Cross-atlas Convolution for Parameterization Invariant Learning on Textured Mesh Surface

Anonymous CVPR submission

Paper ID 2492

Abstract

We present a convolutional network architecture for direct learning on mesh surfaces through their atlases of texture maps. The texture map encodes the parameterization from 3D to 2D domain, rendering not only RGB values but also rasterized geometric features if necessary. Since the parameterization of texture map is not pre-determined, and depends on the surface topologies, we therefore introduce a novel cross-atlas convolution to recover the original mesh geodesic neighborhood, so as to achieve the invariance property to arbitrary parameterization. The proposed module is integrated into classification and segmentation architectures, which takes the input texture map of a mesh, and infers the output predictions. Our method not only shows competitive performances on classification and segmentation public benchmarks, but also paves the way for the broad mesh surfaces learning.

1. Introduction

The 3D mesh is one of the most popular representation for 3D shape, which consists of an array of vertices and an array of face indices indicating the surface geometry. It could be augmented with the texture coordinate array and the texture map to render the color appearance of the mesh.

Learning on textured meshes is challenging: on the geometry side, the arrays of vertices and faces are permutable. The mesh geometry by its design, could be very adaptive, meaning a similar shape can be meshed in very different patterns, with irregular vertex densities and inconsistent triangle qualities. On the texture side, the parameterization of texture map could be arbitrary and still renders the same appearance, as long as the texture coordinate is in accordance with the content in texture map.

Existing methods that can directly or indirectly operate on textured meshes include 1) multi-view projection [33, 11, 10], which renders the RGBD images from all perspectives of the mesh. The multi-view images can be learned via individual 2D convolutional neural networks (CNNs) and the global feature is aggregated by view pooling. However, this is only feasible for small objects [39] where occlusion is not significant. When it comes to larger scenes where occlusion and view selection are non-trivial, their methods would degrade . 2) Volumetric grids can be obtained from meshes, where voxels store the color value and then are learned by 3D CNNs [39, 5, 22]. As mentioned in [6], the volumetric representation is memory-consuming and loses rich valuable image details, thus hindering the performance. 3) Point-based learning is a recent breakthrough [25, 27]. The colored point cloud can be easily sampled from the mesh. As the point cloud learning employs multi-layer perceptrons rather than convolutions, it takes significantly more parameters and thus the maximum number of points can be learned is far less than image pixels under the same environment. 4) Geometric deep learning [3] techniques can directly process on meshes via spectral analyses [4, 8, 16, 13, 40] or geodesic convolutions [21, 2, 23]. While their methods focus more on learning the intrinsic geometries for the dynamic correspondence task [1]. It's unclear how they can combine the texture or image for semantic learning.

In fact, the textured mesh is a self-contained combination between 2D texture and 3D shape. Some recent works are aware of this importance and perform joint learning on 2D images and 3D geometries [26, 6, 32], achieving improved performances. However, processing on images, rather than a single texture map, is redundant since the same 3D area is seen and processed multiple times. Besides, such bundle of meshes, images and cooresponding camera poses is difficult to acquire. With the popularity of 3D sensing techniques, it is more likely to obtain the standone textured mesh model.

To this end, we propose to perform direct learning on the textured mesh via its texture map. The texture map can include not only the color information as it usually does, but also arbitrary geometric features as long as they are rasterized in the map. Each pixel in texture map becomes a generic feature vector, encoding both color and geometric information. More importantly, the texture map is already in 2D domain, enjoying numerous benefits: 1) it can be learned via the standard CNNs, which can leverage the efficient designs from rich previous researches. 2) The texture map rasterization is analogous to sampling, and thus invariant to the irregular meshing. 3) The hierarchy of a 2D map is simply the image pyramid, making multi-scale learning (*e.g.*, for global feature extraction) easily achievable. 4) The texture map inherits the geodesic neighborhood of meshes. *i.e.*, neighbor pixels on texture map indicate their corresponding 3D points must be neighbors on mesh surface, whereas other (volumetric or point-based) representations discard this neighborhood information.

The practice of learning 3D meshes via 2D domain is seen in previous methods [31, 20], but their performances suffer from two crucial problems. The first one is distortion: unlike the generic texture map parameterization that segments the mesh to multiple atlases and pack them tightly in the texture map [17, 41], their parameterization conducts only one cut on the mesh and unfolds it to a complete 2D squared map. This inevitably introduces distortions – when the mesh is far from a sphere, the distortion could be unpredictably large [31]. The second problem is their networks are not invariant to parameterization, which is further determined by the cut on the mesh surface. The author suggests to try multiple cuts on testing stage [20], and select the most responsive one, which reduces usability of their method.

In this paper, we address two above problems. First, for the distortion problem, we do not unfold the 3D surface onto a complete 2D map. Instead, we segment the mesh to multiple atlases, project them to 2D domain and pack them tightly inside the texture map. By doing so, each atlas finds its best projection to minimize the distortion. Second, regarding the parameterization, the positions of atlases in texture map are unpredictable, depending on the packing algorithm. Each atlas is isolated, meaning the neighborhood information is taken apart when crossing the texture seams on mesh surface. To tackle these issues, we introduce the *cross-atlas convolution*. When the filter convolves across the boundary of an atlas, the pixel located outside of the atlas is redirected to the correct position, which corresponds to the actual neighbor point on mesh surface. This is done with a precomputed offset map. We have integrated the cross-atlas convolution into the classification and segmentation network architects, and verified their effectiveness on public benchmarks. Overall, our method enjoys several benefits:

1. We unlock a novel approach to mesh learning directly through their atlases of texture maps.
2. Our method addresses the distortion and the variance of parameterization problems in previous related methods [31, 20]. We also impose no restriction to the input mesh whereas they requires genus-0 meshes.
3. Our designed model is flexible and introduce no extra parameters: it can be trained on natural images, and tested on texture maps using our cross-atlas modules (see also Section 5.3).

2. Related Works

Deep learning on non-Euclidean geometric 3D data is an active and ongoing research topic. The mesh is one of the most commonly used representations in 3D vision and graphics, yet the irregularity of mesh makes it challenging to learn. We survey existing approaches in three categories.

