Neural Mesh Simplification

Rolandos Alexandros Potamias[1] Stylianos Ploumpis[1,2] Stefanos Zafeiriou[1,2]
[1]Imperial College London [2]Huawei Technologies Co. Ltd
{r.potamias, s.ploumpis, s.zafeiriou}@imperial.ac.uk

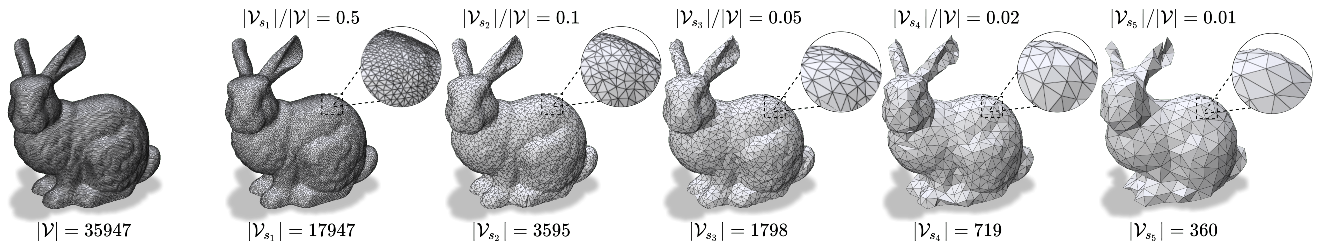
$|\mathcal{V}_{s_1}|/|\mathcal{V}| = 0.5$
$|\mathcal{V}_{s_2}|/|\mathcal{V}| = 0.1$
$|\mathcal{V}_{s_3}|/|\mathcal{V}| = 0.05$
$|\mathcal{V}_{s_4}|/|\mathcal{V}| = 0.02$
$|\mathcal{V}_{s_5}|/|\mathcal{V}| = 0.01$
$|\mathcal{V}| = 35947$
$|\mathcal{V}_{s_1}| = 17947$
$|\mathcal{V}_{s_2}| = 3595$
$|\mathcal{V}_{s_3}| = 1798$
$|\mathcal{V}_{s_4}| = 719$
$|\mathcal{V}_{s_5}| = 360$

Figure 1. We present a fast and learnable method for mesh simplification that generates simplified meshes in real-time.

Abstract

Despite the advent in rendering, editing and preprocessing methods of 3D meshes, their real-time execution remains still infeasible for large-scale meshes. To ease and accelerate such processes, mesh simplification methods have been introduced with the aim to reduce the mesh resolution while preserving its appearance. In this work we attempt to tackle the novel task of learnable and differentiable mesh simplification. Compared to traditional simplification approaches that collapse edges in a greedy iterative manner, we propose a fast and scalable method that simplifies a given mesh in one-pass. The proposed method unfolds in three steps. Initially, a subset of the input vertices is sampled using a sophisticated extension of random sampling. Then, we train a sparse attention network to propose candidate triangles based on the edge connectivity of the sampled vertices. Finally, a classification network estimates the probability that a candidate triangle will be included in the final mesh. The fast, lightweight and differentiable properties of the proposed method makes it possible to be plugged in every learnable pipeline without introducing a significant overhead. We evaluate both the sampled vertices and the generated triangles under several appearance error measures and compare its performance against several state-of-the-art baselines. Furthermore, we showcase that the running performance can be up to 10× faster than traditional methods.

1. Introduction

Triangle meshes remain the most popular 3D structure to represent a surface. The advent of 3D scanning devices have made feasible to collect highly detailed 3D meshes that typically hold thousand of faces. However, extreme details lead to enormous memory requirements that limit their usage. A wide range of applications including rendering and editing along with their mobile implementations, require lightweight meshes to enable real-time processing. Additionally, many monocular 3D reconstruction methods that utilize analysis-by-synthesis are required to be computational efficient and differentiable in terms of their topologies. These types of iterative optimization methods can drastically benefit from an differentiable on-the-fly simplification technique that reduce their computational footprint.

Mesh simplification is a long studied problem, with an immense amount of methods developed to sustainably reduce the size of the original mesh without extremely distorting its appearance. Traditional simplification techniques decimate the input mesh in a greedy-fashion by prioritizing vertices and edges according to a cost function [10, 34]. However, in large-scale objects scenario, simplifying over 90% of the original mesh size requires iterating through thousands of vertices resulting in an inevitable computational burden. In addition to their computational footprint, traditional simplification techniques are non-differentiable and thus can not be used directly in end-to-end training processes that optimize the mesh surface. To alleviate the aforementioned limitations we propose a learnable strategy for mesh simplification that reduces both time and computational requirements and provides a plug-and-play method, ready to be adapted in any differentiable framework.

