Picasso: A CUDA-based Library for Deep Learning over 3D Meshes

Huan Lei   Naveed Akhtar   Ajmal Mian

The University of Western Australia

huan.lei@research.uwa.edu.au, {naveed.akhtar,ajmal.mian}@uwa.edu.au

Code: https://github.com/hlei-ziyan/Picasso

Abstract

We present Picasso, a CUDA-based library comprising novel modules for deep learning over complex real-world 3D meshes. Hierarchical neural architectures have proved effective in multi-scale feature extraction which signifies the need for fast mesh decimation. However, existing methods rely on CPU-based implementations to obtain multi-resolution meshes. We design GPU-accelerated mesh decimation to facilitate network resolution reduction efficiently on-the-fly. Pooling and unpooling modules are defined on the vertex clusters gathered during decimation. For feature learning over meshes, Picasso contains three types of novel convolutions namely, facet2vertex, vertex2facet, and facet2facet convolution. Hence, it treats a mesh as a geometric structure comprising vertices and facets, rather than a spatial graph with edges as previous methods do. Picasso also incorporates a fuzzy mechanism in its filters for robustness to mesh sampling (vertex density). It exploits Gaussian mixtures to define fuzzy coefficients for the facet2vertex convolution, and barycentric interpolation to define the coefficients for the remaining two convolutions. In this release, we demonstrate the effectiveness of the proposed modules with competitive segmentation results on S3DIS. The library will be made public through github.

1. Introduction

Data in computer vision vary commonly from homogeneous format in 2D projective space (e.g. images, videos) to heterogeneous format in 3D Euclidean space (e.g. point clouds, meshes). The success of convolutional feature learning on homogeneous data [21, 28, 37, 38, 46, 47, 50, 55] has sparked research interest in geometric deep learning [6, 7, 14, 26, 54], which aims for equally effective feature learning on heterogeneous data. Due to the rise of autonomous driving and robotics, 3D deep learning has now become an important branch of the geometric research direction. Compared to 3D point clouds, 3D meshes convey richer geometric information about the object surface and topology. Yet, the heterogeneous facet shapes and sizes combined with unstructured vertex locations make its adaption to deep learning more difficult as compared to point clouds. This is why most approaches address real-world

3D scene understanding via convolutions on point clouds [31, 32, 33, 35, 43, 44]. However, point clouds still lack in preserving the structural details that are easily represented by meshes.

There are a few works that learn features from meshes, but they are largely constrained to *shape analysis* on small synthetic models [5, 20, 39, 41, 45, 68]. These methods either apply convolutions throughout the network to a single mesh resolution (the input), or exploit inefficient CPU-based algorithms to decimate the mesh [16, 17, 51, 71]. However, non-hierarchical network configurations and slow network coarsening are both problematic while dealing with real-world meshes because of their large-scale nature. This calls for mesh simplification methods that are fast and amenable to deep learning for the real-world applications.

We present a GPU-accelerated mesh simplification algorithm to facilitate the exploration of hierarchical architectures on meshes. The proposed method is not only fast in decimating small-scale watertight meshes from CAD modelling [4, 9, 11, 36], but is also efficient in simplifying large-scale unstructured real-wold meshes [2, 8, 12, 22]. We perform all computations in parallel on GPU, except for the grouping of vertex pairs to be contracted. Meanwhile, to increase its compatibility with modern deep learning modules such as normalization [3, 24, 60, 66], we contract vertex clusters by controlling the desired vertex size of the decimated mesh. This also advances mesh-based modules to be exploited in conjunction with the various point cloud based modules [52]. Our algorithm is able to reduce the number of mesh vertices by half in each iteration. Figure 1 compares the runtime of our method with two well-founded decimation methods, VC [51] and QEM [17]. Notice that our method is $30\times$ faster than QEM. During the simplification, we record all vertex clustering information into a 1-D tensor. Based on this tensor, we also define max, average and weighted poolings, as well as unpooling.

Earlier attempts for convolution on meshes [5, 39, 41] explored local patch operators in hand-crafted coordinate systems. The development of spatial graph convolutions has led recent methods [20, 45, 52] to predominantly consider (*triangular*) mesh as a special graph and convolve features of each vertex from their geodesic k-ring neighborhood.
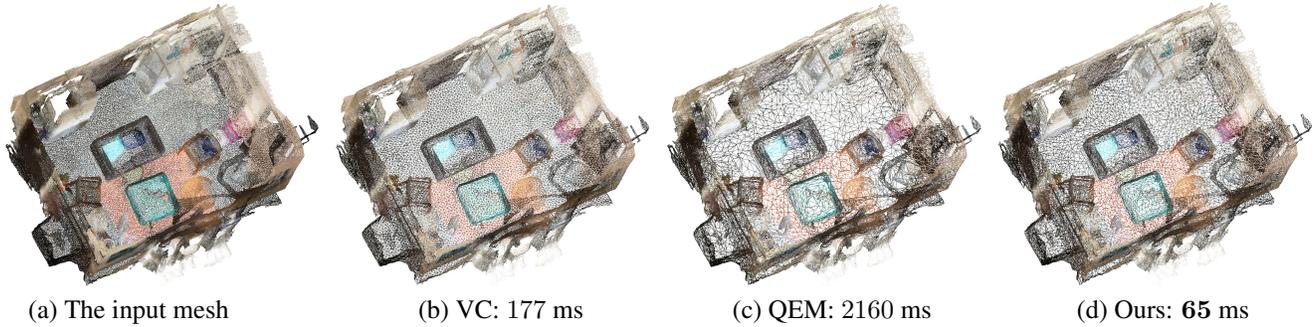
| (a) The input mesh | (b) VC: 177 ms | (c) QEM: 2160 ms | (d) Ours: **65** ms |

Figure 1: Runtime comparison of mesh simplification. The input mesh consists of $115,114$ vertices and $231,293$ facets. Different methods simplify it to similar mesh sizes. The number of vertices and facets in the decimated meshs are $41,483$ and $85,424$ for vertex clustering (VC) [51], $41,133$ and $83,050$ for the quadric error metrics (QEM) [16, 17]; and $41,449$ and $83,051$ for our GPU-accelerated algorithm. The runtime of VC, QEM, and our method are respectively 177 ms, 2160 ms, and 65 ms. For VC and QEM, we utilize their popular implementations in Open3D [71]. For better visualization, we show wireframes of the meshes only. Best viewed in color and enlarged.