Converting to regular structures The most straightforward approach is to convert the irregular mesh to regular data structure suitable for CNN processing. This can be done by projecting the mesh to multi-view 2D images, and then applying 2D CNN and view pooling to aggregate the global feature [33, 11, 10]. These methods show great performance on small object classification such as ModelNet [39] but may degrades when it comes to self-occluded objects or larger scenes where view selection is non-trivial. Another branch of methods convert the mesh to volumetric domains, and extract deep features from volumes by 3D convolutions [39, 5, 22]. The voxelization may introduce discretion errors and is highly memory-consuming. Using Octrees [28, 36] can abbreviate the resolution problem to some extent. Recently, some approaches combine both 2D views and 3D volumes and achieve even better results [26, 6].

Point cloud approaches Point cloud can be easily obtained by sampling on meshes. The irregular point cloud data can be learned from PointNet [25], and its extended hierarchical version [27]. They use combinations of multi-layer perceptions and pooling operations to achieve permutation invariance on the point set. Their insights also inspire several following works on point cloud learning in terms of improving the scalability and enhancing the local information of point cloud structure [18, 32, 14, 37]. In these methods, the neighborhood of a point is found by the radius-search or K nearest neighbors, which does not keep the original geodesic neighborhood on meshes. This could be a potential problem when the geodesic distance and Euclidean distance vary significantly in the mesh model.

Geometric deep learning on meshes The mesh can be learned by geometric deep learning techniques [3] for the non-rigid shape correspondence task. 1) If the mesh is seen as a graph, several works have proposed to apply the spectral analysis on the eigen decomposition of the Laplacian of mesh graph [4, 8] to establish dense correspondences between deformable shapes. A general limitation is the cross-domain generalization issue, which is later addressed by spectral transformers [40] to some extent. Dirac operator is an alternative to Laplacian operator which yields better stability in some scenarios [16]. 2) If seen as a manifold surface, the mesh can be learned by geodesic convolutions [21, 2, 23] on local patches, enjoying better generalization across domains than spectral methods. These techniques show great performance on non-rigid shape correspondence task but the receptive field of a polar filter is very small and thus it is unclear how to extract high-level features. Besides of using polar filter, a recent work [24] proposes to apply standard convolution on tangential projections of local patches and construct the hierarchy via mesh simplifications. 3) The third class of techniques applies global parameterization to the mesh and flatten it to 2D images [31, 20]. Our work belongs to this class, while the other two works are the GeometryImage [31] and the "flat-torus" method [20]. These methods enjoy common benefits derived from CNNs, but both of them unfold the mesh to a complete squared map, which induces considerable distortions. Besides, their methods require genus-0 input, otherwise they would crudely fill all topological holes. Regarding the parameterization, the one in GeometryImage [31] is not seamless, while the network in "flat-torus" method [20] is not invariant to the parameterization, depending on the cut of three chosen points on the mesh.

3. Mesh learning via Texture maps

This section illustrates the detailed procedure of mesh learning in the texture map space. In Section 3.1, we describe how the input texture map is generated from a pure mesh or a textured mesh. In Section 3.2, we introduce cross-atlas modules to recover the connectivities of separated atlases. In Section 3.3, we present network architectures for classification and semantic segmentation tasks.

3.1. Generating Texture Maps

In the preprocessing step, the input is a triangular mesh $\mathcal{M} = \{V, T\}$, where $V = \{v_i\}$ and $T = \{t_i\}$ correspond to the vertices and triangles respectively. If it's a polygon mesh we simply triangulate the faces. The mesh can be with or without textures. The output includes a texture map $(H \times W \times C)$ and an offset map $(H \times W \times 2k^2)$ for the network input, where k is the kernel size for convolution.

Mesh without textures: if a pure mesh is given (*e.g.*, Fig. 2(a) in ModelNet [39]), we create the UV parameterization by segmenting the mesh to multiple atlases (Fig. 2(c)), and then rasterize the geometric features to a $H \times W \times C$ texture map Fig. 2(d). Our goal is similar to previous parameterization algorithms [17, 41] aiming at how to find the cuts for minimized atlas distortions. Specifically, we first find a minimum set of dominant projection directions $\mathbf{P} = \{P_i \in \mathbb{R}^3\}$ such that the angle between each triangle normal $n(t_i)$ and its best projection vector P_{t_i} is less than a threshold, *i.e.*, $\angle(\overrightarrow{n(t_i)}, \overrightarrow{P_{t_i}}) < \tau_{angle}$. By default $\tau_{angle} = 40°$. When the angle $\angle(\overrightarrow{n(t_i)}, \overrightarrow{P_{t_i}})$ approaches $0°$, the projection is exactly tangential and has minimized

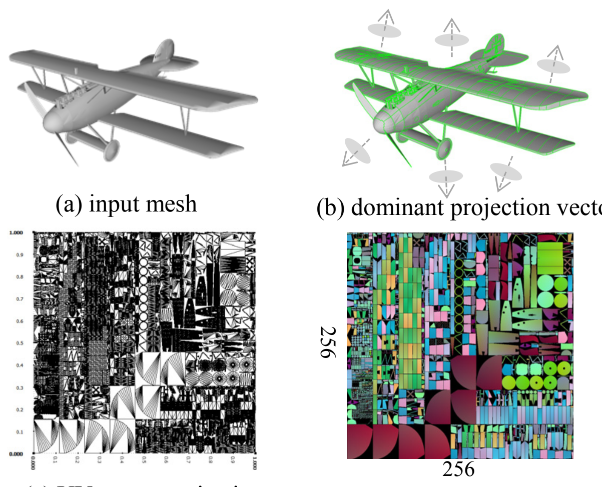

(a) input mesh
(b) dominant projection vectors
(c) UV parameterization by bin-pack
256
256
(d) rasterized XYZ texture map

Figure 1: For a pure mesh (a), we compute its dominant projections (b), and pack atlases in one UV map (c). The geometric feature (*e.g.*, vertex coordinate) is rasterized to a texture map (d).

local distortion [24]. After that, we cluster the projected triangles to atlases via connected components, and pack them into one squared map using bin-packing [15]. *(We leave the full algorithm in supplementary materials)* With this UV parameterization, we can rasterize the geometric feature of the mesh to a $H \times W$ resolution texture map $\mathbf{T}$. Note that the pixel in texture map $\mathbf{T}(x, y) = [f_1, f_2, ..., f_c]^T$ is a C-dimensional feature vector, instead of RGB values of the standard definition of "texture maps". The choice of geometric feature could be intrinsic features (*e.g.*, curvatures, heat kernel signatures) or extrinsic properties (*e.g.*, spatial coordinates, normals), or even concatenating multiple of them, depending on the specific task. For invalid regions we simply fill zeros. The output texture map is of size $H \times W \times C$, where $H \times W$ is the spatial resolution and C the feature channel.