A major barrier that limits learnable simplification methods is the discrete nature of the mesh connectivity, i.e. edges and triangles. Although mesh simplification can be achieved in a two-step process using a learnable sampling method followed by an off-the-shelf triangulation algorithm (e.g. Delaunay or Ball Pivoting) [28], such setting, apart from being time consuming, limits the direct optimization of the mesh surface. Recently, several approaches have been proposed to solve the non-trival task of differential triangulation using soft relaxations of the discrete setting. However, most of them are considered impractical since they are applied to volumetric representations [5], demand iterative optimizations for the triangulation for each mesh [37] or require 2D parametrizations [30, 31]. Our aim is to utilize a simple but intuitive differentiable process to directly triangulate the 3D points in one-pass. To do so, we model the triangulation process by generating a distribution over possible edges and triangles and select the ones that preserve the appearance of the original mesh.

In this study we propose the first learnable mesh simplification method that generates both points and triangulation of the simplified mesh. In contrast to previous studies [28], we propose soft relaxation of the discrete triangulation setting by learning the mesh connectivity distribution in an unsupervised manner. The proposed method can simplify meshes of any scale in real-time by using an extremely efficient point sampling method and a lightweight triangle classifier. To follow the initial mesh appearance, we constrain the distribution of the edges to the priors defined by the original mesh connectivity. The proposed method is fully-differentiable and can be adapted to any training procedure without a significant computational footprint. Finally, the proposed method can generalize to out-of-distribution samples exhibiting zero-shot capabilities.

The main contributions of this work are summarized as:

- We propose the first, to the best of our knowledge, learnable mesh simplification framework that is trained to both select vertices and generate the underlining triangulation of the surface.
- The proposed model is fully differentiable and can be directly adapted to any learnable framework.
- We introduce an efficient point selection method that extends the naive random sampling [15] to a trainable module that samples vertices from the underlying multinomial distribution.
- Finally, we showcase a simple but intuitive triangulation strategy that can be adapted to point cloud meshing.

2. Related Work

Mesh Simplification. Traditional simplification algorithms repeatedly decimate the input mesh according to a cost function to preserve its rendered appearance, until the desired simplification ratio is reached. Simplification methods can be distinguished in two major categories: *vertex clustering/decimation* and *edge collapse* methods. Vertex decimation methods rank vertices according to a heuristic geometric cost function, such as their distance from the average plane [34–36], to ensure that least important vertices will be decimated first. However, after every vertex deletion, a re-tessellation of the generated hole is required, thus, making such algorithms non-practical. On the other hand, edge collapse methods preserve the input topology by sequentially contracting pairs of vertices (i.e. edges). Hoppe *et al*. [14] pioneered an energy cost function defined over the edges that is attempted to be minimized in every contraction step. Following this idea, in the seminal works of [10, 33], each vertex was associated with the set of planes in its 1-ring neighborhood and was expressed by a fundamental *quadric* matrix. The authors showcased, that using the quadric matrix, the distance of a point from a set of planes can be expressed using the sum of their quadrics, which is known as Quadric Error Metric (QEM). Using this property, edges that introduce the minimum point-to-plane distance were the first to be collapsed. Several approaches have been built upon QEM to incorporate texture [1, 11], curvature [16, 17] and spectral properties [21, 23] or to speed-up the process using parallel processing [19, 26]. Recently, Hanocka *et al*. [13] proposed the utilization of an adaptive greedy edge collapse method as a learnable pooling strategy, where edge weights are learned through the network. However, apart from the inefficient greedy nature of the edge collapse methods, the resulting mesh faces can only be decimated approximately by a factor of two and thus limiting its applicability. Potamias *et al*. [28] proposed a learnable point cloud simplification technique that preserves the curvature features of the point cloud. They have also extended their method to mesh simplification task by utilizing off-the-self algorithms to triangulate the resulting simplified point cloud. Nevertheless, such methodology requires careful selection of the triangulation parameters for every sample. Importantly, the original topology of the mesh is neglected, i.e. the mesh connectivity might be totally different, and cannot be directly optimized in a learnable process.