In contrast, we study mesh as a set of vertices and facets, following its natural geometric structure. To learn features of each vertex, we aggregate their context information from the adjacent facets. We refer to the resulting operation as *facet2vertex* convolution. Lei *et al.* [32] showed that fuzzy mechanism makes the convolutional network robust to point density variation. Hence, we further exploit fuzzy coefficients in the facet2vertex convolution. Due to the fact that facet normals are distributed strictly on the surface of a unit sphere, i.e. $\mathbf{S}^3 = \{\mathbf{x} \in \mathbf{R}^3 : \|\mathbf{x}\| = 1\}$, we associate learnable filter weights to Gaussian mixtures defined on $\mathbf{S}^3$. The parameters of the Gaussian components can optionally be kept fixed or trainable within the network in our library. On the other hand, to learn features of the mesh facets, we introduce *vertex2facet* and *facet2facet* convolutions. The former propagates features from the vertices of a facet to the facet itself, while the latter is applied when facets of the input mesh are rendered with textures. We incorporate fuzziness into these two convolutions using barycentric interpolation. The three proposed convolutions altogether enable flexible vertex-wise and facet-wise feature learning on the mesh.

We provide CUDA implementations for all the above mentioned modules and organize them into a self-contained library, named Picasso[1], to facilitate deep learning over the unstructured real-world 3D meshes. We note that meshes and point clouds are tightly bonded together, and it is more desirable to extract features from the two data modalities cooperatively rather than individually or competitively. DCM-Net [52] also validates this argument. For this reason, we additionally incorporate all the point cloud modules from Lei *et al.* [32, 33] in our library (with author permission). In this maiden release of our library, we demonstrate promise of its proposed modules with competitive segmentation results on S3DIS [2]. The segmentation network is

abbreviated as PicassoNet for consistency. We summarize the main contributions of our work below:

- We present a fast GPU-accelerated 3D mesh decimation technique to reduce mesh resolution on-the-fly. A public implementation with complementary CUDA-based pooling and unpooling operations is provided.

- We propose three novel convolution modules, i.e. *facet2vertex*, *vertex2facet*, *facet2facet*, to alternatively learn vertex-wise and facet-wise features on a mesh. Diverging from existing methods, we do not rely on restrictive treatment of mesh as an undirected graph with edges. Instead, it's a geometric structure composed of vertices and facets for our modules, which is also a more conducive representation for the digital devices.

- With this paper, we release Picasso — a self-contained library for deep learning over unstructured real-world 3D meshes, along with synthetic watertight meshes. The provided anonymous github link will be made public for the broader research community.

2. Related Work

Convolutions on point clouds: 3D-CNNs [15, 18, 40, 49, 67] are the most intuitive solutions of transferring CNNs from images to point clouds. A few methods also explore similar regular-grid kernels in a transformed data domain [56, 57]. The permutation invariant networks exploit multi-layer perceptrons (MLPs) and max-pooling to learn features from point clouds [27, 34, 43, 44, 48, 53, 65]. They demonstrate the effectiveness of taking point coordinates xyz as input features. Graph-based networks allow convolutions to be performed in either spectral or spatial domain. However, the mandatory alignment of different graph Laplacians makes the application of spectral graph convolutions to point clouds more difficult than spatial graph convolutions

---

[1]Paying homage to Pablo Picasso for cubism in paintings.

[70]. As a pioneering work, ECC [54] exploited dynamic filters [13] to analyze point clouds with the spatial graph convolutions. Subsequent works explored more effective filter or kernel parameterizations [19, 35, 62, 65, 69]. The recent discrete kernels [31, 32, 33, 59] are appealing alternatives to those dynamic methods as they avoid the dependence of filter generations within the network. KPConv [59] reported more competitive results, while the spherical kernels [32, 33] are more memory and runtime efficient.

Convolution on meshes: In nascency of this direction, researchers generally performed convolutions over local patches defined in a hand-crafted coordinate system [5, 39, 41]. The coordinate system could either be established by geodesic level sets [39] or surface normals and principle curvatures [5, 41]. In FeaStNet [61], Verma *et al.* capitalized on a learnable mapping between filter weights and graph neighborhoods to replace those hand-crafted local patches. TextureNet [23] takes surface meshes with high-resolution facet textures as input, and explores a 4-RoSy field to parameterize the mesh into local planar patches such that standard CNNs [28] are applicable. Ranjan *et al.* [14] proposed to learn human facial expressions with hierarchical mesh-based networks, using the spectral graph convolutions. Schult *et al.* [52] proposed to extract features from both meshes and point clouds simultaneously. Similar to [35, 54, 64, 65], they still conduct convolution using dynamically generated filters. Whereas most methods learn vertex-wise features, MeshCNN [20] defines convolution to learn edge-wise features on a mesh.

Generally, previous methods treat mesh as an edge-based graph and employ geodesic convolutions over it. In this paper, we explore convolutions on the mesh following its own geometric structure, i.e. vertices and facets. To promote this more natural perspective, we also provide computation and memory optimized CUDA implementations for forward and backward propagations of all the convolutions we propose.

Mesh decimation: Hierarchical networks allow convolutions to be applied on an increasing receptive fields of the input data. To create such hierarchical architectures on point clouds, researchers usually exploit random point sampling or farthest point sampling (FPS) [33, 44, 65]. However, because of their inability to track vertex connections, they are not applicable to mesh processing. Fortunately, mesh simplification is a well-studied topic in the graphics community. There are methods available that can be used to decimate a mesh [16, 17, 51]. For example, Ranjan *et al.* [45] used the quadric error metrics [17] to simplify their synthetic facial meshes. Schult *et al.* [52] explored vertex clustering [51] and quadric error metrics [16, 17] to simplify their indoor room meshes. In particular, they made use of the simplification functions provided by Open3D [71] - a popular library for 3D geometry processing in Python. However, the implementations in Open3D are CPU-based,

which are not amenable to deep learning.

