garnered significant attention since the introduction of Generative Adversarial Network (GAN) [Goodfellow et al. 2014]. GANs are commonly trained on a large data set (typically images), attempting to generate novel samples that come from the distribution of the training data. Recently, some works presented GANs trained on a single image [Zhou et al. 2018; Shocher et al. 2018; Gandelsman et al. 2019; Shaham et al. 2019; Sun et al. 2019]. The basic idea is to learn the distribution of local patches from the patches of the reference image, and then apply the knowledge in various applications. In the same spirit, in this work, we learn the distribution of local patches, but of 3D triangular meshes, which, unlike images, have an irregular structure.

Deep generative models in 3D. In recent years, a large body of works have proposed generating or synthesizing 3D shapes using deep neural networks. 3D shapes are commonly represented by irregular structures, which challenge the use of traditional convolutional neural networks. Thus, early approaches proposed using a volumetric representation, which naturally extends 2D image CNN concepts to a 3D voxel grid [Wu et al. 2015, 2016]. However, applying CNNs on 3D voxel-grids is highly inefficient, as it necessarily incurs huge amounts of memory, particularly when a high resolution is required.

On the other hand, a sparse and more direct portrayal of shapes uses the point cloud representation, which is simple and native to scanning devices. Achlioptas et al. [2018] pioneered the concept of deep generative models for point clouds, using the operators defined in pointnet [Qi et al. 2017a], which uses 1×1 convolutions. Later, these ideas were extended to hierarchical structures and synthesis [Qi et al. 2017b; Li et al. 2018b]. Recently, Yang et al. [2019] proposed a probabilistic framework for generating 3D point clouds. However, since the point cloud representation struggles to accurately portray fine grained details [Hertz et al. 2020], it is not a common choice for representing shapes in 3D art or animation.

Alternatively, a 3D object surface and the fine grained details can be represented more accurately using some form of parameterization. Groueix et al. [2018] propose representing surfaces as local charts, and use a deep network to learn a 2D parameterization for different tasks such as auto-encoders and surface reconstruction. The concept of using local charts for 3D shape generation has been further explored by Ben-Hamu et al. [2018]. Kostrikov et al. [2018] propose generating surfaces through intrinsic networks. An alternative to an explicit surface is an implicit representation, for example using a signed-distance function and extracting the surface from its zero level-set. Recently, implicit fields have been proposed for 3D shape generation [Park et al. 2019; Chen and Zhang 2019; Chen et al. 2019b].

The most common 3D representation in computer graphics is the polygonal mesh, a favorite of many due to the efficient, yet accurate portrayal of the underlying surface. Existing techniques for explicit mesh generation typically deform a template mesh, which preserves the genus and connectivity of the template. Pixel2Mesh [Wang et al. 2018] is a network for generating genus-0 shapes by deforming a sphere from an input RGB image. DIB-R [Chen et al. 2019a] is a differentiable renderer which was demonstrated to reconstruct

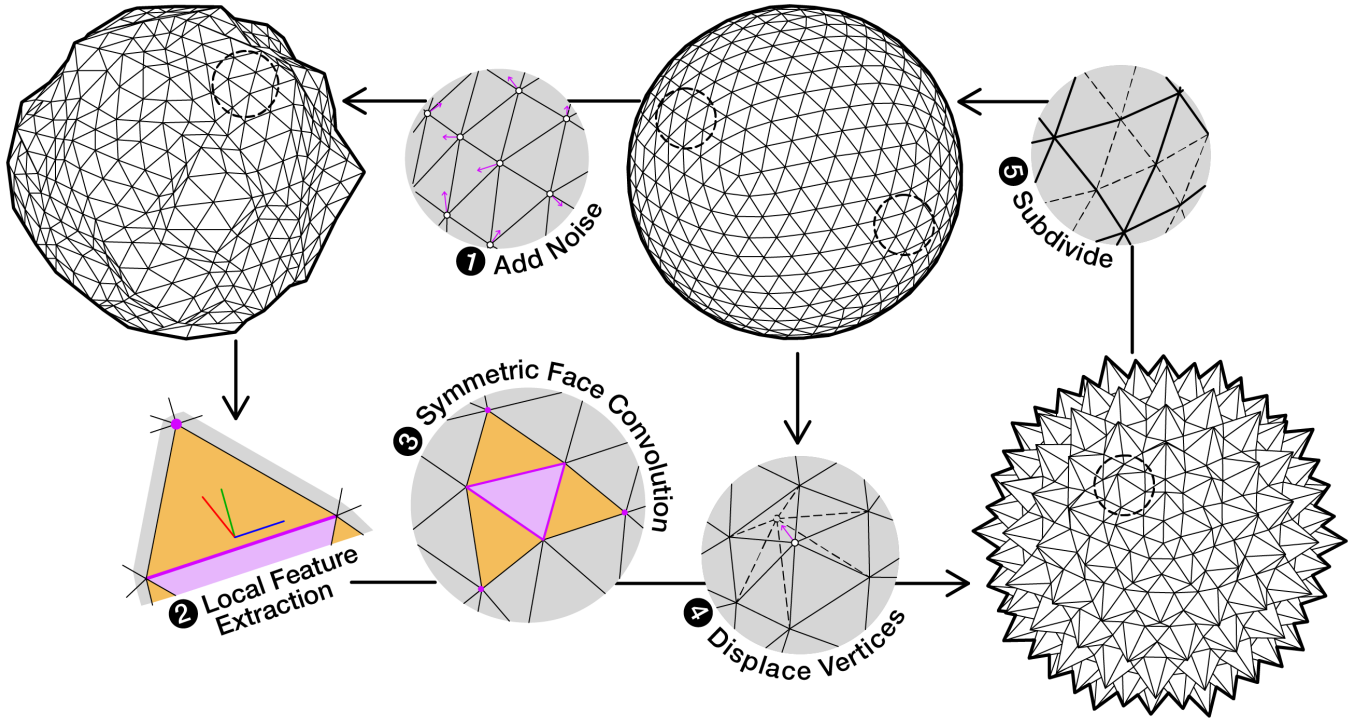


Fig. 3. Method overview. Starting with the training input in the current scale in the hierarchy, we (1) add noise to the vertices and (2) extract local rotation and translation invariant input features per triangular face. We (3) learn face-based equivariant convolutional kernels which learn to generate differential displacements per vertex with respect to the input. Subdividing the generated mesh progresses to the next level in the hierarchy.