Mesh with textures: if the mesh comes with textures, we still generate our own parameterization using above algorithm. The RGB color in original texture maps is an additional feature that can be concatenated to the geometric feature vector. We do not use the original parameterization as we need to ensure two criteria. 1) The parameterization should be *area-preserving*, which means the areas of mesh triangles and their projected areas in 2D maps are proportional. It ensures the receptive field of 2D convolution over the texture map corresponds to equal geodesic area over the mesh surface (also mentioned in [31]). 2) It should be *rotation-aligned*: the rotation of atlas in the original texture map could be arbitrary, making the later convolution suffer from rotation ambiguities. In our generated texture map,

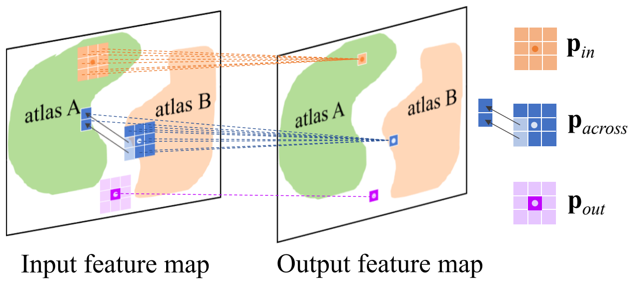


Figure 2: Illustration of the cross-atlas convolution. There are three situations when applying convolution over the texture map: $\mathbf{p}_{in}$ (standard convolution); $\mathbf{p}_{across}$ (cross-atlas convolution with offsets); $\mathbf{p}_{out}$(invalid pixel always = 0).

the rotation of each atlas is aligned with the negative Z-axis (usually the gravity direction) of the mesh coordinate, such that the visual content of atlas is upright.

Offset maps: Unlike natural images, atlases in texture map are discontinuous and locate unpredictably. It is not visually meaningful, and standard CNN approaches would not take effect on such input. To bridge the atlases and apply convolution across them, we also generate the offset map of size $(H \times W \times 2k^2$, k is the kernel size), which encodes the neighborhood information between atlas boundaries. We will describe how to create it in the next section together with the cross-atlas convolution.

3.2. Cross-atlas convolution via kernel offsets

The standard 2D convolution computes the output feature map $\mathbf{F}_o$ via the weighted sum of a k size regular patch over every pixel $\mathbf{p} = (x, y)$ at the input feature map $\mathbf{F}_i$:

$$\mathbf{F}_o(\mathbf{p}) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{F}_i(\mathbf{p} + \mathbf{p}_n) \cdot g(\mathbf{p}_n), \tag{1}$$

where $g(\cdot)$ is the kernel weight and $\mathcal{R} = \{\mathbf{p}_n\} = \{(x, y) : -\frac{k-1}{2} \leq x, y \leq \frac{k-1}{2}\}$ enumerates neighboring locations of the center pixel and indicates the receptive field.

This equation holds when the $k \times k$ local neighborhood is straightforward in natural images. For texture maps where atlases are isolated, two neighboring surface points on the mesh can lie on two separated atlases. To recover the original mesh geodesic neighborhood, the 2D convolution should be able to apply across atlases.

To this end, when rasterizing the texture maps in Section 3.1, we encode the atlas connectivity information by generating the corresponding offset map $\mathcal{R}_{offset}$ with the same spatial resolution $H \times W$ and channel length $2k^2$, where each pixel $\mathbf{p}$ in the offset map has a $k \times k$ patch indicating the offsets of x-axis and y-axis:

$$\mathcal{R}_{offset}(\mathbf{p}) = \{(\triangle x, \triangle y)\} \tag{2}$$

$$= \{\triangle \mathbf{p}_n\}, \quad |\mathcal{R}_{offset}(\mathbf{p})| = k^2. \tag{3}$$

These offset values are augmented to the standard neighboring locations $\mathcal{R} = \{\mathbf{p}_n\}$ and redirect the pixel to another location which corresponds to the actual mesh geodesic neighborhood, *i.e.*, $\mathcal{R} + \mathcal{R}_{offset}(\mathbf{p}) = \{\mathbf{p} + \mathbf{p}_n + \triangle \mathbf{p}_n\}$ are the geodesic neighboring locations of $\mathbf{p}$. The original equation in Eq. 1 becomes

$$\mathbf{F}_o(\mathbf{p}) = \sum_{\substack{\mathbf{p}_n \in \mathcal{R} \\ \triangle \mathbf{p}_n \in \mathcal{R}_{offset}(\mathbf{p})}} \mathbf{F}_i(\mathbf{p} + \mathbf{p}_n + \triangle \mathbf{p}_n) \cdot g(\mathbf{p}_n). \tag{4}$$

The offset value can be fractional, and we use bilinear interpolation to sample the pixel.

As illustrated in Fig. 2, we classify the texture map into three regions, denoted by pixels $\mathbf{p}_{in}, \mathbf{p}_{out}, \mathbf{p}_{across}$.

$\mathbf{p}_{in}$: When the convolution is applied over the inner-atlas region, the standard pixel neighborhood is corresponding to the mesh geodesic neighborhood, and thus no offset should be added: $\mathcal{R}_{offset}(\mathbf{p}_{in}) = \{(0, 0)\}$.