In this work, we also introduce a fast mesh decimation method based on the algorithm of Garland *et al.* [16, 17]. Their method simplifies a mesh through iterative contractions of vertex pairs, and demands that the vertex pair contribution to quadric error be determined after each iteration. This progressive strategy makes the algorithm impossible for parallel deployment on GPUs. In comparison, we sort all the vertex pairs by their quadric errors only once, and group them into isolated vertex clusters. All other computations in our method are mutually independent and can be accelerated via parallel GPU computing. We represent the vertex cluster information as a 1-D tensor to facilitate the pooling and unpooling operations.

3. GPU-Accelerated Mesh Decimation

To explore flexible deep neural networks for meshes, there has been an increasing demand for a fast mesh decimation method in the 3D community. The quadric error simplification [17] performs iterative contraction of vertex pairs until the desired number of facets is obtained. This method is known to be effective for simplifying meshes while retaining the quality. However, it is not suited to parallel computing due to the implicit dependencies between its iterative contractions. Considering its high relevance, we provide the quadric error method as Algorithm 1. The iterative dependency of the method is clear from *lines 7–13* of the algorithm.

We extend [17] and propose a fast mesh decimation method that exploits the parallel computing power of a GPU. The main difference is that we do not perform iterative contractions any more. Instead, we group the vertices into multiple isolated clusters based on their connections. Figure 2 provides a toy example to illustrate the clustering process. During clustering, we control the expected vertex number rather than the number of facets or edges. Since the contractions of isolated clusters are independent of each other, they can be executed on a GPU in parallel. More specifically, we initialize the candidates of vertex pairs to be contracted using existing mesh edges only. The vertex clusters are established with disjoint vertex pairs in the candidates, before which we sort the candidates in an ascending order by their quadric cost[2]. See Algorithm 2 for our method. We note that our core algorithm reduces the vertex number roughly by a half, per-iteration. Yet, we can still handle arbitrary number of output vertices. It is achieved by running the core algorithm multiple times. For clarity, we present Algorithm 2 with a single mesh taken as the input. In our library, the decimation function can take mini-batches of concatenated meshes as input. This improves its compatibility with deep learning, e.g. batch normaliza-

---

[2]Shuffling strategy is inserted into the sorting of quadrics for variations of the decimated mesh at different epochs.

Algorithm 1 The original quadric error simplification

Input: A triangular mesh $\mathcal{T}^i = (\mathcal{V}^i, \mathcal{F}^i)$, the number of output facets M, threshold τ.
Output: A decimated mesh $\mathcal{T}^o = (\mathcal{V}^o, \mathcal{F}^o)$.
1: select any pair $(\mathbf{v}_i, \mathbf{v}_j)$ that is an edge or $\|\mathbf{v}_i - \mathbf{v}_j\| < \tau$ as candidates
2: compute the quadric of each vertex pair
3: **for** each candidate pair $(\mathbf{v}_i, \mathbf{v}_j)$ **do**
4:    (b) determine the target position $\bar{\mathbf{v}}$ of contraction
5:    (c) apply consistency checks and penalties
6: **end for**
7: **repeat**
8:   **if** the pair $(\mathbf{v}_i, \mathbf{v}_j)$ has the least quadric cost Q **then**
9:     (b) perform contraction $(\mathbf{v}_i, \mathbf{v}_j) \to \bar{\mathbf{v}}$
10:     (c) update $Q_i = Q_i + Q_j$
11:     (d) for each affected pair $(\mathbf{v}_i, \mathbf{v}_k)$, recompute its target position and cost as in steps 3-6
12:   **end if**
13: **until** $|\mathcal{F}^o| < M$
14: **return**

Algorithm 2 Fast GPU-accelerated mesh simplification

Input: A triangular mesh $\mathcal{T}^i = (\mathcal{V}^i, \mathcal{F}^i)$, the number of output vertices N.
Output: A decimated mesh $\mathcal{T}^o = (\mathcal{V}^o, \mathcal{F}^o)$; vertex clustering information *replace*; vertex degeneration information *mapping*.
1: select any pair $(\mathbf{v}_i, \mathbf{v}_j)$ that is an edge.
2: compute the quadric of each vertex pair
3: sort all pairs ascendingly based on their quadrics
4: set the number of vertices to remove as $N_r = |\mathcal{V}^i| - N$; the number of vertices removed as $n_r = 0$
5: **for** each pair $(\mathbf{v}_i, \mathbf{v}_j)$ **do**
6:   **if** $\mathbf{v}_i, \mathbf{v}_j$ are not in any cluster **and** $n_r < N_r$ **then**
7:     (a) form $(\mathbf{v}_i, \mathbf{v}_j)$ as a new cluster
8:     (b) set $n_r = n_r + 1$
9:   **end if**
10: **end for**
11: add the remaining vertices that are not in any cluster to an existing cluster containing vertices connected to it.
12: **for** each vertex cluster **do**
13:    (a) compute quadric Q
14:    (b) determine the target position $\bar{\mathbf{v}}$ of contraction
15:    (c) perform contraction $(\mathbf{v}_i, \mathbf{v}_j) \to \bar{\mathbf{v}}$
16: **end for**
17: **return**

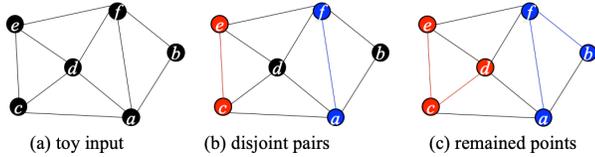(a) toy input   (b) disjoint pairs   (c) remained points

Figure 2: Illustration of edge-based vertex clustering. (a) shows an input mesh with 10 vertex pairs. (b) initializes the disjoint vertex pairs (c, e), (a, f) as clusters $\{c, e\}$, $\{a, f\}$ - shown by red and blue. (c) groups the remaining points d, b to the vertex clusters they each connect to, resulting in the final vertex clusters $\{c, d, e\}$ and $\{a, b, f\}$.

tion [24] is easily applicable. The vertex clustering operations (*lines 3–11* in Algorithm 2) is currently executed on CPU, that helps in deploying heavy computations to GPU. We note that consistency checks and penalties are excluded in our method in favor of better runtime efficiency. For continuity, we delegate the details on complementary output parameters *replace* and *mapping* to the source code itself.