Pooling. Inspired by the regular pooling layers in CNNs, several graph pooling operators have been introduced to reduce the size of graphs and enable hierarchical training in graph neural network (GNN) architectures. Most of these methods utilize non-trainable point selection methods, such as Graclus clustering algorithm [6–8] and Farthest Point

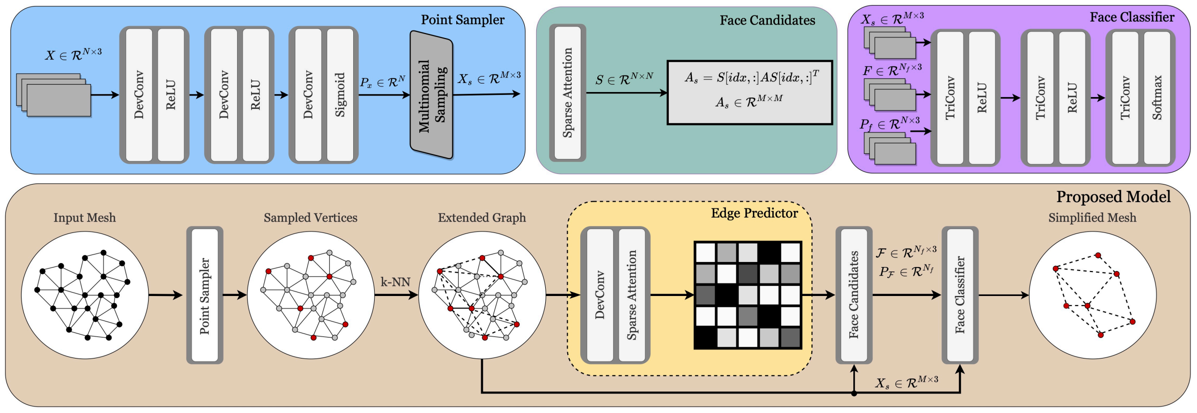
Point Sampler
$X \in \mathcal{R}^{N\times 3}$
DevConv
ReLU
DevConv
ReLU
DevConv
Sigmoid
$P_x \in \mathcal{R}^N$
Multinomial Sampling
$X_s \in \mathcal{R}^{M\times 3}$
Face Candidates
Sparse Attention
$S \in \mathcal{R}^{N\times N}$
$A_s = S[idx,:]AS[idx,:]^T$
$A_s \in \mathcal{R}^{M\times M}$
Face Classifier
$X_s \in \mathcal{R}^{M\times 3}$
$F \in \mathcal{R}^{N_f\times 3}$
$P_f \in \mathcal{R}^{N\times 3}$
TriConv
ReLU
TriConv
ReLU
TriConv
Softmax
Proposed Model
Input Mesh
Point Sampler
Sampled Vertices
k-NN
Extended Graph
Edge Predictor
DevConv
Sparse Attention
Face Candidates
$\mathcal{F} \in \mathcal{R}^{N_f\times 3}$
$P_{\mathcal{F}} \in \mathcal{R}^{N_f}$
Face Classifier
Simplified Mesh
$X_s \in \mathcal{R}^{M\times 3}$

Figure 2. Overview of the proposed model (bottom block). Initially, Point Sampler module (top left) samples a subset of the input vertices from the generated multinomial distribution. To avoid isolated nodes, a k-nn graph is constructed to extend the already existing edges of the graph. Following that, a sparse attention layer weights the connectivity between nearby vertices and generates a set of candidate triangles (top middle). Finally, Face Classifier module defines a graph over the proposed faces and assigns a probability to each triangle based on their relative features (top right).

Sampling (FPS) method [29], to generate hierarchical representations of the input graph. Ying *et al*. [40] proposed the first trainable task-dependent pooling layer by learning a clustering assignment matrix. However, the dense nature of the soft-clustering assignment matrix makes such approach infeasible for large scale graphs [3]. To tackle such limitation, several sparse pooling methods have been proposed that select Top-K nodes based on a learnable score [3, 9, 20]. However, Top-K approaches retain only a subset of the edge set of the input graph, leading to isolated nodes. Recently, Ranjan *et al*. [32] utilized a sparse soft clustering matrix to address the node connectivity issues of the previous methods. For additional details on graph pooling layers we refer the reader to [12]. In an abstract sense, one may regard mesh simplification as a pooling process, since the input topology is known and we seek to find its simplified version. However, in contrast to common graph pooling architectures, mesh simplification methods should also respect the surface properties of the mesh, such as smoothness and manifoldness. A natural approach to bypass the limitations is to attempt to bridge both words. In this work, we adapt graph pooling strategies along with geometric losses to tackle the problem of mesh simplification.

Learnable Triangulation. Albeit surface reconstruction method have been extensively studied over the years, less attention has been devoted to learnable and differentiable triangulation methods. Most of the existing techniques in the literature generate mesh surfaces by estimating implicit funtions [4, 27], calculating voxel grids and occupancy fields [22, 25] or deforming a template mesh [39]. Less progress has been established to the challenging task of direct point set triangulation, mainly due to the discrete nature of the edges that compose the mesh connectivity. PointTriNet [37] utilizes two models to suggest and classify triangles in local patches, enforcing the selection of watertight and manifold triangles using ad-hoc losses. A similar principle is also used in [24] where the candidate triangles are iteratively filtered using the estimated ratio of the geodesic versus the euclidean distance. Recently, there has been an exertion to employ traditional Delaunay surface triangulation into the learnable triangulation process. In [31], the authors propose to learn a parametrization that maps the input point patches to two-dimensional spaces and triangulate them using Delaunay technique. In a similar manner, a soft relaxation of weighted Delaunay triangulation was also proposed [30] that enables gradient flow, using an inclusion score to discard proposed triangles from the final mesh. Similarly to [31], the input point cloud is triangulated to a spectral partitioned 2D subspace produced using Least-Squares Conformal Map parametrization. In this paper we propose a modular architecture that directly learns to generate a triangulated version of the input vertex set without adhering to any kind of 2D projection and mapping.