$\mathbf{p}_{out}$: When applying over the out-of-atlas region, this pixel value is invalid and should be kept isolated in order not to contaminate other pixels. We use offset values inverse to $\mathcal{R}$, *i.e.*, $\mathcal{R}_{offset}(\mathbf{p}_{out}) = -\mathcal{R}$ and $\mathbf{p}_{out} + \mathbf{p}_n + \triangle \mathbf{p}_n = \mathbf{p}_{out}$. This ensures $\mathbf{F}_o(\mathbf{p}_{out}) = \mathbf{F}_i(\mathbf{p}_{out}) = 0$ throughout the network.

$\mathbf{p}_{across}$: When the standard $\mathcal{R}$ is just across the border of an atlas, we add the precomputed offset values to its standard locations, so $\mathbf{p}_{across} + \mathbf{p}_n + \triangle \mathbf{p}_n$ should just locate at the true mesh geodesic neighborhood.

Therefore, only pixels $\mathbf{p}_{across}$ need to compute their offset values. The pixels $\mathbf{p}_{across}$ are determined by the kernel size: if we place a k-size filter kernel within the atlas at $\mathbf{p}$ and there are some pixels of this kernel locate out of the atlas, then $\mathbf{p} = \mathbf{p}_{across}$. For a 3×3 kernel as an example, we compute the offsets for 1-ring atlas boundary pixels.

To compute the offset value for a center pixel $\mathbf{p}$ regarding its neighbor pixel $\mathbf{p} + \mathbf{p}_n$ lying out of atlas, we rasterize the vertex coordinates to a map with the original resolution $H \times W$, and thus can instantly query via this map the pixel $\mathbf{p}$ corresponds to the 3D point $\mathbf{X}$ on mesh surface. Then we use the Fast Marching [34] to search the geodesic neighbor point of $\mathbf{X}$ along $\overrightarrow{\mathbf{p}_n}$ direction, yielding the point $\mathbf{X}'$. The $\mathbf{X}'$ finds its corresponding texture coordinates $\mathbf{p}'$ on the texture map. Finally $\triangle \mathbf{p} = \mathbf{p}' - \mathbf{p} - \mathbf{p}_n$ is the offset value.

Note that unlike standard convolution on natural images can add paddings to image boundaries, the cross-atlas convolution has no "image boundaries" – if the mesh surface is water-tight, every pixel in texture map can find its neighborhood. If it is an open mesh and the pixel exactly corresponds to the mesh boundary that finds no neighborhood, it simply fills zero value (analogous to the zero padding effect). In all, the spatial resolution of feature map can only be decreased by the strides of convolution or pooling.

Deconvolution In semantic segmentation tasks where the feature map is finally up-sampled to the input resolution, deconvolution (or transposed convolution) is a popular approach [19]. Essentially, deconvolution can be decomposed into 1) scattering pixels from the sparse feature map to a dense feature map with even strides and 2) applying convolution over this map. We can simply replace this convolution with the cross-atlas version if applied on texture maps.

Pooling The standard pooling operation is often used to reduce dimensions of feature maps. The pooling type can be *max*, *average* or *sum* operation.

$$\mathbf{F}_o(\mathbf{p}_o) = pool_{k\times k}(\mathbf{F}_i(\mathbf{p}), \mathbf{F}_i(\mathbf{p}+\mathbf{p}_n), ...), \forall \mathbf{p}_n \in \mathcal{R} \quad (5)$$

where $\mathbf{p}_o$ is the location at the lower spatial resolution output feature map. For $k = 2$ as an example, $\mathbf{p}_o \cdot 2 = \mathbf{p}$. It computes the pooling value from every 2×2 patch.

In cross-atlas pooling, the behavior is similar as cross-atlas convolution by replacing the standard image pixel neighborhood with the mesh geodesic neighborhood:

$$\begin{aligned}\mathbf{F}_o(\mathbf{p}_o) =& pool_{k\times k}(\mathbf{F}_i(\mathbf{p}), \mathbf{F}_i(\mathbf{p}+\mathbf{p}_n+\triangle\mathbf{p}_n), ...),\\ &\forall \mathbf{p}_n \in \mathcal{R}, \triangle\mathbf{p}_n \in \mathcal{R}_{offset}(\mathbf{p}).\end{aligned} \quad (6)$$

Hierarchies For a specific $H \times W$ dimension feature map and the kernel size k, the corresponding offset map is unique. In the CNN pipeline, the spatial dimension of the feature map keeps changing by in-network upsampling and downsampling. Therefore, the corresponding offset maps for all possible feature map dimensions need to be computed beforehand. Generally, we compute the hierarchy of offset maps by rasterizing a pyramid of resolutions, where each lower level is half in width and height of the upper level. The offset map with larger kernel size can be reused in the operation with smaller kernel. *e.g.*, we can take the central 3×3 from 5×5 offset locations. Note that in lower spatial dimensions, some small atlases might disappear as they are too small to be rendered, but their feature information does not lose – it is absorbed by pixels in other atlases via cross-atlas convolution.

There are some similarities between our design and the deformable convolution [7] as we both leverage offset values to the convolution. However, our offset map is precomputed by the atlas neighborhood so as to recover the geodesic receptive field, while offset values in [7] are all trainable for more a flexible receptive field in objection detection problem.

3.3. Network architecture

We have integrated the cross-atlas convolution into classification and semantic segmentation architectures. The comprehensive architecture is illustrated in Fig. 3.

Classification A generic classification network inputs an image or a feature map, bypassing multiple layers of convolution and pooling. To recover the high-level feature, a common practice to reduce the spatial dimension (by strides) and increase the channel dimension (by convolution). Then it is flattened to a 1D global feature vector, followed by fully connected (FC) layers and softmax, yielding the class label.