Pooling and unpooling: We record the vertex clustering information as well as the degenerated vertex information into different 1-D tensors. The sizes of the two tensors are both identical to the input vertex number. They largely facilitate the pooling and unpooling operations. Given the vertex clusters as neighborhood, different pooling operations can be defined, such as sum($\cdot$), average($\cdot$), max($\cdot$) and weighted($\cdot$). We provide average($\cdot$), max($\cdot$), weighted($\cdot$) poolings in the library. For unpooling, we simply interpolate features of all vertices in a cluster based on features of the representative vertex in the cluster.

4. Mesh Convolution

Let $\mathcal{T} = (\mathcal{V}, \mathcal{F})$ be a triangular mesh. The input features of its vertices are (x, y, z, r, g, b), while the normals and areas of each facet are $\mathbf{n} = (n_x, n_y, n_z)$ and A. If the mesh is rendered, we denote the textures of a facet as $\Gamma \times 3$, in which Γ represents the texture resolution of the facet, 3 relates to the (r, g, b) colors. Since the facets usually have varying areas, we allow varying Γ for different facets as well.

4.1. Vertex2Facet convolution

We compute the features of each facet based on the features of its vertices. In particular, we define a kernel composed of 3 filters associated to the three vertices of the triangular facet. The Barycentric interpolation is used to incorporate fuzzy scheme into the convolution. We determine the total number of interpolated points K as

$$K = \frac{k \times (k+1)}{2}, \quad k = \left\lfloor \frac{A - A_{\min}}{A_{\max} - A_{\min}} \right\rfloor \times \alpha + \beta, \quad (1)$$

in which $A_{\min}, A_{\max}$ are the minimum and maximum facet areas of the mesh, while α, β are the hyper-parameters. We use $\alpha = \beta = 1$ in our experiments.

Let (I_1, I_2, I_3) be the vertex features, and (w_1, w_2, w_3) be the filter weights. The Barycentric coordinates satisfies $\xi_{k1} + \xi_{k2} + \xi_{k3} = 1$, $\xi_{k1}, \xi_{k2}, \xi_{k3} \geq 0$. We use it to interpolate interior points uniformly on the facet, whose features

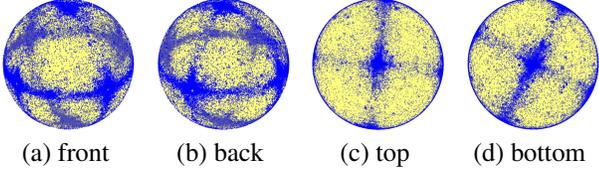(a) front   (b) back   (c) top   (d) bottom

Figure 3: The distributions of surface normals of an arbitrary room. We show it from different views, including front, back, top and bottom. The six apparent clusters have centers approximating to $[\pm 1, 0, 0]$, $[0, \pm 1, 0]$, $[0, 0, \pm 1]$.

and filter weights are computed respectively as

$$I_k = \xi_{k1} * I_1 + \xi_{k2} * I_2 + \xi_{k3} * I_3, \qquad (2)$$

$$W_k = \xi_{k1} * w_1 + \xi_{k2} * w_2 + \xi_{k3} * w_3, \ k \in K. \qquad (3)$$

Here, K is the total number of interpolated points defined based on the facet area. With $\{I_k\}$ and $\{W_k\}$, we finally compute the feature of each facet as

$$J = \frac{1}{K} \sum_k W_k * I_k. \qquad (4)$$

Facet2Facet convolution: The facet2facet convolution is only applicable when the input mesh is rendered with textures. Each facet contains Γ interpolated points with texture features of size $\Gamma \times 3$. The definition of the facet2facet convolution is quite similar to that of the vertex2facet convolution. The main difference is that in facet2facet convolution, there is no need to interpolate features for the interior points since their features are already available. We only need to interpolate the filter weights $\{W_k\}$, and then compute the facet features following Eq. (4).

4.2. Facet2Vertex Convolution

We compute vertex features from the features of their adjacent facets. Considering that the normals of each facet are strictly located on the surface of a unit sphere, we define filter weights of our kernel by associating them to different positions on the sphere. Besides, we observe that normals of the real-world meshes generally distribute in distinctive patterns on a unit sphere, especially for the indoor meshes. This is related to the construction preferences of human-beings. We show an example in Fig. 3. The data is taken from S3DIS [2]. There are six main clustering patterns. They correspond to different normal directions, which are roughly $[\pm 1, 0, 0], [0, \pm 1, 0], [0, 0, \pm 1]$. Consequently, we exploit the mixture of different Gaussian components to divide the sphere surface and implicitly cluster normals.

Let the total number of Gaussian components be T, their expectations and covariance matrices be $\{\boldsymbol{\mu}_t\}$ and $\{\Sigma_t\}$. Using its normal $\mathbf{n}_i$, we compute the fuzzy coefficients

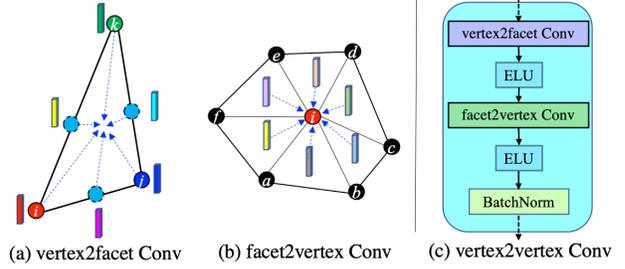(a) vertex2facet Conv   (b) facet2vertex Conv   (c) vertex2vertex Conv

Figure 4: Convolution operations introduced in Picasso. (a) The vertex2facet convolution propagates features from the original and interpolated points on a facet to the facet itself. (b) The facet2vertex convolution propagates features from the adjacent facets of a vertex to the vertex itself. We omit illustration of the facet2facet convolution for its visual similarity to the vertex2facet convolution. (c) The vertex2vertex convolution is composed of a vertex2facet convolution followed by a facet2vertex convolution. We apply batch normalization to the vertex features.