3. Method

The architecture of the proposed model is composed by mainly three components: the *Point Sampler*, the *Edge Predictor* and the *Face Classifier*. All modules are fully differentiable and are trained in an end-to-end fashion. Figure 2 illustrates an overview of the proposed method.

3.1. Point Sampler

The first module of the proposed model is a network that samples the vertex set in a way that the structure of the mesh will remain intact. Previous works, utilized Farthest Point Sampling (FPS) to sample input point sets since it has been shown that it accurately preserves the underlying shape of the object [29]. However, the iterative nature of FPS is not scalable and thus impractical for large scale point sets. In contrast, as shown in [15], random sampling is extremely fast, size agnostic, holding an $\mathcal{O}(1)$ complexity opposed to the FPS's quadratic $\mathcal{O}(N^2)$ complexity. Based these observations, we propose to extend random sampling to a sophisticated learnable module that breaks the uniform hypothesis of random sampling and samples nodes under the assumption of a multinomial distribution. To do so, we train the Point Sample module to assign an inclusion score to each point in the vertex set.

Given that the simplified point set needs to approximate the structure of the input point set, the Point Sampler module needs to be trained to select points that provide the best coverage of the input space. Taking into account that nearby points will also have similar latent descriptors, we propose to utilize a graph neural network (GNN) approach that will intuitively provide shape insights for the vertices. We employ an update rule based on the relative vertex coordinates that can approximate the deviation of a point from its neighborhood as:

$$\mathbf{f}_i = \sigma\left(\mathbf{W}_\phi \max_{j \in \mathcal{N}(\mathbf{x}_i)} \mathbf{W}_\theta(\mathbf{x}_i - \mathbf{x}_j)\right) \tag{1}$$

where $\mathcal{N}(\mathbf{x}_i)$ denotes the neighbouring points of $\mathbf{x}_i$, $\mathbf{W}_\phi, \mathbf{W}_\theta$ are learnable parameters and $\sigma(\cdot)$ a non-linearity. We refer to this GNN layer as **DevConv**.

With such formulation, the advantage is two-fold. Primarily, the point descriptors may better describe the topological characteristics of a given point, where points in sharp and rough regions will receive larger values compared to smooth areas. Secondly, the sampling module gains robustness to noise, since the combination of the $maximum$ as aggregation function and the relative coordinates provides to the network the ability to easily identify outliers.

3.2. Edge Predictor

Following the Point Sampler, the Edge Predictor module is responsible to predict the connectivity between the sampled points. To do so, we initially extend the original mesh connectivity by inserting edges defined by k-nn graph over the sampled points to avoid isolated nodes in the final mesh. The extended graph $\mathcal{G}_{ext}$ is then processed by a DevConv layer followed by a sparse self-attention layer [38] that predicts the probability that the point $\mathbf{x}_i$ is connected with the point $\mathbf{x}_j$. Such probability is formulated as:

$$\mathbf{S}[i,j] = \frac{\exp\left((\mathbf{W}_q\mathbf{f}_j)^T(\mathbf{W}_k\mathbf{f}_i)\right)}{\sum\limits_{k \in \mathcal{N}(\mathbf{x}_i)} \exp\left((\mathbf{W}_q\mathbf{f}_j)^T(\mathbf{W}_k\mathbf{f}_k)\right)} \tag{2}$$

where $\mathbf{W}_q, \mathbf{W}_k$ are learnable parameters and $\mathbf{f}_i$, $\mathbf{f}_j$ are the features of points $\mathbf{x}_i, \mathbf{x}_j$.

To avoid having edges of equal probability between nearby points, we utilize DevConv to enable feature dissimilarity between them. Finally, the estimated adjacency matrix is defined using the product of the estimated probabilities and the original adjacency matrix, following the formulation of [40], as:

$$\mathbf{A}_s[i,j] = \mathbf{S}[i,:]\,\mathbf{A}\,\mathbf{S}[j,:]^T, \quad i,j \in \mathcal{G}_{ext} \tag{3}$$

Our motivation behind the use of the product between the estimated and the original connectivity is to enforce the edges of the simplified mesh to respect the original topology. Note that all of the aforementioned multiplication operations are between the sparse matrices and can be calculated very efficiently.