To apply classification on the (textured) mesh, we first convert it to the $H_1 \times W_1 \times C_1$ input feature map (Section 3.1) and its corresponding offset map. Then, we replace the standard convolution and pooling with our cross-atlas versions. After bypassing n layers of cross-atlas convolutions, it obtains a feature map with dimensions $H_n \times W_n \times C_n$. At this moment, we cannot simply flatten it to a 1D feature vector like the standard network does, because the spatial locations of pixels in this feature map are permutable when atlases are packed in a different way (*e.g.*, swap the atlas A and B in Fig. 2). Inspired by PointNet [25], we regard each valid pixel in the $H_n \times W_n \times C_n$ feature map is a permutable "point". We reshape it to $(H_n \times W_n) \times C_n \times 1$, *i.e.*, each C_n-channel pixel is expanded to one row. Then we apply multi-layer perceptrons (MLPs) and row-wise max pooling to obtain a m-channel 1D feature vector. Finally fully connected layers is applied and outputs the n_{class}-channel 1D feature vector indicating the probabilities of classes. We use ReLU as our activation function. In general we use 4∼19 convolutional/pooling layers, 5∼7 MLP layers and 2∼3 FC layers. The specific numbers of layers and the feature map dimension vary in different tasks.

Segmentation A segmentation task is a pixel-wise classification task. Its former part is similar to classification which yields the $H_n \times W_n \times C_n$ feature map via several layers of convolution and pooling. Unlike classification which extracts the global feature in the later modules, it upsamples the feature map to an original resolution annotation map $H_1 \times W_1 \times C_{n_{class}}$. Here, we leverage the deconvolution operations in FCN [19] as our up-sampling layers, and replace their convolution with our cross-atlas version. We add three skip connections between the convolution and deconvolution layers.

4. Understanding the cross-atlas convolution

In this section, we discuss three important properties of the proposed method. 1) It is robust to irregular meshes. 2) It is invariant to parameterization. 3) The receptive field follows the mesh geodesic distance.

Robust to irregular meshes By the design of mesh, a similar shape can be meshed by very different patterns, but it is not expected different meshing would cause inconsistent results. To tackle this, the texture map rasterization

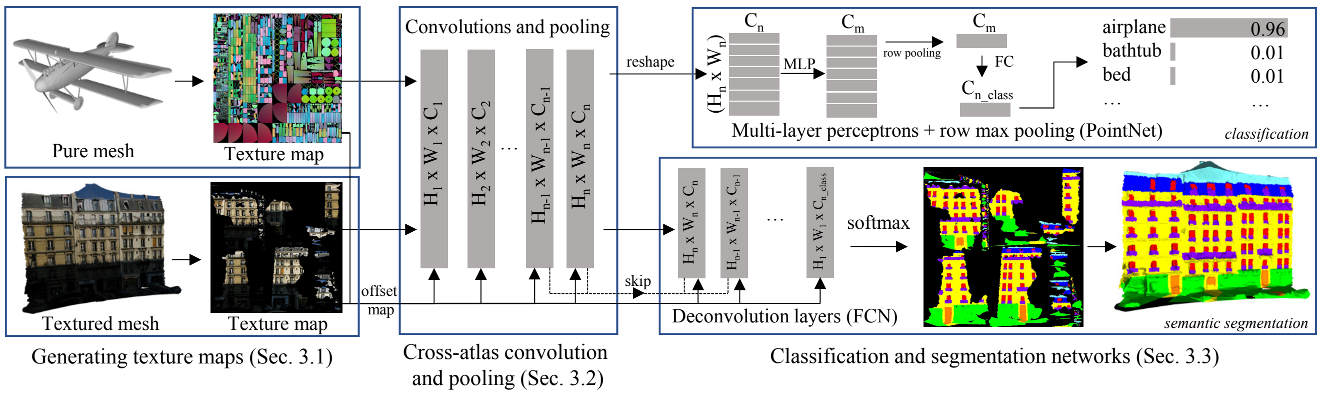


Figure 3: The classification and segmentation network architectures of our method. We use 4∼19 convolutional/pooling layers, 5∼7 MLP layers, 2∼3 FC layers and 4∼8 deconvolution layers. The concrete number of layers varies in specific tasks.

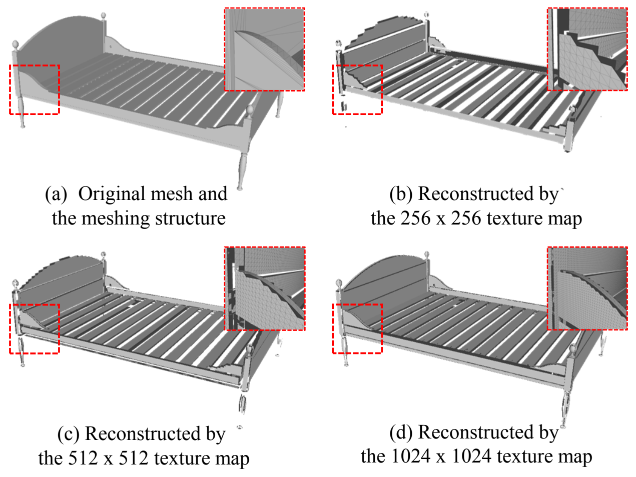

(a) Original mesh and the meshing structure
(b) Reconstructed by the 256 x 256 texture map
(c) Reconstructed by the 512 x 512 texture map
(d) Reconstructed by the 1024 x 1024 texture map

Figure 4: The original mesh (a) and reconstructed meshes from texture maps of different resolution (b)(c)(d). Our method is invariant to irregular mesh input: applying convolution over the texture map is analogous to applying over the vertices of these reconstructed meshes.

in our first step can be deemed as the procedure of sampling the surface points to pixels. If we connect every 4-neighborhood of pixels to edges, it reconstructs a regular mesh as shown in Fig. 4. Therefore, applying convolution over the texture map is analogous to applying over the vertices of these regular meshes. When a lower resolution (256×256) texture map is used, thin structures such as bed legs are missing (Fig. 4(b)), whereas higher resolution (1024×1024) involves more geometric details (Fig. 4(d)).

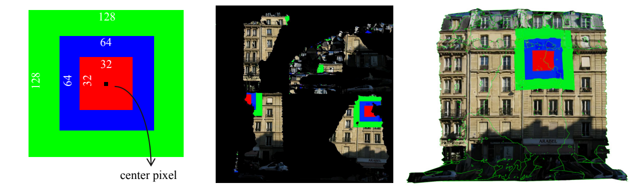
Invariant to parameterization When generating the texture map (Section 3.1), the parameterization is unpredictable. Essentially, it includes two uncertainties: 1) *how atlases are cut* and 2) *how atlases are packed*. Here we explain why our method is invariant to these two issues.