$\{\pi_{it}\}$ of the facet $\mathbf{f}_i$ as

$$z_{it} = (\mathbf{n}_i - \boldsymbol{\mu}_t)^T \Sigma_t^{-1} (\mathbf{n}_i - \boldsymbol{\mu}_t), \qquad (5)$$

$$\pi_{it} = \frac{\exp(-z_{it})}{\sum_m \exp(-z_{im})}. \qquad (6)$$

For simplicity, we use homogeneous diagonal matrix for Σ_k in our experiments. Let the filter weights in the kernel be $\{w_t\}$, the adjacent facets of vertex $\mathbf{v}$ be $\mathcal{N}(\mathbf{v})$, and the facet features be $\{J_i\}$. The vertex feature gets computed as

$$I_v = \frac{1}{\mathcal{N}(\mathbf{v})} \sum_{\mathbf{f}_i \in \mathcal{N}(\mathbf{v})} \left(\sum_{t=1}^{T} \pi_{it} w_t \right) J_i. \qquad (7)$$

In our library, we allow the expectations and covariance matrices of the Gaussians to be constant, or learnable together with the filter weights during training. We initialize the Gaussian means as regularly distributed points on the sphere in our experiments and keep those constant, while allowing the covariance matrices as learnable parameters. We note that the facet2vertex convolution is scale and translation invariant but not rotation invariant because its fuzzy coefficients are computed based on facet normals.

For all the proposed convolutions, we split the operation into channel-wise and depth-wise operations. Doing so is known to improve the computational efficiency of the operation [10, 33]. Our mesh decimation technique forces the vertex number, rather than the facet number, of each decimated mesh in a batch to be the same. Therefore, we apply batch normalization to vertex features for stable computation of the batch statistics. Combining the vertex2facet and facet2vertex convolutions, we can perform vertex2vertex convolution. Figure 4 illustrate the concept of vertex2facet and facet2vertex convolutions, along with a flow chart to elucidate the operations in vertex2vertex convolution.

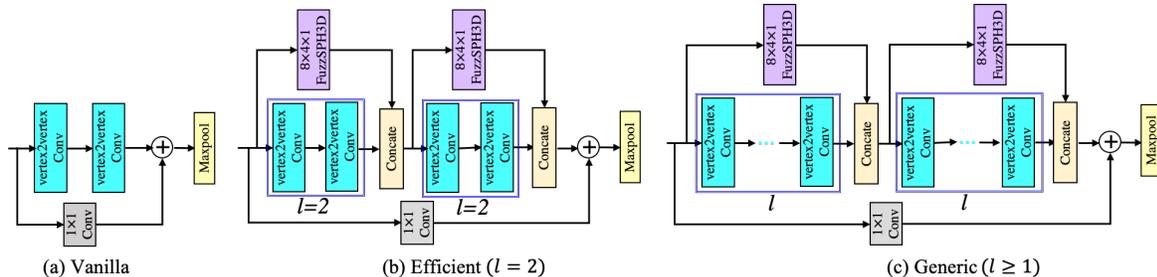(a) Vanilla   (b) Efficient ($l = 2$)   (c) Generic ($l \geq 1$)

Figure 5: Convolutional blocks of PicassoNet. The network consists of 5 mesh resolutions including the input, which relates to a network of 5 hierarchical layers. Convolutional blocks are used to learn features within a single hierarchical layer. We test the performance of different block configurations for feature learning. (a) The *vanilla* block consists of two vertex2vertex convolutions with a shortcut connection. (b) The *efficient* block type contains two vertex2vertex convolutions as well as one point-based convolution before each feature concatenation. It can process meshes with *1 million* facets per-second at the inference time, while maintaining highly competitive results. (c) The *generic* convolution block uses $l \geq 1$ vertex2vertex convolutions together with a single point-based convolution before the feature combination. We report at-par results to state-of-the-art on S3DIS [2] by using generic block with $l = 4$.

5. Network Architecture

Receptive field: All the convolutions we propose follow the natural organisation of vertex lists in the mesh facets. Therefore, the operations accumulate context from the first-order neighborhood. This is similar to the receptive field reached by 3×3 convolution in standard CNNs for image processing. To benefit feature learning from larger context, we apply multiple cascaded vertex2vertex convolutions in our network, which leads to a deeper neural network.

Disconnected components: Real-world meshes are not guaranteed to form a connected graph. It is probable that a mesh is composed of multiple disconnected components. This breaks the context aggregation between different components for mesh convolutions. On the other hand, *forcing* the mesh to be connected can damage its geometric structure. In this work, we counter the potential issues of mesh connections using point cloud convolution, e.g. the fuzzy SPH3D [32]. By using range search [42], point cloud convolutions permit arbitrary respective fields. Finally, this strategy enables our feature learning to go beyond single connected components and learn features across multiple components.

The PicassoNet: To explore the proposed modules, we design a U-net [50] like architecture for the popular semantic segmentation task. For consistency, we name our network as PicassoNet. The proposed network is composed of 5 mesh resolutions, indicating 5 hierarchical layers. We use convolution blocks to learn features within a single network resolution. Figure 5 shows different block configurations considered in this work. The *vanilla* block uses only the mesh convolutions, while the *efficient* block uses $l = 2$ mesh convolutions accompanied by a point-based convolution in parallel, before feature concatenation. PicassoNet based on the efficient block is able to process 1 million facets per-second at the inference time, still reporting com-

petitive results. The *generic* block uses arbitrary number of mesh-based convolutions with one point-based convolution to extract features before the concatenation. Unlike DCM-Net [52], we do not tune the ratio of feature channels between mesh-based and point-based convolutions. We use identical feature channels for both the convolutions, which correspond to a fixed ratio of 0.5. We use max mesh pooling to obtain down-sampled features for the low-resolution networks. For efficiency, we apply the convolution blocks only in the encoder. Whereas in the decoder, we use 1×1 convolutions and mesh unpooling to upsample the features, somewhat similar to the methods in [32, 59].