3D objects from images. SDM-Net [Gao et al. 2019] is a VAE-based network for generating genus-0 mesh parts, yet, the collective sum of the parts can define non-genus zero shapes.

The most related work is MeshCNN [Hanocka et al. 2019], a neural network with operators that delete and un-collapse edges from a mesh for discriminative tasks like segmentation. However, unlike Hanocka et al. [2019], in this work, we propose a *generative* network for synthesizing new mesh geometries. Since we learn from local geometric patches, our framework is oblivious to genus, and can transfer textures between arbitrary genus shapes (Figures 2 and 8).

Texture transfer on Meshes. Texturing a target surface has been a fundamental problem in computer graphics. Basically, texture mapping requires parameterizing the target surface to define a low-distortion mapping between the source surface and target surface [Sorkine et al. 2002; Lévy et al. 2002; Sheffer et al. 2007]. In the most common setting, the source surface is a plane with a trivial parameterization. Naively, mapping a topological disc with boundary seams, where the boundaries are mapped and form discontinuities. Various works dealing with special textures with symmetries have developed continuous seamless mappings between closed surfaces (i.e., no boundaries) which have compatible genus [Aigerman et al. 2015; Aigerman and Lipman 2015; Knöppel et al. 2015; Campen et al. 2018].

Rather than mapping textures between surfaces, the textures can be synthesized over the target surface. Ying et al. [2001] and Turk et al. [2001] have presented texture synthesis techniques that synthesize textures from a 2D exemplar directly on the triangles of a target mesh. Their methods extend basic image space texture synthesis techniques by forming local parameterization over the mesh. Xu et al. [2009] present a more advanced method applied on meshes which is based on texture optimization [Wexler et al. 2004; Kwatra et al. 2005].

The above texture synthesis techniques are based on the premise that there is a simple local mapping between patches on the target and the source surfaces. Thus, they assume that the source surface is a flat image with a trivial parameterization [Chen et al. 2012]. The method we present does not map local patches, but learns the local geometries from the source mesh and synthesizes local geometries over the target mesh using a neural generative model. As noted earlier, local geometries are often too complex to be modeled by a simple 2D displacement maps. Recently, Liu and Jacobson [2019] proposed an approach for cubic stylization, which can *cubify* a 3D mesh directly, without any parameterization. Applying as-rigid-as-possible [Sorkine and Alexa 2007] reconstruction with an ℓ_1 regularization on the normals leads to a cubic stylization that is detail-preserving.

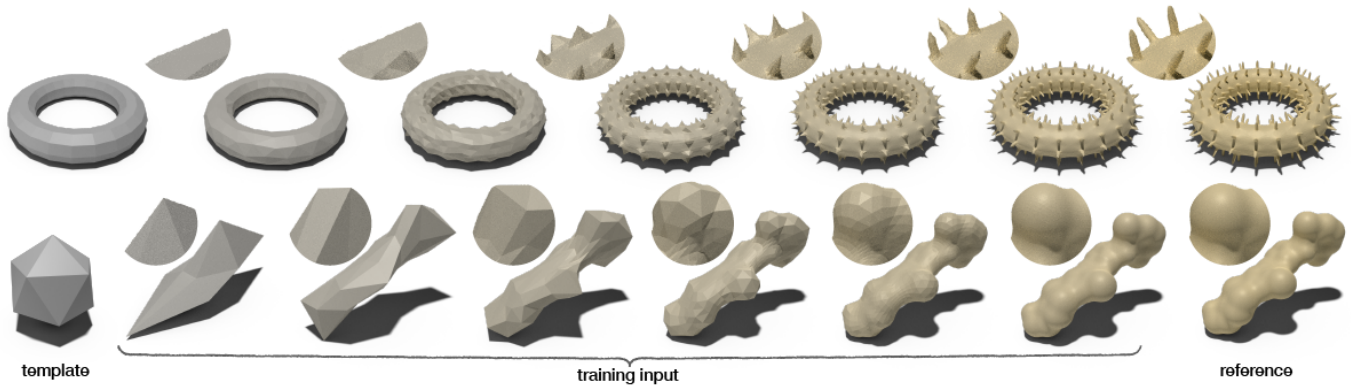


Fig. 4. Multiscale training data generation. Given a reference mesh with geometric texture, we create a series of multi-resolution training data using an optimization strategy. Starting with a low-res template mesh we repeatedly subdivide and optimize the mesh geometry, to obtain a training input with increasing resolution.

3 DEEP GEOMETRIC TEXTURE SYNTHESIS

We present a framework for learning to synthesize the local geometric texture from a single mesh. We learn the statistics of the patches in a hierarchical, coarse-to-fine manner, where the input to each level is a subdivided version of the output coarser level. Figure 3 illustrates an overview of a single level in the hierarchy. We train a generative adversarial network (GAN) on the patches (*i.e.*, local triangulations) of a single mesh, where the generator aims to synthesize local mesh patches that are indistinguishable from the reference patch.

Given a reference mesh with geometric textures, we create a series of meshes which depict the reference mesh across multiple resolutions. This multi-resolution series is used as input to train the hierarchical network. We obtain these *multi-scale training inputs* via a preliminary optimization strategy. Starting with a low-resolution template mesh, the vertices are optimized such that its surface will match the reference mesh. This template is repeatedly subdivided and optimized to fit the reference, resulting in a multi-scale representation of the reference mesh (example in Figure 4). From this point forward, the reference mesh is *discarded*, and these multi-scale training inputs are used to train the discriminator and generator.

Starting with the coarsest scale training mesh, we add Gaussian noise to its vertices, which are then used as input to the network. Then, we extract *local* geometric features per triangular face, which are invariant to rigid transformations. The initial geometric features pass through a series of face convolutions to learn deep features. The output of the final convolutional layer is a displacement vector per triangular face, which describes a displacement for each of the three incident vertices. To generate the final displacement vector per-vertex, we average the displacement vectors of all its incident faces. The displaced mesh is then refined by a subdivision and fed as input to the next level in the hierarchy.