The set of candidate faces can be easily constructed from the non-zero entities of the simplified adjacency matrix $\mathbf{A_s}$. An initial inclusion probability p_t^{init} is assigned to each triangle defined as $p_t^{init} = \frac{1}{3}(\mathbf{A}_s[i,j] + \mathbf{A}_s[i,k] + \mathbf{A}_s[j,k])$, where i,j,k are the vertices of triangle t. This initial inclusion probability can be thought as a prior that will be invoked by the Face Classifier to produce the final (posterior) probability of the edge.

3.3. Face Classifier

The face classifier is responsible to assign to each triangle an inclusion score p_t that captures the probability that this triangle will be present in the simplified mesh. To estimate this probability, we first construct a k-nn graph $\mathcal{G}_{tri}$ that connects each candidate triangle with its k-neighbours based on their barycenter distances. Then a GNN module, namely **TriConv**, that acts on $\mathcal{G}_{tri}$ embeds faces to the latent space. To better capture the interactions of two triangles n and m in space, we utilize a relative position encoding $\mathbf{r}_{n,m}$ defined as:

$$\begin{gathered}\mathbf{r}_{n,m} = [(\mathbf{t}_n^{min} - \mathbf{t}_m^{min})||(\mathbf{t}_n^{max} - \mathbf{t}_m^{max})||(\mathbf{b}_n - \mathbf{b}_m)], \\ \mathbf{t}_n^{max} = max(\mathbf{e}_{ij}^n, \mathbf{e}_{ik}^n, \mathbf{e}_{jk}^n)\end{gathered} \tag{4}$$

where $\mathbf{t}_n^{max}, \mathbf{t}_n^{min} \in \mathcal{R}^3$, $\mathbf{e}_{ij}^n$ the edge vectors $\mathbf{x}_i - \mathbf{x}_j$ for triangle n; b_n, b_m the barycenters of triangles n, m and $||$ the concatenation operation. Finally, the update rule of TriConv can be defined as follows:

$$\mathbf{f}_n^{(l)} = \sum_{k \in \mathcal{N}(t_n)} MLP([\mathbf{r}_{n,k}||(\mathbf{f}_n^{(l-1)} - \mathbf{f}_k^{(l-1)})]) \tag{5}$$

where $\mathcal{N}(t_n)$ the neighborhood and $\mathbf{f}_n^{(l-1)}$ the previous feature of face n. In order to generate the final inclusion probability p_t we stack three TriConv layers toped with a softmax activation function. We use the prior probability p_t^{init} as the initial feature of each triangle.

3.4. Losses

To train the proposed framework we utilize a set of loss functions that oblige the simplified meshes to preserve the visual appearance of the originals. The basic idea underlying the utilized losses is to enforce the selection of salient points that are representative of the mesh structure and to penalize badly formed triangles.

Probabilistic Chamfer Distance: To improve uniform sampling we force the Point Selector to assign high probabilities to the points that cover the surface of the point cloud. We mathematically formulate this by using a modified probabilistic Chamfer distance:

$$\begin{aligned} d_{\mathcal{P},\mathcal{P}_s} = & \sum_{\mathbf{y}\in\mathcal{P}_s} p(\mathbf{y}) \min_{\mathbf{x}\in\mathcal{P}} \|\mathbf{x}-\mathbf{y}\|^2 \\ & + \sum_{\mathbf{x}\in\mathcal{P}} p(\mathbf{y}) \min_{\mathbf{y}\in\mathcal{P}_s} \|\mathbf{x}-\mathbf{y}\|^2 \end{aligned} \tag{6}$$

where $\mathcal{P}$ denotes the input vertex set, $\mathcal{P}_s$ the sampled points and $p(\mathbf{y})$ their respective probabilities. Note that this loss is applied only to the Point Sampler module and therefore can be pre-trained and adapted to any learnable framework.

Probabilistic Surfaces Distance: To avoid having triangles in regions that are not existing in the original mesh and penalize the presence of surface holes, we formulate a Chamfer-inspired loss that measures the distance between a ground truth and a probabilistic surface. In this setting, the forward term of the distance, i.e. the distance between the generated surface $\mathcal{S}_s$ and the original, enforces triangles to fit the ground truth surface $\mathcal{S}$. We may calculate this term by using the barycenters of the two surfaces as follows:

$$d^f_{\mathcal{S},\mathcal{S}_s} = \sum_{\hat{\mathbf{b}}\in\mathcal{S}_s} p_{\hat{\mathbf{b}}} \min_{\mathbf{b}\in\mathcal{P}} \|\hat{\mathbf{b}}-\mathbf{b}\|^2 \tag{7}$$

where $\mathbf{b}$ stands for the barycenters of the ground truth surface, $\hat{\mathbf{b}}$ the barycenters of the generated triangles with respective probabilities $p_{\hat{\mathbf{b}}}$. In this manner, barycenters of triangles in non existing regions, e.g. a triangle connecting the dogs legs, will have greater distance compared to barycenters of triangles in existing regions of the ground truth surface.