6. Experiments

PicassoNet takes 6-dimensional input features (x, y, z, r, g, b), similar to many point cloud networks [32, 33, 59]. Although normals are readily available for both facets and vertices on meshes, we do not observe noticeable performance gain by using vertex normals as input features in our experiments. To investigate the modules presented in the Picasso library, we focus on the semantic segmentation of the real-world dataset S3DIS [2] using the PicassoNet. We choose S3DIS dataset due to the public availability of its training and testing sets. The rgb color values are re-scaled in the range $[-1, 1]$ before feeding into the network.

Our network is trained on a single GeForce RTX 2080 Ti GPU with Adam Optimizer [25]. For training, we adopt initial learning rate 0.001 with exponential decay. In specific, we decay the learning rate with a rate of 0.7 every 20K batch updates. Throughout the experiments, we use 18 Gaussian components whose centers are located regularly on the unit sphere for all facet2vertex convolutions. See supplementary for their specific values. The standard deviations of these Gaussians are universally initialized to 0.25, which is de-
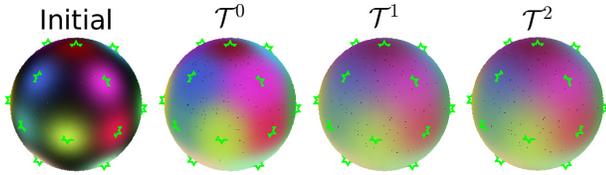
Figure 6: Visualized comparison of the trained Gaussians at different mesh resolutions with their universal initialisation. The brightness of the colors also accounts for the difference in the values. The green markers indicates centers of each Gaussian. We show only the results at mesh resolutions $\mathcal{T}^0$, $\mathcal{T}^1$, $\mathcal{T}^2$. The following resolutions look visually similar to $\mathcal{T}^2$.

termined by the nearest center distances between different Gaussians. We switch off training of the Gaussian expectations on purpose because we observe that their values do not change too much in our experiments. Figure 6 provides an example to show the updates of the standard deviations of the Gaussians during the training.

We use the fuzzy spherical kernel proposed in [32] to achieve point-based convolution in our *efficient* and *generic* blocks. The kernel size and neighborhood search configurations are identical to [32]. Our network uses batch size 16 in the training and takes point clouds of size $8,192$ as inputs. We exploit the widely used data augmentations in our experiments, including random flipping, shifting, random scaling, noisy translation, random azimuth rotation and arbitrary rotation perturbations. We also apply random color dropping to augment the vertex textures. The shuffling strategy we inserted in the mesh simplification also played a role of data augmentation. We apply these augmentations on-the-fly during the network training.

Network Configuration: We use identical feature configurations for different convolution blocks shown in Fig. 5. In specific, the desired vertex sizes of meshes $\mathcal{T}^0$, $\mathcal{T}^1$, $\mathcal{T}^2$, $\mathcal{T}^3$, $\mathcal{T}^4$ are 8192, 2048, 1024, 512, 256 respectively. To construct the neighborhoods for point-based convolution, we use an increasing range search radius 0.1, 0.2, 0.4, 0.8, 1.6. The output feature sizes of the 5 hierarchical layers are 128, 128, 256, 256, 256, respectively. When mesh-based and point-based convolutions are both included, they each compute features of half the size of the defined output feature channels. For example, when the output feature channels are set to 128, they each compute features of 64 channels. This results in their concatenated features to have 128 channels. We set the multiplier λ of all convolutions to 1 except for the convolution performed on the inputs, which has $\lambda = 2$. In the decoder, we explore mesh unpooling for feature interpolation. The output feature channels of the 1×1 convolutions are the same as their corresponding encoder features. PicassoNet also classifies the features obtained at resolution $\mathcal{T}^0$ in the decoder directly for efficiency.

6.1. Semantic Segmentation

The Stanford large-scale 3D Indoor Spaces (S3DIS) dataset [2] is a real-world dataset composed of dense 3D point clouds but sparse 3D meshes of 6 large-scale indoor areas. The data is collected using Matterport scanner from three different buildings on the Standard campus. Semantic segmentation on this dataset is to classify the 13 defined classes, *ceiling, floor, wall, beam, column, window, door, table, chair, sofa, bookcase, board*, and *clutter*. We follow the standard training/test split by using Area 5 as the test set while the other areas as training set [30, 35, 43, 58, 63]. The evaluation metrics for this dataset comprise the Overall Accuracy (OA), average Accuracy of all 13 classes (mAcc), class-wise Intersection Over Union (IoU), together with their average (i.e. mIoU). mIoU is commonly considered a more reliable metric than the others.

DCM-Net [52] used over-tessellation and interpolation to produce high resolution meshes with labels from the original meshes in S3DIS. In contrast, we triangulate the labelled point cloud into triangular meshes using the algorithm from [1]. Our network takes meshes of size $2.0m \times 2.0m$ cropped from the room mesh as input data. The experiment results for different block configurations in the PicassoNet are provided in Table 1. It can be noticed that PicassoNet ($l = 4$) provides competitive results to KPConv [59], and slightly outperforms SegGCN [32] and DCM-Net [52]. Table 2 reports the training time of PicassoNet ($l = 2$) for batch size 16, and its testing time for different batches. It can be seen that PicassoNet ($l = 2$) can process over a million facets per-second on a commonly available single RTX 2080 Ti GPU.