The synthesized mesh in each scale is passed to the discriminator in the same scale, which learns to discriminate whether local patches (*i.e.*, faces) are *real* or *fake*. The discriminator trains face-based convolutional kernels to abstract the input geometric features to salient deep features, which indicate whether the local mesh is

synthesized or real. Note that the series of generators and discriminators have decreasing receptive fields that control the scale-space for synthesizing the geometric textures, in a similar hierarchical fashion as Shaham et al. [2019] demonstrated on images.

After training is complete, we discard the discriminators, and use the series of multi-scale generators to displace vertices of *any* novel target mesh. The scale space of the synthesized geometric texture is determined by the scale of the generators employed. See for example Figure 6, where the training input (gold) is transferred to a target (gray) starting from the coarsest scale (left) to the finest scale (right). Note that the target mesh may have a different triangulation and genus from the training input data.

In the process, we exploit a unique property of triangle meshes: the one-ring neighborhood of a mesh triangle has a fixed size of three triangles (step (3) in Figure 3). Similar to the edge-based convolution in MeshCNN [Hanocka et al. 2019], we learn convolutional kernels which operate on the faces of each triangle, and the three neighboring faces of each triangle. We apply convolutions on the features of each face and the neighboring 1-ring faces. To be invariant to the initial ordering of the mesh, we apply symmetric functions to the features in the neighboring 1-ring, resulting in an equivariant triangular face convolution.

3.1 Triangular Mesh Representation

A triangular mesh is a special type of graph defined by a set of vertices and triangles: (V, F) , where $V = \{v_1, v_2 \dots v_n\}$ is the unordered set of vertex positions in \mathbb{R}^3 . The mesh connectivity, or adjacency information, is designated by an unordered set of triangular faces $F = \{f_1, f_2 \dots f_m\}$, each containing a triplet of vertices, which implicitly constructs the edges of the graph $E = \{e_1, e_2 \dots\}$ (pairs of vertices).

Input Features. At each resolution level, the input features to our network are defined locally on each face and describe the relations between each face and its three neighboring faces. We define a local coordinate system for each edge in every face, where the origin is the edge midpoint. We use the face normal to define a consistent orientation for each local x, y, z axis. The local z -axis orientation

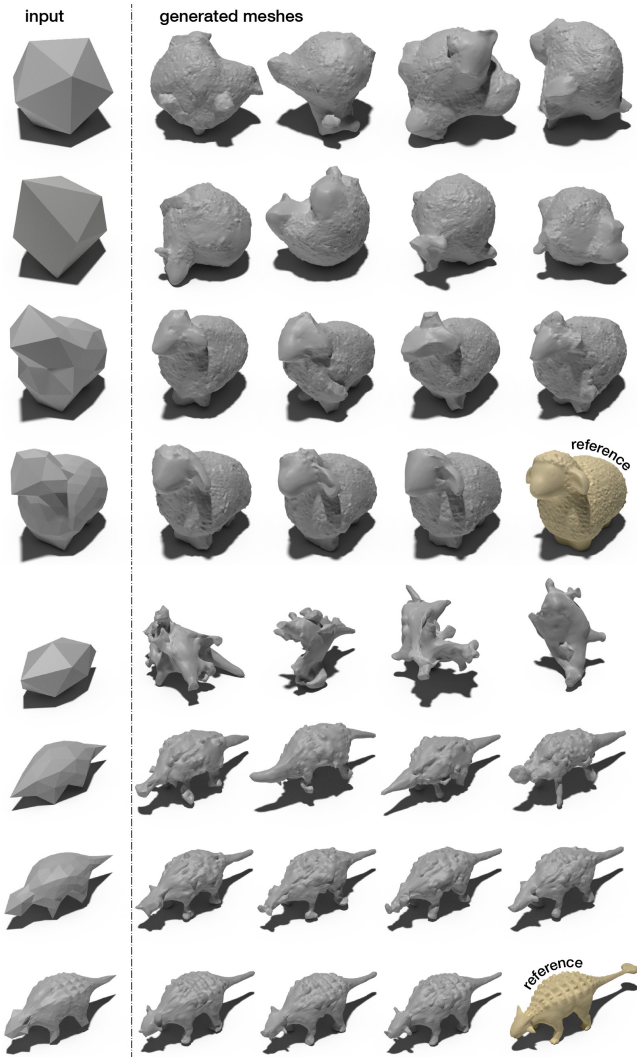


Fig. 5. Unconditional mesh generation. Our method can unconditionally generate meshes (top rows), or conditionally generate meshes in different scale spaces. Higher levels in the scale space conditioned on a higher level input mesh results in a synthesis that maintains the global structure of the reference mesh.

is defined by the face normal, x -axis is the edge direction and y is their cross product. Finally, we extract 4 features for each edge: edge length and Cartesian coordinates of the opposite vertex to that edge, projected onto the local coordinate system (see step 2 of Figure 3).

We denote the features of the three edges of the face f by the matrix $S \in \mathbb{R}^{3 \times 4}$, where each row contains the features of a single edge. These features are invariant to translations and rotations of the mesh. Moreover, these features contain enough information to reproduce the mesh in any global position and orientation from any face.

3.2 Symmetric Face Convolution

In our network, we first perform a 1×1 convolution on the input features to learn an order invariant *face feature embedding*. Then, we perform a symmetric convolution that takes into account the three 1-ring neighbours of the face.

Face Feature Embedding. The geometric features per face serve as input features to our face-based convolutional neural network, which are subsequently abstracted to deep features. We denote the dimension of the feature vector in convolution layer i by d_i . For the first convolutional layer of the network, we extract neural features $\hat{f} \in \mathbb{R}^{d_1}$ for each face side via a linear layer $\hat{S} = g(S|W, b) = SW + b$, where $S \in \mathbb{R}^{3 \times 4}$ are the extracted features of that face, and $W \in \mathbb{R}^{4 \times d_1}$ and $b \in \mathbb{R}^{d_1}$ are learned weights. As we want to generate a face embedding that is invariant to the order of neighbouring faces, we apply a *max* operation on the rows of \hat{S} . This leads to an initial face embedding \hat{f} that is invariant to rigid transformations and mesh face order.