In contrary to the forward term, the reverse term of the distance function aims to penalize areas with small probabilities, i.e. areas that when discarded may result into the introduction of surface holes. We can capture this mathematically as follows:

$$\begin{aligned} d^r_{\mathcal{S},\mathcal{S}_s} = & \sum_{y\in\mathcal{S}_s} p_{\mathbf{y}} \min_{\mathbf{x}\in\mathcal{P}} \|\mathbf{x}-\mathbf{y}\|^2 \\ & + (1-p_{\mathbf{y}}) \frac{1}{k} \sum_{k} p_{t_k} \|\mathbf{x}_{t_k}-\mathbf{y}\|^2 \end{aligned} \tag{8}$$

where $\mathbf{x}$ denotes a point from the ground truth surface $\mathcal{S}$ and $\mathbf{y}$ a point from the generated surface $\mathcal{S}_s$ with probability $p_{\mathbf{y}}$. The second term of equation (8) estimates the average distance between point $\mathbf{y}$ and its k-nearest triangles t_k in the generated surface $\mathcal{S}_s$ apart from the one that point $\mathbf{y}$ belongs to. This last term can be intuitively described as the error introduced when the triangle in which $\mathbf{y}$ belongs is not present in the generated triangulation. To make the reverse term robust, we sample a sufficient amount of points from each generated triangle.

Triangle Collision: To avoid having triangles that penetrate each other we introduce a loss term that directly penalizes the probabilities of such triangles. We measure the collision of a triangle in terms of line segments (i.e. edges of nearby triangles) penetrating its surface. In particular, we compute the planes defined by each face and we penalize nearby triangles formed from edges that penetrate this plane. The penalty applied to such irregular triangle is proportional to the number of planes that it penetrates and it is defined as:

$$\mathcal{L}_c = \frac{1}{|\mathcal{F}_s|} \sum_{t\in\mathcal{F}_s} p_t m_c(t) \tag{9}$$

where p_t denotes the probability of triangle t, $m_c(t)$ the number of faces penetrated by triangle t and $\mathcal{F}_s$ the set of generated triangles.

Edge Crossings and Overlaping triangles: Although triangle collision loss may be sufficient to penalize triangles that penetrate the surface of their neighboring triangles, it can not capture and penalize overlaping triangles with parallel planes. To address this limitation we introduce two additional losses that penalize such scenarios, namely the edge crossings loss $\mathcal{L}_e$ and the overlap loss $\mathcal{L}_o$. We calculate edge crossings of line segments (edges) of nearby triangles and we directly penalize triangles that carry an edge that crosses another edge. Finally, to avoid overlaping triangles in space, we sample a sufficient number of points from each generated face and compute an estimate that belongs to a given face. This can be efficiently implemented by measuring the sum of the areas produced by the sampled point and the vertices of each of the k-nearest triangles. Similar to collision loss, the penalty applied to each triangle is proportional to the number of faces that it penetrates.

Overall Objective Finally, the overall loss to be minimized is formed as:

$$\mathcal{L} = d_{\mathcal{P}_1,\mathcal{P}_2} + d^f_{\mathcal{S}_1,\mathcal{S}_2} + d^r_{\mathcal{S}_1,\mathcal{S}_2} + \lambda_c\mathcal{L}_c + \lambda_e\mathcal{L}_e + \lambda_o\mathcal{L}_o \tag{10}$$

where $\lambda_c, \lambda_e, \lambda_o$ are hyperparameters that scale the loss functions.