6.2. Mesh Simplification Efficiency

We summarize the statistics of vertex and facet sizes of room meshes we generate in S3DIS. The minimum, maximum, average of (vertex, facet) numbers are $(10K, 20K)$, $(1157K, 2288K)$, and $(125K, 247K)$. The standard deviations are as large as $(122K, 241K)$. Our fast mesh decimation method contributes largely to the efficiency of our network, e.g. million facets processed per second. We apply our simplification algorithm to all room samples in S3DIS to decimate them each into a mesh of 65536 vertices. We compare the speed of our algorithm to the quadric error metrics (QEM) method [17] that our method builds upon. The results are shown in Fig. 7 (left plot), in which the x-axis indicates input vertex sizes of the room meshes while the y-axis reports the runtime of the algorithms.

Besides, we also test the runtime of our algorithm by decimating those room meshes into different resolutions, which corresponds to vertex sizes of 65536, 32768, 16384, 8192, 4096, 2048, 1024, 512 and 256. These results are reported in Fig. 7 (right plot). We can tell from the figure that the runtime of our algorithm is essentially linear in the size

Table 1: Performance of different configurations for PicassoNet on the fifth fold (Area 5) of S3DIS dataset. The results of PicassoNet using $l = 4$ outperforms SegGCN and DCM-Net slightly, and are competitive for KPConv. See supplementary for details of input representations exploited by each method.

	Methods	OA	mAcc	mIoU	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter
	PointNet [43]	-	49.0	41.1	88.8	97.3	69.8	0.1	3.9	46.3	10.8	58.9	52.6	5.9	40.3	26.4	33.2
	SEGCloud [58]	-	57.4	48.9	90.1	96.1	69.9	0.0	18.4	38.4	23.1	70.4	75.9	40.9	58.4	13.0	41.6
	Tangent-Conv [57]	82.5	62.2	52.8	-	-	-	-	-	-	-	-	-	-	-	-	-
	SPG [30]	86.4	66.5	58.0	89.4	96.9	78.1	0.0	42.8	48.9	61.6	75.4	84.7	52.6	69.8	2.1	52.2
Area 5	PointCNN [35]	85.9	63.9	57.3	92.3	98.2	79.4	0.0	17.6	22.8	62.1	74.4	80.6	31.7	66.7	62.1	56.7
	SSP+SPG [29]	87.9	68.2	61.7	-	-	-	-	-	-	-	-	-	-	-	-	-
	GACNet [62]	87.8	-	62.9	92.3	98.3	81.9	0.0	20.4	59.1	40.9	78.5	85.8	61.7	70.8	74.7	52.8
	SPH3D-GCN [33]	87.7	65.9	59.5	93.3	97.1	81.1	0.0	33.2	45.8	43.8	79.7	86.9	33.2	71.5	54.1	53.7
	KPConv [59]	-	70.9	65.4	92.6	97.3	81.4	0.0	16.5	54.5	69.5	90.1	80.2	74.6	66.4	63.7	58.1
	SegGCN [32]	88.2	70.4	63.6	93.7	98.6	80.6	0.0	28.5	42.6	74.5	80.9	88.7	69.0	71.3	44.4	54.3
	DCM-Net[52]	-	71.2	64.0	92.1	96.8	78.6	0.0	21.6	61.7	54.6	78.9	88.7	68.1	72.3	66.5	52.4
	PicassoNet (Prop. vanilla)	86.6	65.6	58.0	93.2	98.0	78.3	0.0	16.8	28.5	61.7	75.5	86.4	49.2	69.0	45.4	52.1
	PicassoNet (Prop. $l = 1$)	87.8	67.8	61.0	93.8	97.6	80.6	0.0	32.6	48.2	69.3	77.8	87.0	56.8	70.1	25.5	53.3
	PicassoNet (Prop. $l = 2$)	88.2	69.4	62.5	93.2	98.4	81.1	0.0	32.1	45.9	75.6	78.5	85.9	51.5	69.0	45.2	55.3
	PicassoNet (Prop. $l = 4$)	89.4	70.9	64.6	93.3	97.7	83.5	0.0	31.9	53.4	69.2	81.7	88.0	50.5	74.3	58.2	57.9

Table 2: Training and testing time of the efficient PicassoNet ($l = 2$) for different mini-batch settings. The 'mesh size' indicates total number of vertices and facets as (vertices, facets) in the concatenated mesh of a batch. We add the testing runtime of SegGCN under identical settings for pure point cloud input at the bottom row as a reference.

Phase	training	testing				
batch size	16	16	32	64	68	128
mesh size	(0.13M,0.24M)	(0.13M,0.24M)	(0.26M,0.48M)	(0.52M,0.96M)	(0.56M,**1.02M**)	(1.04M,1.92M)
runtime	920 ms	318 ms	500 ms	890 ms	**915** ms	1750 ms
testing runtime of SegGCN [32]		153 ms	250 ms	470 ms	485 ms	930 ms

of vertices of the input mesh. These conclusions about the efficiency of our simplification algorithm are generic, as we observe similar results consistently on other mesh datasets also. See the supplementary for further results.

7. Conclusion

We provide a fast CUDA-accelerated mesh decimation technique to facilitate the 3D community to explore hierarchical neural architectures on meshes. We propose 3 novel convolutions, namely; vertex2facet, facet2facet, and facet2vertex. We provide CUDA implementations for forward and backward passes for these convolutions. We also introduce the vertex2vertex convolution on top of vertex2facet and facet2vertex convolutions. We use Gaussian mixtures and Barycentric interpolation to incorporate fuzziness into the proposed convolutions. Our mesh simplification gather vertex clustering information into a 1-D tensor, which is convenient for the CUDA-based pooling and unpooling operations. Most importantly, we introduce Picasso, a self-contained library for deep learning over 3D meshes and point clouds. We also present PicassoNet for semantic segmentation, and test its performance on S3DIS which provides at par results to state-of-the-art. The efficient version of PicassoNet can process 1 *million* facets during inference while retaining competitive accuracy. We also
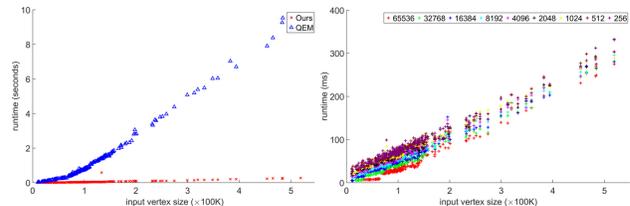

Figure 7: The left figure compares the runtime of our mesh decimation method with QEM. The data is plotted by decimating every sample in S3DIS to a mesh of 65536 vertices by both methods. We notice that QEM takes much more time than ours, usually in 'seconds', while our algorithm runs in 'milliseconds'. The right figure shows the time taken by our simplification algorithm to decimate all mesh samples in S3DIS to different vertex sizes, including 65536, 32768, 16384, 8192, 4096, 2048, 1024, 512, and 256. It can be observed that the runtime of our algorithm remains generally linear to the vertex size of the input mesh.

observe that more mesh convolutions in the deeper version increases the receptive field, and hence learn better features. In the future, we will maintain and upgrade the introduced library for efficiency and further functionalities.