Convolutions on Faces. In the subsequent convolutions, the input face features are the deep embedding from the previous layer. Unlike the first block, in subsequent layers, the convolution operates on the face feature vector and the 3 neighboring face feature vectors. Abusing notation, denote by $S \in \mathbb{R}^{3 \times d_i}$ the matrix whose rows contains the intermediate face embedding of the neighbouring faces of a face f and by $\hat{f} \in \mathbb{R}^{d_i}$ its intermediate embedding. Then, we define the linear operation for the face by $g(S, \hat{f}|W_S, W_f, b) = SW_S + \hat{f}^T W_f + b$, where $W_S \in \mathbb{R}^{d_i \times d_{i+1}}$, $W_f \in \mathbb{R}^{d_i \times d_{i+1}}$, $b \in \mathbb{R}^{d_{i+1}}$ are the learned weights. To ensure the convolution is invariant to the face ordering, we take a *Max* operation across the rows of $g(S, \hat{f}|W_S, W_f, b)$.

Vertex displacement. In this work, the face-based convolutions are used to build both the discriminator and generator networks. The discriminator uses the deep feature embedding to distinguish between *real* and *fake* faces, while the generator outputs 3D displacement vectors which modify the input mesh geometry. The generator outputs a single displacement vector per face, which is used to displace its three vertices symmetrically.

Each face predicts a displacement vector that is shared across all three vertices in that face, which is then projected onto the local coordinate axis of each edge, respectively. Since each vertex is shared by several faces, it receives multiple displacements. We average all of them to calculate its final displacement. Note that while each face predicts a symmetric displacement to each vertex, the vertices can be moved in all directions since they receive displacements from all the incident faces.

3.3 Subdivision and Multi-Scale Meshes

Realizing our goal to learn to synthesize geometric textures from a reference mesh, requires defining a method for *upsampling*, or subdividing the input mesh to achieve a hierarchical scale space. After defining a subdivision operator, it will be used to iteratively increase the resolution of some input mesh, such that with each subdivision, additional details from the reference mesh are added to the input mesh (see golden mesh in Figure 6).

Uniform Subdivision. In images upsampling is trivial, since downsampling and upsampling results in the same connectivity,

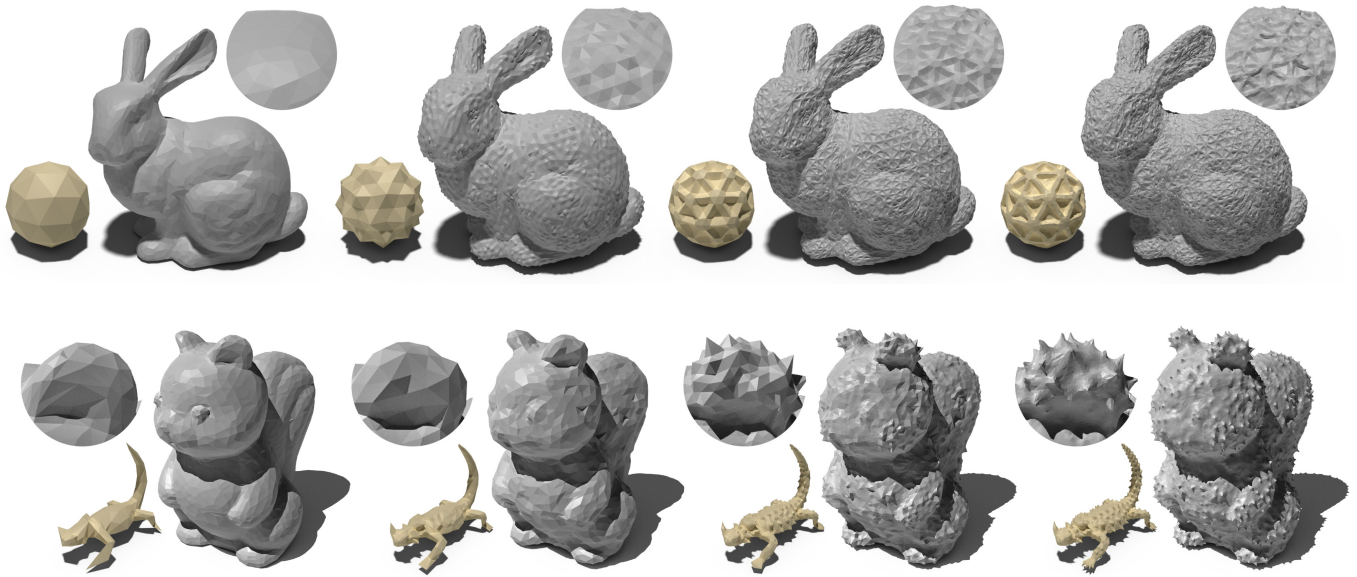


Fig. 6. Hierarchical texture scale space. A series of multi-scale generators are trained to synthesize geometric textures across multiple scales using the multi-scale training inputs (gold). During test time the geometric textures are synthesized on a novel target shape (gray). The scale space of the synthesized geometric texture is defined by the scale of the generators employed. The target shape is input to the first-level generator which synthesizes the first texture scale in the output (left). This output is passed to the second-level generator which synthesizes the next scale, and so on.

i.e., the local scale space of the image grid is preserved. However, for the irregular mesh structure, we must define an operator which can upsample both the training and inference meshes in a similar manner. For example, it is not sufficient to simply collapse edges in some pre-defined order, and then restore the collapses, since there would be no way of transferring this operation to a novel mesh. To this end, we propose using a uniform subdivision operator, which will have the same behavior on any given connectivity.

Uniform subdivision divides each face into four faces, by placing a triangle inside each face (see step 5 of Figure 3). A vertex is placed in the midpoint of every edge in the triangle, which increases the mesh resolution by four. This operation is *fixed*, meaning that given a specific connectivity (regardless of vertex placement), the uniform subdivision will always generate the same mesh. This property is desirable for transferring to novel target meshes which have a different connectivity than the training mesh.

Multiscale Input Shapes. Given a reference mesh with geometric texture, we employ a pre-processing phase to prepare a series of multi-scale input shapes which we will use for training. The user defines a low resolution template which is iteratively subdivided and deformed to match the reference mesh. The template was chosen to be either: an icosahedron, torus or coarse mesh (simplified version of the reference). Note that for a given reference mesh, the exact tessellation will not be recovered using uniform subdivision from some template. For this reason, we *remesh* the reference shape prior to training, and only use the multi-scale inputs during training (*i.e.*, discard the reference).