Atlas cutting: remind that in Section 3.1, we set an angle threshold τ_{angle} to tradeoff the amount of atlases and their distortions – setting τ_{angle} too small may lead to many fragmentary atlases, and vice versa. Despite how they are cut, their neighborhood information is encoded in the offset map, informing the network to convolve across discontinuous atlases. With a proper value of τ_{angle}, we can assume the distortion is relatively small. Although such a small distortion of atlas still has an unpredictable variance, it is analogous to the practice in augmenting training data where a random mild transform is applied to training samples, which helps prevent from over-fitting.

Atlas packing: the only uncertainty in atlas packing is their spatial locations (*i.e.*, translations), as we have already aligned the rotation to Z-axis and preserve the scale of atlas area. Our network architectures are designed to be invariant to the translation of atlas. Imagine if we exchange the position between atlas A and B in Fig. 2: regarding the segmentation task, the network is fully convolutional, which is translation equivalent. Therefore, altering the translation of atlas shall yield consistent predictions if the correct offset map is given. As for classification task, the global feature is extracted by PointNet [25], which disambiguates the spatial location variance of the last convolutional feature map ($H_n \times W_n \times C_n$), thus the global feature is again invariant to the translation of atlas.

Note that the rotation alignment of Z-axis only disambiguates X and Y axises rotational varieties. The in-plane rotation perpendicular to Z-axis is still ambiguous, which is a common problem in many previous works [22, 31, 25]. Likewise, we alleviate this problem by augmenting the training data with randomly rotations along Z-axis.

Receptive fields Although the convolution is applied over 2D texture maps, its receptive field follows the mesh geodesic distance. This is done by using the offset map



(a) The illustration of three range receptive fields (b) The receptive field on 512 x 512 texture map (c) The corresponding receptive field on the textured mesh

Figure 5: Illustration of the receptive field. If we dilate the receptive field from a center pixel on the texture map and redirect when reaching to atlas boundaries, the field would be separated in severel atlases (b). Its corresponding field on textured mesh is approximately the geodesic field.

to redirect pixel locations. Fig. 5 illustrates this behavior: we color-code the receptive fields of 32×32, 64×64, 128×128 of a center pixel in red, blue and green respectively. As we dilate a pixel to a block-wise field in the texture map (redirecting to the offset location when reaching to atlas boundaries), the field is actually separated in several atlases (Fig. 5(b)). On the contrary, its corresponding textured mesh shows the block-wise geodesic receptive field over its surface (Fig. 5(c)). Note that the receptive field on the mesh is not strictly following the geodesic distance due to the distortion in atlas projections, but it is a good approximation given the distortion is constrained by τ_{angle} in Section 3.1.

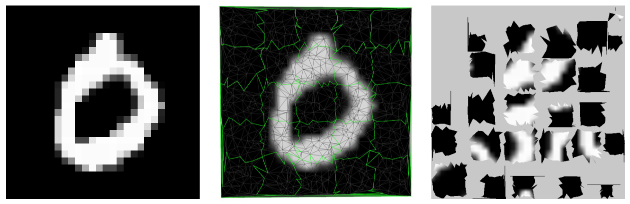
5. Experiments

We implement the texture map generation (Section 3.1) in C++, and the network (Section 3.3) using Tensorflow. Our method is evaluated on three benchmarks, namely MeshMNIST dataset [16], ModelNet [39] and Ruemonge2014 dataset [29].

5.1. MeshMNIST

The original MNIST dataset contains handwritten digit images at 28×28 resolution (Fig. 6(a)). The MeshMNIST [16] converts the digit image to a triangulated mesh by mapping the intensity to the height-field of the mesh. Although MeshMNIST is first used in [16] for the evaluation of their generative model, we conduct a classification experiment using meshes.

We inversely map the height-field back to intensity (Fig. 6(b)), segment the mesh and pack atlases in the texture map (Fig. 6(c)). The texture map and its corresponding offset map are fed into our network to train a classifier. There are a total number of 60,000 training samples and 10,000 testing samples. The network follows the classification architecture in Section 3.3 with 4 conv. layers, 5 MLP layers and 3 FC layers. We use cross entropy as the loss function. The network is trained with 100 epochs with 100 batch size and AdamOptimizer (learning rate = 0.001).



(a) MNIST sample (b) MNIST textured mesh (c) The texture map

Figure 6: The MNIST data sample (a) is texture mapped to a mesh (b), and its texture map is not visually recognizable as in (c).

Ablation study To better validate the effectiveness of each component, we start with the standard LeNet5 on original MNIST dataset, and then replace with our components one by one. Table 1 shows the comparisons of using differnet modules and dataset.

Conv. layers	FC layers	Dataset	Acc.
Standard conv.	Standard FC	MNIST	99.2%
		MeshMNIST	30.8%
Standard conv.	MLP+max pooling+FC	MNIST	96.8%
		MeshMNIST	88.6%
Cross-atlas conv.	MLP+max pooling+FC	MNIST	96.8%
		MeshMNIST	96.5%

Table 1: The testing accuracy of different combinations of conv. layers, FC layers and datasets. Our method achieves 96.5% accuracy on MeshMNIST dataset.

The standard LeNet5 consists of 4 conv. layers and 3 FC layers, achieving 99.2% on the original MNIST dataset, and 30.8% if directly applied on texture maps. If we replace the standard FC layers with the MLP+max pooling+FC modules, the accuracy drops by 2.4% on standard image, and increases to 88.6% on texture maps, due to the loss of spatial information and meaning achieving translational invariance. If we further integrates the cross-atlas convolution and pooling in the network, the accuracy on texture maps increases to 96.5%.