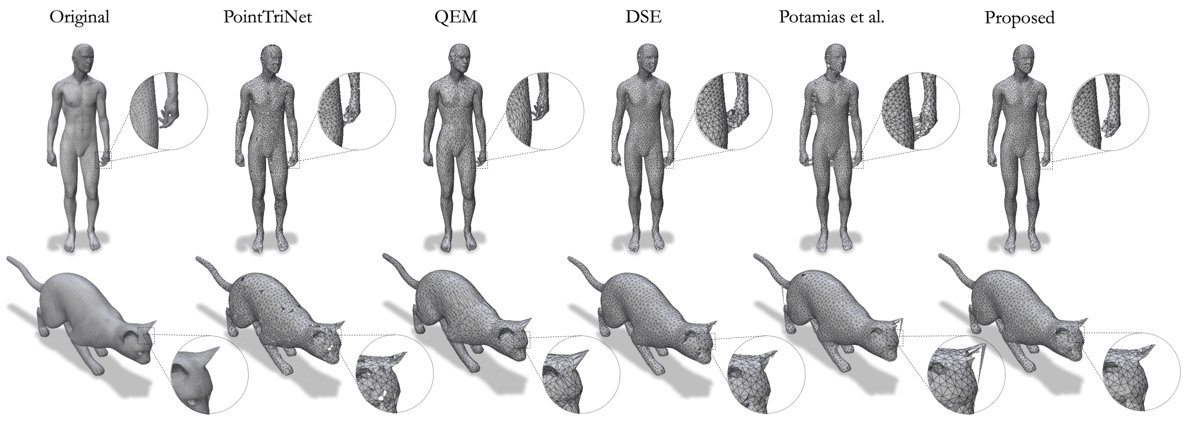

Figure 3. Qualitative comparison of the proposed and the baseline methods. The top row contains a human shape simplified by 90% and the bottom row shows a cat model simplified by 80%. Figure better viewed in zoom.

Table 1. Quantitative comparison of the proposed and the baseline methods under several simplification ratios. Best approaches are highlighted in **bold** and second best in **blue**.

	N_s/N_{org} = 0.05					N_s/N_{org} = 0.1					N_s/N_{org} = 0.2					N_s/N_{org} = 0.5				
Method	CD	WA	LE	NE	Time	CD	WA	LE	NE	Time	CD	WA	LE	NE	Time	CD	WA	LE	NE	Time
PointTriNet [37]	1.06	12.14	0.98	0.20	107.2	0.47	11.64	0.52	0.17	238.1	0.21	11.48	0.27	0.13	581.9	0.08	14.92	0.12	0.08	1333.3
QEM [10]	**0.45**	**0.00**	**0.94**	**0.14**	**45.6**	**0.22**	**0.00**	0.53	**0.10**	**31.1**	**0.11**	**0.00**	0.27	**0.07**	**28.4**	**0.05**	**0.00**	0.13	**0.03**	**25.6**
DSE [30]	1.39	6.64	0.95	0.23	271.8	0.51	4.38	**0.50**	0.19	490.7	0.24	3.12	**0.26**	0.15	941.0	0.07	**2.39**	**0.12**	0.08	2245.2
Potamias el al. [28]	10.47	8.53	1.23	0.21	105.2	0.83	8.39	0.76	0.17	149.3	0.49	4.72	0.43	0.14	158.7	0.20	4.67	0.22	0.10	183.7
Proposed	**1.02**	**2.17**	**0.90**	**0.19**	**4.1**	**0.42**	**2.21**	**0.47**	**0.15**	**4.2**	**0.19**	**2.49**	**0.24**	**0.11**	**4.2**	**0.06**	3.57	**0.10**	**0.06**	**4.4**

4. Experiments

In this section we evaluate the simplified meshes produced by the proposed framework. We initially examine the effect of *Point Sampler* and then we assess the quality of the produced triangulation along with an assessment of the respective run times. For additional experimental results we refer the reader to the supplementary material.

Dataset: To train the proposed method we utilized the benchmark TOSCA [2] dataset that contains 80 high resolution meshes. We used the same train test split as in [28] and test the model using topologies not present in the training set. In such setting, the devised model may be directly utilized to out-of-distribution meshes and generalize across different topologies.

Implementation: We trained our model for 150 epochs with learning rate of $1e-5$ and a weight decay of 0.99 on every epoch using the Adam optimizer [18]. The generated simplified meshes are produced by selecting only the faces with a probability above 0.5. We further constrain the generated meshes to be manifold by selecting for each edge the two incident to it faces with the highest probability.

Baselines: For comparison, we selected several baselines with different properties. In particular, we compared against the popular quadric mesh simplification (QEM) [10] which remains among the most popular and efficient methods in mesh simplification. Additionally, we selected two learnable and differentiable triangulation methods, i.e. PointTriNet [37] and DSE [30], to triangulate point clouds sampled with FPS and thus introduce an alternative for differentiable mesh simplification. Finally, we compared the proposed method against a recently introduced learnable point cloud simplification method [28], which utilizes the Ball-Pivoting algorithm to triangulate the simplified point clouds.