The proposed re-meshing procedure will generate a series of multi-scale training inputs. We create increasing resolutions (or

scales) of the reference mesh via an optimization procedure. Starting with a template mesh, we iteratively subdivide and minimize the distance to the reference mesh. As the number of mesh elements increase, the optimization will obtain a better fit to the reference mesh.

We solve this optimization problem through back-propagation, where the minimizer is the vertex locations of the training meshes. The optimization objective is measured by a bi-directional Chamfer distance between uniformly sampled points on both the reference and the optimized mesh. This distance is the Euclidean distance between each point on the training mesh and its closest point in the reference (and vice-versa), in addition to a negative cosine similarity between the normals on the meshes at those points.

We add two regularization terms to this optimization process to obtain a locally uniform triangulation and a smooth shape. The first (uniform) term encourages the minimization of the variance of the edge lengths and the second (smoothness) term reduces the distance between each vertex v_i on the mesh to the average coordinate of its one-ring:

$$\left| v - \frac{1}{d_i} \sum_{j:(i,j) \in E} v_j \right|,$$

where d_i is the degree of the vertex v .

3.4 Learning from a Single Mesh

We describe how we use our face-based convolutional layers to design a GAN model (generator and discriminator) that learns the local geometric statistics from a single mesh using a hierarchy of



Fig. 7. Geometric texture synthesis which are learned from a reference shape (gold) and transferred to different target shapes. Textures can be synthesized from natural shapes with geometric textures (e.g., the thorny lizard).

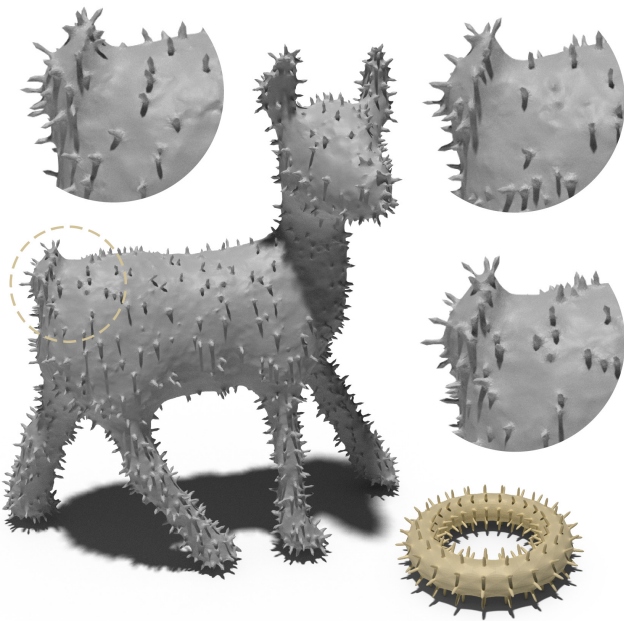


Fig. 8. Latent synthesized textures. By sampling different noise vectors, we can synthesize variations of the geometric texture.

generators. The generator network learns to predict vertex displacements to generate local geometries which are indistinguishable from the local statistics of the reference texture.

Hierarchical GAN training. We synthesize geometric textures via a series of generators which create local geometries incrementally. The output of a generator at a given level is local refinements to the input mesh, which is subdivided and used as input to the generator in the next level. In this manner, displacements in the coarse generator correspond to large refinements on the final mesh, and as the mesh progresses through the hierarchy, the generator displacements become fine-grained. This eases the training since each generator level only needs to capture the local refinements of each scale.

The generator in our model receives an input mesh and a tensor of noise z that is added to the input mesh vertices. Then, the generator outputs a displacement vector per-face, which is applied on the input mesh shape (without noise). The discriminator receives both the modified input mesh (*fake*) and the corresponding *real* mesh (*i.e.*, training shape with the same resolution) as input. An illustration of the *real* meshes for each level is shown in Figure 4. The discriminator is patch-based, so it learns to classify whether faces are *real* or *fake*. In other words, given an input mesh, the discriminator estimates a probability per face of being *real*.

As with standard GAN training, the goal of the generator is to fool the discriminator by generating shapes that are as similar to the *real* mesh, and the goal of the discriminator is to be able to distinguish between the generator output and the true mesh. We use the WGAN-GP [Gulrajani et al. 2017] framework to train both.

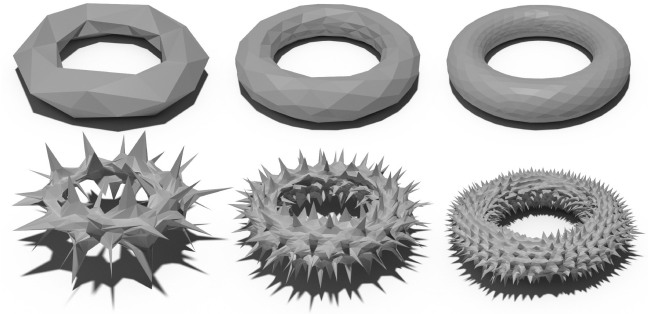


Fig. 9. Transferring geometric texture from the spikey ball (shown in Figure 1) to a torus with different resolutions. Transferring the spikes to a *low* resolution torus results in coarse texture scale space. Increasing the resolution of the torus increases the transferred texture scale space.

In addition to the adversarial loss, we add also a reconstruction loss as suggested in [Shaham et al. 2019]. We require that for a given fixed noise vector $z = c$, the generator will be able to reconstruct the *real* mesh. We use the *MSE* distance between the vertices of the generated and real meshes. To combine the two loss functions, we use a parameter γ to weight the reconstruction loss.

Training starts with the generator and discriminator in the coarsest level. The input to the coarsest generator is the template (+ noise $z = c$), while the *desired output* (*i.e.*, using reconstruction and adversarial loss) is the *training input* in the coarsest level. Both the generator and the discriminator are trained until convergence. When progressing to train the next level, the generator from the previous level is kept fixed. The output from the previous level is subdivided and scaled and then input to the next level. After subdivision, we uniformly scale the mesh such that the mean-edge length is preserved. We set c to be a fixed random noise vector in the coarsest resolution, and a vector of zeros in the higher resolutions.