5.2. ModelNet classification

We evaluate our approach for 3D shape classification task on the two versions of the large scale Princeton ModelNet dataset [39]: ModelNet40 and ModelNet10, which consist of 40 and 10 classes respectively. We follow the same experiment setting as in [39]. The vertex coordinates are rasterized to the texture map at 256×256 resolution for the network input. As our texture bin-packing algorithm is randomized, we generate the input texture map by running multiple times to augment the training data, as well as randomly rotating the model along Z-axis. We use almost the same network and parameters as in Section 5.1, with 8 conv. layers and 5 batch size.

Table 2 shows the classification accuracy in testing. We have listed representative state-of-the-art methods of using

Method	Input	ModelNet40 accuracy	ModelNet10 accuracy
MVCNN [33]	image	90.1%	-
RotationNet [11]	image	97.37%	98.46%
VoxNet [22]	volume	83%	92%
MVCNN+MultiRes [26]	img.+vol.	91.4%	-
PointNet [25]	point	89.2%	-
PointNet++ [27]	point	91.9%	-
SHR [12]	mesh	68.2%	79.9%
GeometryImage[31]	mesh	83.9%	88.4%
Ours	mesh	87.5%	91.2%

Table 2: The overall classification accuracies of the multi-view image, volume, point and mesh representations.

different geometry representations, namely multi-view images, volumes, points and meshes. Our method achieve better results than the other two mesh-based methods [12, 31], while it is overall not as competitive as multi-view image projection or point-based methods. We perceive the CAD mesh has a signification problem that hinders the performance of mesh-based methods: some structure in the mesh model should have been topologically connected, but in fact they are just overlaid together. This makes the geodesic receptive field erroneous. On the contrary, multi-view image projection or point-based method can avert the problem.

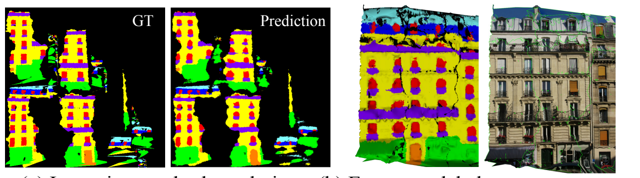
5.3. Ruemonge2014 segmentation

Method	triangle accuracy	class avg. IoU
Riemenschneider [29]	-	41.92%
Gadde [9]	-	63.7%
Ours (texture, w/o fusion)	72.28%	65.24%
Ours (texture, w/ fusion)	84.90%	75.34%
Ours (image, w/ fusion)	76.02%	72.38%
Ours (texure+image, w/ fusion)	85.63%	75.67%

Table 3: The evaluation of mesh-based semantic segmentation on the Ruemonge2014 dataset [29]

We evaluate the semantic segmentation performance on the Ruemonge2014 dataset [29], which consists of 428 high resolution images capturing the facade along the street, as well as registered camera poses and reconstructed meshes by multi-view stereo. The images and mesh come with ground truth semantic labels of seven classes, namely the window, wall, balcony, door, roof, sky and shop. The dataset is separated into training samples and testing samples. To evaluate, the class-averaged intersection over union (IoU) are per-triangle label accuracy are used.

To generate the textured mesh, we run the multi-view texturing algorithm [35] using the given undistorted images and camera poses. Then, we segment the training mesh part into 100 overlapped mesh segments, each consists of a 512×512 texture map. We integrate our cross-atlas convolution modules into the fully convolutional networks (FCN) [19] with VGG19 [30]. This network exactly corresponds to our generic segmentation architecture in Fig. 3: the VGG part corresponds to the cross-atlas convolution. The up-sampling layers are identical to the ones used in the FCN. We replace all 3×3 convolutions and 2×2 pooling in VGG and the deconvolution in FCN with ours cross-atlas version and apply on the texture maps with offset maps. Here, the RGB values are used in the texture map channel. We train the network with 100 epochs with AdamOptimizer (learning rate = 1e-4) and 10 batch size.



(a) Inconsistent atlas boundaries between GT and prediction (b) Erroneous labels at texture seams and small atlases

Figure 8: The problem of semantic segmentation on texture maps: the atlas boundaries are not alway consistent with ground truth (a), leading to erroneous labels at texture seams (b).

Fig. 8 shows one testing result. We notice that the predicted annotation map does not have exactly the same atlas boundaries as the ground truth, and some small atlases are even filtered out (Fig. 8(a)). This problem is analogous to "over-rounded" artifacts annotation maps of natural images, whereas in texture maps the atlas may slightly dilate or erode. This issue leads to erroneous labels at texture seams when mapping the annotation map to the mesh (Fig. 8(b)).

Fusion in testing stage Inspired by the multi-scale testing trick used in common semantic segmentation methods, we conduct the testing on individual and overlapped parts, and fuse them afterwards: each triangle label is finally determined by the label of majority pixels. This improves the result significantly, from 65.24% to 77.34% IoU as shown in Table 3. Overall, our method surpass the second place in the Ruemonge2014 challenge benchmark [29] by a large margin. Fig. 7 shows the qualitative results.

Train on images, test on textured meshes With cross-atlas convolution, the network can be trained and tested on texture maps with corresponding offset maps. One may wonder if we can also train on normal images and test on texture maps – we can regard the image as a one-atlas "texture map" with zero offsets. To validate, we take street view images and their corresponding ground truth given in the dataset for training. The testing accuracy turns out decent, achieving 76.02% triangle label accuracy and 72.38% IoU. If we combine the normal images and texture maps for training, the result is by 0.33% marginally better than only

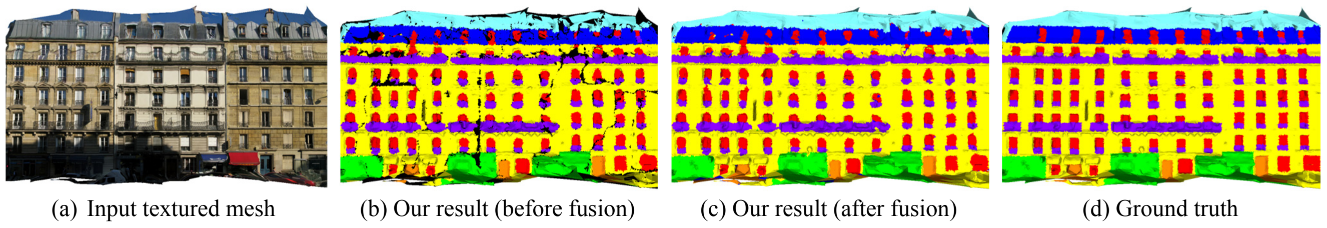


Figure 7: The qualitative comparison of semantic segmentation results on the Ruemonge2014 dataset [29].

training on texture maps. This shows our method is flexible in terms of the training data – it can be trained on normal images and test on texture maps of meshes with the corresponding content in original images.