13861

References

[1] Surface reconstruction from scattered points. `https://au.mathworks.com/matlabcentral/fileexchange/63731-surface-reconstruction-from-scattered-points-cloud-open-surfaces`. accessed 17-Nov-2020.

[2] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3D semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1534–1543, 2016.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Federica Bogo, Javier Romero, Matthew Loper, and Michael 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.

[5] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in neural information processing systems*, pages 3189–3197, 2016.

[6] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[7] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.

[8] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017.

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

[10] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[11] Luca Cosmo, Emanuele Rodolà, Michael M Bronstein, Andrea Torsello, Daniel Cremers, and Y Sahillioglu. Shrec'16: Partial matching of deformable shapes. *Proc. 3DOR*, 2(9):12, 2016.

[12] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5828–5839, 2017.

[13] Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. In *Advances in Neural Information Processing Systems*, 2016.

[14] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[15] M. Engelcke, D. Rao, D. Zeng Wang, C. Hay Tong, and I. Posner. Vote3Deep: Fast object detection in 3D point clouds using efficient convolutional neural networks. In *IEEE International Conference on Robotics and Automation*, June 2017.

[16] Michael Garland. Quadric-based polygonal surface simplification [thesis]. *Pittsburgh: Carnegie Mellon University*, 1999.

[17] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.

[18] Benjamin Graham, Martin Engelcke, and Laurens Van Der Maaten. 3D semantic segmentation with submanifold sparse convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9224–9232, 2018.

[19] Fabian Groh, Patrick Wieschollek, and Hendrik PA Lensch. Flex-convolution. In *Asian Conference on Computer Vision*, pages 105–122. Springer, 2018.

[20] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[22] Binh-Son Hua, Quang-Hieu Pham, Duc Thanh Nguyen, Minh-Khoi Tran, Lap-Fai Yu, and Sai-Kit Yeung. Scenenn: A scene meshes dataset with annotations. In *International Conference on 3D Vision (3DV)*, 2016.

[23] Jingwei Huang, Haotian Zhang, Li Yi, Thomas Funkhouser, Matthias Nießner, and Leonidas J Guibas. Texturenet: Consistent local parametrizations for learning from high-resolution signals on meshes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4440–4449, 2019.

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[26] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[27] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud mod-

els. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 863–872. IEEE, 2017.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[29] Loic Landrieu and Mohamed Boussaha. Point cloud over-segmentation with graph-structured deep metric learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[30] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[31] Huan Lei, Naveed Akhtar, and Ajmal Mian. Octree guided cnn with spherical kernels for 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9631–9640, 2019.

[32] Huan Lei, Naveed Akhtar, and Ajmal Mian. Seggcn: Efficient 3d point cloud segmentation with fuzzy spherical kernel. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11611–11620, 2020.

[33] Huan Lei, Naveed Akhtar, and Ajmal Mian. Spherical kernel for efficient graph convolution on 3d point clouds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[34] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9397–9406, 2018.

[35] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *Advances in Neural Information Processing Systems*, pages 820–830, 2018.

[36] Z Lian, A Godil, B Bustos, M Daoudi, J Hermans, S Kawamura, Y Kurita, G Lavoua, and P Dp Suetens. Shape retrieval on non-rigid 3d watertight meshes. In *Eurographics workshop on 3d object retrieval (3DOR)*. Citeseer, 2011.

[37] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37, 2016.

[38] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[39] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

[40] Daniel Maturana and Sebastian Scherer. VoxNet: A 3D convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 922–928. IEEE, 2015.

[41] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[42] Franco P Preparata and Michael I Shamos. *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[43] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

[44] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems*, 2017.

[45] Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J Black. Generating 3d faces using convolutional mesh autoencoders. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 704–720, 2018.

[46] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[47] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[48] Dario Rethage, Johanna Wald, Jurgen Sturm, Nassir Navab, and Federico Tombari. Fully-convolutional point networks for large-scale point clouds. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 596–611, 2018.

[49] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. OctNet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3577–3586, 2017.

[50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

[51] Jarek Rossignac and Paul Borrel. Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in computer graphics*, pages 455–465. Springer, 1993.

[52] Jonas Schult, Francis Engelmann, Theodora Kontogianni, and Bastian Leibe. Dualconvmesh-net: Joint geodesic and euclidean convolutions on 3d meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8622, 2020.

[53] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining point cloud local structures by kernel correlation and graph pooling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 4, 2018.

[54] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[55] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015.

[56] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan 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.

[57] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3887–3896, 2018.

[58] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *International Conference on 3D Vision*, pages 537–547. IEEE, 2017.

[59] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[60] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[61] Nitika Verma, Edmond Boyer, and Jakob Verbeek. Feastnet: Feature-steered graph convolutions for 3d shape analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2598–2606, 2018.

[62] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. Graph attention convolution for point cloud semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition*, June 2019.

[63] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. Graph attention convolution for point cloud semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10296–10305, 2019.

[64] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[65] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019.

[66] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[67] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong 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.

[68] Jin Xie, Yi Fang, Fan Zhu, and Edward Wong. Deepshape: Deep learned shape descriptor for 3d shape matching and retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1275–1283, 2015.

[69] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European Conference on Computer Vision*, pages 87–102, 2018.

[70] Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2282–2290, 2017.

[71] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018.