Inference. Our generator network is fully convolutional, and therefore it can be applied to any mesh with any connectivity and resolution. Given a new shape (*i.e.*, *target* mesh), we use the generator to synthesize the learned local structures of the reference mesh. This is achieved by scaling the target shape to have an average edge length of one (*i.e.*, input feature normalization). We use the target shape plus random noise, as an input to the generator in one of the lower resolutions in the hierarchy. This leads to transferring the (local) structure of the reference mesh to the target mesh. Note, unlike the reference mesh, the given connectivity of the target mesh does not need to be re-meshed.

4 EXPERIMENTS

The reference meshes using for training our models are provided by Thing10K dataset [Zhou and Jacobson 2016] or built by hand. Our PyTorch [Paszke et al. 2017] implementation as well as pre-trained models and multiscale training meshes will be made publicly available upon publication.

Each model contains 5 – 7 levels (scales), depending on the complexity of the geometric texture in the reference mesh. Across all

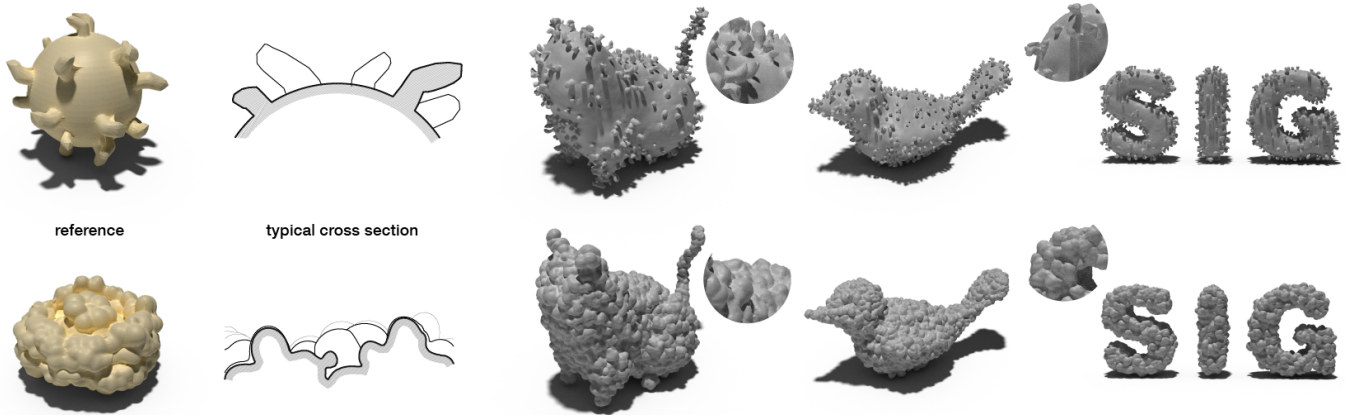


Fig. 10. Geometric textures with complex 3D displacements. The network learns to synthesize geometric textures from the reference geometry, whose tangential movement is highlighted in a cross section illustration, respectively. Synthesizing the geometric textures on different target shapes (right - gray).

levels, the generator and discriminator have 7 layers of face convolutions with instance normalization [Ulyanov et al. 2016] and leaky ReLU. The face feature embedding dimension (d_1 in Section 3.2), or the output of the first convolution layer increases as we move up the hierarchy. Starting with 32 in the first level, and reaching 128 at the third level. From the fourth level and onward, we initialize both the generator and the discriminator models, with the weights of the previous level. Each level was trained for 2000 iterations using the Adam optimizer [Kingma and Ba 2014] with a learning rate of $5e^{-4}$ and learning rate decay of 0.5 applied in intervals of 500 iterations. In all experiments, the reconstruction weight γ was set to 5.

Hierarchical Generation. Our hierarchical training allows synthesizing meshes starting from different levels of the generator. When starting from the lowest level with the template, the generator outputs different meshes with different global structures (see top rows of Figure 5). When synthesizing from higher levels by using higher resolution inputs, the generator preserves the global structure of the input and only deforms local regions on the mesh (see bottom rows in Figure 5).

This hierarchical characteristic enables applying our model on a variety of meshes with different resolutions during inference from any given level. In general, we usually start from level 2-3 of the source shape when performing the texture synthesis. However, this is dependent on the training meshes, and the scale spaces that are defined within each level. Some geometric textures require more levels, while others can be compactly defined in a few levels.

We evaluate the pretrained models by applying them on unseen target meshes. Figure 6 shows the hierarchical generation of textures. Notice how the geometric texture is transferred gradually from the source shape (in gold) to the target shape, where the process starts using the source shape in a lower resolution and progresses while increasing it.

Texture Synthesis. Figure 7 presents additional texture synthesis for various unseen target meshes from different reference shapes. Observe how our method is able to synthesize different geometrical structures on directly on the target shape, without the use of any parametrization.

A remarkable property of our approach is that it can process pairs with a different genus. See for example the torus and the pig in Figure 7. In Figure 2 the generator was trained on the cat model (genus of one), and was transferred to a target fertility shape with a genus of four.

Notice that the resolution (number of mesh faces) used in the target shape determines the scale of the texture synthesized on them. Figure 9 demonstrates this effect by synthesizing spikes on a torus mesh, where we start from different mesh resolutions and transfer the texture from levels 2 – 4 (top row). In all three cases, we synthesized the textures using the same generator scales trained on the spiky ball. Naturally, the resolution of the target affects the size of the synthesized texture.

Latent Space Interpolation. Since our framework is generative, it enables synthesizing different textures from the same reference shape. This can be done by sampling different noise vectors, resulting in different synthesized textures on the target mesh. We show examples on two different shapes and textures in Figure 8. Note that since the generator was trained on a single reference mesh, the differences in the synthesized texture on the same target shape are solely due the noise vector. This enables smoothly interpolating between shapes by simply interpolating over the latent variable that was used for generation. Performing smooth interpolations between shapes enables animation of the textures. We provide several such examples in the supplementary material.