6. Conclusion

We have proposed a parameterization invariant approach to textured mesh learning. The key to this method is the cross-atlas convolution which recovers the mesh geodesic receptive field although it actually convolves on 2D domain. Our work unlocks the possibility for direct classification and semantic segmentation on textured meshes via their texture maps, whereas in previous methods this only can be done by multi-view image projection or point cloud sampling from meshes.

Limitations and future works The biggest limitation of the proposed method is the texture map requires the rotation of atlases should be aligned to upright, so the convolution is not suffer from rotation variances of two axises. For the plane perpendicular to Z-axis, there's still inplane rotation ambiguity, which we resolve it via augmenting the training data. Some related works use polar convolution [21, 2, 23] or pooling after multi-directional convolution [24] to achieve fully rotational invariance, as their shape correspondence task is more sensitive to geometries. We do not use them in our current framework but it could be a potential improvement [38]. Besides, object detection over textured meshes is another potential task, which is well-addressed in 2D images but rarely targets to 3D models.

References

[1] F. Bogo, J. Romero, M. Loper, and M. J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Piscataway, NJ, USA, June 2014. IEEE. 1

[2] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 3189–3197, 2016. 1, 2, 9

[3] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. 1, 2

[4] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013. 1, 2

[5] A. Dai, A. X. Chang, M. Savva, M. Halber, T. A. Funkhouser, and M. Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, volume 2, page 10, 2017. 1, 2

[6] A. Dai and M. Nießner. 3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018. 1, 2

[7] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. *CoRR, abs/1703.06211*, 1(2):3, 2017. 5

[8] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016. 1, 2

[9] R. Gadde, V. Jampani, R. Marlet, and P. V. Gehler. Efficient 2d and 3d facade segmentation using auto-context. *IEEE transactions on pattern analysis and machine intelligence*, 40(5):1273–1280, 2018. 8

[10] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri. 3d shape segmentation with projective convolutional networks. In *Proc. CVPR*, volume 1, page 8, 2017. 1, 2

[11] A. Kanezaki, Y. Matsushita, and Y. Nishida. Rotationnet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints. In *Proc. 2018 IEEE Conf. on Computer Vision and Pattern Recognition*, pages 5010–5019, 2018. 1, 2, 8

[12] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3 d shape descriptors. In *Symposium on geometry processing*, volume 6, pages 156–164, 2003. 8

[13] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 1

[14] R. Klokov and V. Lempitsky. Escape from cells: Deep kdnetworks for the recognition of 3d point cloud models. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 863–872. IEEE, 2017. 2

[15] R. E. Korf. A new algorithm for optimal bin packing. In *AAAI/IAAI*, pages 731–736, 2002. 3

[16] I. Kostrikov, Z. Jiang, D. Panozzo, D. Zorin, and B. Joan. Surface networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*, 2018. 1, 2, 7

[17] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. In *ACM transactions on graphics (TOG)*, volume 21, pages 362–371. ACM, 2002. 2, 3

[18] Y. Li, R. Bu, M. Sun, and B. Chen. Pointcnn. *arXiv preprint arXiv:1801.07791*, 2018. 2

[19] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 5, 8

[20] H. Maron, M. Galun, N. Aigerman, M. Trope, N. Dym, E. Yumer, V. G. Kim, and Y. Lipman. Convolutional neural networks on surfaces via seamless toric covers. *ACM Trans. Graph*, 36(4):71, 2017. 2, 3

[21] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015. 1, 2, 9

[22] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015. 1, 2, 6, 8

[23] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR*, volume 1, page 3, 2017. 1, 2, 9

[24] H. Pan, S. Liu, Y. Liu, and X. Tong. Convolutional neural networks on 3d surfaces using parallel frames. *arXiv preprint arXiv:1808.04952*, 2018. 3, 9

[25] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017. 1, 2, 5, 6, 8

[26] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016. 1, 2, 8

[27] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017. 1, 2, 8

[28] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 3, 2017. 2

[29] H. Riemenschneider, A. Bódis-Szomorú, J. Weissenberg, and L. Van Gool. Learning where to classify in multi-view semantic segmentation. In *European Conference on Computer Vision*, pages 516–532. Springer, 2014. 7, 8, 9

[30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 8

[31] A. Sinha, J. Bai, and K. Ramani. Deep learning 3d shape surfaces using geometry images. In *European Conference on Computer Vision*, pages 223–240. Springer, 2016. 2, 3, 6, 8

[32] H. Su, V. Jampani, D. Sun, S. Maji, E. Kalogerakis, M.-H. Yang, and J. Kautz. SPLATNet: Sparse lattice networks for point cloud processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2530–2539, 2018. 1, 2

[33] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015. 1, 2, 8

[34] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. In *ACM transactions on graphics (TOG)*, volume 24, pages 553–560. Acm, 2005. 4

[35] M. Waechter, N. Moehrle, and M. Goesele. Let there be color! large-scale texturing of 3d reconstructions. In *European Conference on Computer Vision*, pages 836–850. Springer, 2014. 8

[36] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):72, 2017. 2

[37] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018. 2

[38] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. Harmonic networks: Deep translation and rotation equivariance. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2017. 9

[39] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 1, 2, 3, 7

[40] L. Yi, H. Su, X. Guo, and L. J. Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *CVPR*, pages 6584–6592, 2017. 1, 2

[41] K. Zhou, J. Synder, B. Guo, and H.-Y. Shum. Iso-charts: stretch-driven mesh parameterization using spectral analysis. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 45–54. ACM, 2004. 2, 3