Comparison. We compare against OptCuts [Li et al. 2018c], a state-of-the-art parameterization technique, in Figure 11. We manually create a 2D displacement map which corresponds to the 3D reference shape (gold), and use OptCuts to automatically calculate the parameterization and cutting of the 3D mesh, resulting in a mapping of the displacement map to the target mesh. We use the UV mapping to displace vertices in the normal direction on the target mesh. On the other hand, our technique learns to synthesize geometric textures directly from a 3D reference mesh. Finally, it can be seen that the edges of OptCuts textures are sharp. However, automatically creating 2D displacement maps from 3D geometries is non-trivial. Moreover, a 2D displacement map that moves geometry

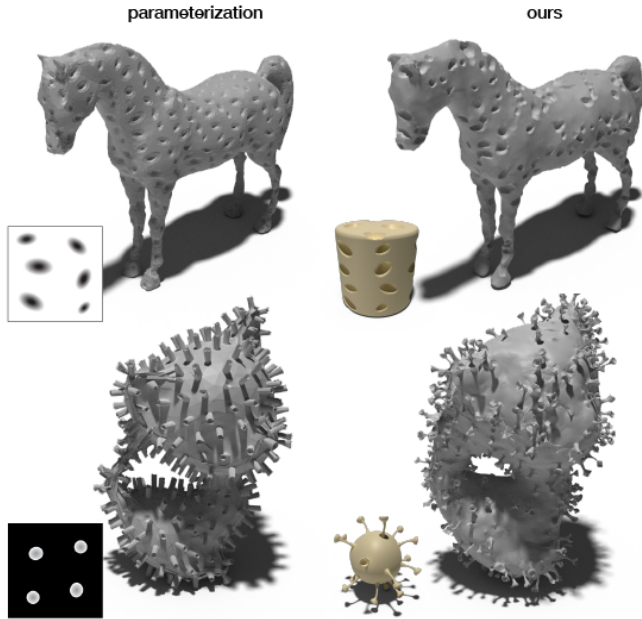


Fig. 11. Comparison to OptCuts [Li et al. 2018c] (left), which projects a 2D displacement map to the target surface. This UV mapping is used to displace vertices in the normal direction. The 2D displacement was estimated manually from the 3D reference shapes in gold: cylinder and coronavirus. Note the tangential displacements of the coronavirus are not captured by a 2D displacement map. Our technique (right) learns to synthesize 3D geometric textures directly from the reference mesh (gold).

in the normal direction is rather limited, since it does not encode tangential movement (e.g., the coronavirus). Lastly, OptCuts took 10 minutes to compute a parameterization, while our technique only requires a few seconds.

5 DISCUSSION AND FUTURE WORK

We have presented a novel concept for geometric texture synthesis, which uses a generative framework to learn the local structures from a given triangular mesh and then synthesizes it on different target models. Our technique learns to match the local statistics of a specified mesh model and transfers it to a target one. To the best of our knowledge, this is the first generative model that learns from a single mesh.

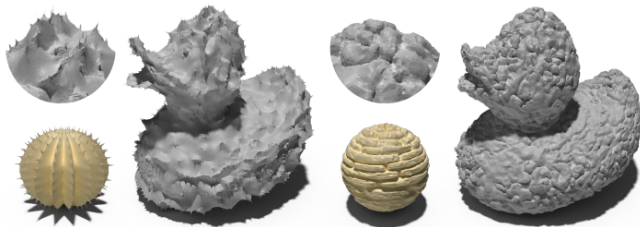


Fig. 12. Our method is limited to isometric textures. The vertical (cactus) and horizontal (brick) texture direction is not transferred to the duck.

A prominent advantage of our scheme is that it does not require any parameterization of the reference or target shape. Given a model with natural organic geometric texture (i.e., lizard Figure 6), which is not given as a displacement map, it is not immediately obvious how to employ a classic parameterization technique to transfer the geometric texture to another (target) shape (i.e., squirrel Figure 6). There is no generic method for decomposing an arbitrary surface with geometric textures into a *base* and displacements. Furthermore, not every geometric texture is simple enough to be represented as displacements along the surface normal (e.g., reference shapes with tangential movement in Figure 10). By contrast, our approach receives a reference 3D model which contains geometric texture (i.e., not a displacement map), and learns to synthesize geometric structures by displacing vertices in all directions (e.g., not only along the normal direction but also tangentially).

However, the presented method has its limitations. First, it learns to synthesize local textures, and cannot capture large structures. Moreover, it currently assumes that the geometric textures are stationary and isometric (e.g., Figure 12). Handling anisotropic textures would entail learning a directional field which can be transferred from the reference to the target mesh, a difficult task in and of itself. Moreover, even after the directional field is estimated, synthesizing the final geometric texture from the directional field is not a trivial task. Another limitation is that the hierarchical learning requires the mesh to have a locally-uniform triangulation and well-behaved subdivision structure. Currently, we achieve this via a preliminary remeshing process. This remeshing procedure may fail on complex shapes, e.g., thin and intertwined structures, and in general, where the euclidean and geodesic distances differ significantly. In the future we would like to relax this requirement, and build the hierarchy by learning vertex splits.

While the focus of this work is geometric texture synthesis from a single mesh, our approach opens the door for a variety of follow up works. For example, it is possible to use the machinery developed in this work for transferring color texture or other attributes. Furthermore, by learning different positions of the same shape, the generative model can be used to interpolate between two positions and thus, animate shapes in a controlled direction.

Another possible application of our method employs a geometric texture transfer using a *two-step* mapping. First, we build a local geometric texture on a simple shape such as a sphere or a torus, where automatic or semi-automatic tools work well. Then, this mesh is used as an intermediate shape toward the ultimate goal of generating textures on the target mesh. This two-step method is reminiscent of the 30+ year old work of Bier and Sloan [1986] for texture mapping.

Learning to synthesize is a challenging task, especially when it comes to irregular geometric data. The proposed face convolution facilitates the development of a GAN framework for triangular meshes. Our learning-based technique leads to results that are difficult to achieve using state-of-the-art graphics tools, or requires a tailored solution for each target shape. We believe that this work is just a first step towards the development of more deep-learning techniques for 3D generative mesh models.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their helpful comments. This work is supported by the NSF-BSF grant (No. 2017729), the European research council (ERC-StG 757497 PI Giryes), ISF grant 2366/16, and the Israel Science Foundation ISF-NSFC joint program grant number 2217/15, 2472/17.

REFERENCES

- Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. 2018. Learning Representations and Generative Models for 3D Point Clouds. In *International Conference on Machine Learning*. 40–49.
- Noam Aigerman and Yaron Lipman. 2015. Orbifold Tutte embeddings. *ACM Trans. Graph.* 34, 6 (2015), 190–1.
- Noam Aigerman, Roi Poranne, and Yaron Lipman. 2015. Seamless surface mappings. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 72.
- Heli Ben-Hamu, Haggai Maron, Itay Kezurer, Gal Avineri, and Yaron Lipman. 2018. Multi-chart generative surface modeling. In *SIGGRAPH Asia 2018 Technical Papers*. ACM, 215.
- Eric A Bier and Kenneth R Sloan. 1986. Two-part texture mappings. *IEEE Computer Graphics and applications* 6, 9 (1986), 40–53.
- Marcel Campen, Hanxiao Shen, Jiaran Zhou, and Denis Zorin. 2018. Seamless Parametrization with Arbitrarily Prescribed Cones. *arXiv preprint arXiv:1810.02460* (2018).
- Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. 2019a. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Advances in Neural Information Processing Systems*. 9605–9616.
- Xiaobai Chen, Tom Funkhouser, Dan B Goldman, and Eli Shechtman. 2012. Non-parametric texture transfer using meshmatch. *Technical Report Technical Report 2012-2* (2012).
- Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. 2019b. BSP-Net: Generating Compact Meshes via Binary Space Partitioning. *arXiv:cs.CV/1911.06971*
- Zhiqin Chen and Hao Zhang. 2019. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5939–5948.
- Yossi Gandelsman, Assaf Shocher, and Michal Irani. 2019. "Double-DIP": Unsupervised Image Decomposition via Coupled Deep-Image-Priors. (6 2019).
- 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 (TOG)* 38, 6 (2019), 1–15.
- Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.
- Thibault Groueix, Matthew Fisher, Vladimir Kim, Bryan Russell, and Mathieu Aubry. 2018. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *CVPR 2018*.
- Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved training of wasserstein gans. In *Advances in neural information processing systems*. 5767–5777.
- Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. MeshCNN: A Network with an Edge. *ACM Trans. Graph.* 38, 4, Article 90 (July 2019), 12 pages. <https://doi.org/10.1145/3306346.3322959>
- Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. 2020. PointGMM: a Neural GMM Network for Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12054–12063.
- Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2015. Stripe Patterns on Surfaces. *ACM Trans. Graph.* 34 (2015), Issue 4.
- Ilya Kostrikov, Zhongshi Jiang, Daniele Panozzo, Denis Zorin, and Joan Bruna. 2018. Surface networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2540–2548.
- Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. 2005. Texture optimization for example-based synthesis. In *ACM Transactions on Graphics (ToG)*, Vol. 24. ACM, 795–802.
- Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérôme Maillot. 2002. Least squares conformal maps for automatic texture atlas generation. In *ACM transactions on graphics (TOG)*, Vol. 21. ACM, 362–371.
- Jiaxin Li, Ben M Chen, and Gim Hee Lee. 2018b. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 9397–9406.
- Minchen Li, Danny M. Kaufman, Vladimir G. Kim, Justin Solomon, and Alla Sheffer. 2018c. OptCuts: Joint Optimization of Surface Cuts and Parameterization. *ACM Transactions on Graphics* 37, 6 (2018). <https://doi.org/10.1145/3272127.3275042>
- Yangyan Li, Rui Bu, Mingchao Sun, and Baoquan Chen. 2018a. PointCNN. *arXiv preprint arXiv:1801.07791* (2018).
- Hsueh-Ti Derek Liu and Alec Jacobson. 2019. Cubic stylization. *ACM Transactions on Graphics (TOG)* 38, 6 (Nov 2019), 11A–110. <https://doi.org/10.1145/3355089.3356495>
- Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. Deepsdf: Learning continuous signed distance functions for shape representation. *arXiv preprint arXiv:1901.05103* (2019).
- Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.
- Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017a. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 652–660.
- Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017b. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*. 5099–5108.
- Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. 2019. SinGAN: Learning a Generative Model from a Single Natural Image. *arXiv preprint arXiv:1905.01164* (2019).
- Alla Sheffer, Emil Praun, Kenneth Rose, et al. 2007. Mesh parameterization methods and their applications. *Foundations and Trends® in Computer Graphics and Vision* 2, 2 (2007), 105–171.
- Assaf Shocher, Nadav Cohen, and Michal Irani. 2018. "zero-shot" super-resolution using deep internal learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3118–3126.
- Olga Sorkine and Marc Alexa. 2007. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, Vol. 4. 109–116.
- Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. 2002. Bounded-distortion piecewise mesh parameterization. In *Proceedings of the conference on Visualization '02*. IEEE Computer Society, 355–362.
- Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A Efros, and Moritz Hardt. 2019. Test-Time Training for Out-of-Distribution Generalization. *arXiv preprint arXiv:1909.13231* (2019).
- Greg Turk. 2001. Texture synthesis on surfaces. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 347–354.
- Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. 2016. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022* (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 *Proceedings of the European Conference on Computer Vision (ECCV)*. 52–67.
- Yonatan Wexler, Eli Shechtman, and Michal Irani. 2004. Space-time video completion. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, Vol. 1. IEEE, I–I.
- 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. In *Advances in neural information processing systems*. 82–90.
- Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1912–1920.
- Kai Xu, Daniel Cohen-Or, Tao Ju, Ligang Liu, Hao Zhang, Shizhe Zhou, and Yueshan Xiong. 2009. Feature-aligned shape texturing. *ACM Transactions on Graphics (TOG)* 28, 5 (2009), 108.
- Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. 2019. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE International Conference on Computer Vision*. 4541–4550.
- Lexing Ying, Aaron Hertzmann, Henning Biermann, and Denis Zorin. 2001. Texture and shape synthesis on surfaces. In *Rendering Techniques 2001*. Springer, 301–312.
- Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint arXiv:1605.04797* (2016).
- Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. 2018. Non-stationary Texture Synthesis by Adversarial Expansion. *ACM Trans. Graph.* 37, 4, Article 49 (July 2018), 13 pages. <https://doi.org/10.1145/3197517.3201285>