4.1. Evaluation of the Simplified Meshes

To assess the triangulation performance, we measure the percentage of non-watertight edges (WA), i.e the edges which have one, three or more incident faces. Compared to manifoldness metric, WA better assesses the triangulation quality when holes are present, given that the edges surrounding a hole are still manifold (a watertight edge is manifold but a manifold edge is not guaranteed to be watertight). Additionally, to evaluate the simplification performance of each method we sample $50K$ points from the surface of every simplified mesh and measure the Chamfer distance (CD) and normal dissimilarity (NE) from its corresponding points in the original mesh. Finally, we calculate the MSE error between the first 200 eigenvectors of the Laplacian of the original and the simplified model (LE). In Table 1 we quantitatively compare the proposed method with the aforementioned baselines. Although the iterative QEM method better preserves the appearance of the simpli-

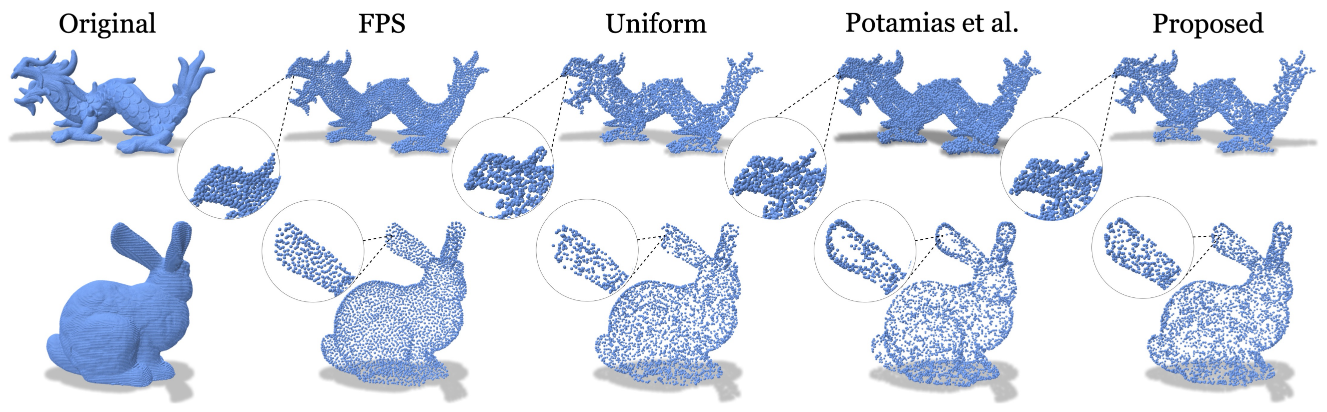

Figure 4. Qualitative assessment of the proposed point sampling module. The proposed Point Sampler selects points that preserve both the structure and the details of the input cloud. The proposed method better maintains the structure of the object compared to uniform and [28] counterparts and improves the smooth point clouds produced by FPS module. The top row shows a dragon point cloud simplified to 5% of its input size where as the bottom row shows the bunny shape simplified to 20% of its input size. Please note that both shapes are not present in the TOSCA dataset.

Table 2. Quantitative evaluation of the point sampling module at different simplification ratios. The right part of the table includes the simplification performance of the various methods when tested to noisy point clouds.

	w/o Noise									w Noise								
	N_s/N_{org} = 0.8			N_s/N_{org} = 0.2			N_s/N_{org} = 0.05			N_s/N_{org} = 0.8			N_s/N_{org} = 0.2			N_s/N_{org}=0.05		
Method	CD	Curv	Time	CD	Curv	Time	CD	Curv	Time	CD	Curv	Time	CD	Curv	Time	CD	Curv	Time
FPS	**0.01**	2.21	21.2	**0.81**	2.99	7.84	**4.02**	3.42	3.86	2.11	4.07	21.2	3.47	4.35	7.84	7.31	4.79	3.86
Uniform	0.24	2.19	**0.05**	1.85	2.44	**0.04**	6.71	2.73	**0.03**	2.58	3.71	**0.05**	3.67	3.97	**0.04**	7.43	4.20	**0.03**
Potamias et al. [28]	0.03	**2.14**	120.	1.18	2.27	35.1	4.78	2.51	17.7	2.11	3.14	120.	3.38	3.65	35.1	7.61	4.12	17.7
Point Sampler	0.05	2.16	0.05	1.22	**2.24**	0.09	5.12	**2.51**	0.09	**1.99**	**3.12**	0.05	**3.21**	**3.18**	0.09	**7.18**	**4.01**	0.09

fied mesh, the proposed method achieves smaller Laplacian error while at the same time attain competing performance over all error metrics. On the contrary, the proposed method outperforms all differentiable methods and overcomes the limitations of previous triangulation approaches. In particular, as can be easily observed in Figure 3, the proposed method has very few holes compared to PointTriNet [37] as well as less triangles in non-existing regions, such as the triangles occurring between the thigh and the hand (top row Figure 3), due to the probabilistic surface distance loss that penalizes such triangles. For further qualitative assessment see also Figure 1.

4.2. Time Performance

One of the most prominent applications of mesh simplification is inevitably rendering. Real-time rendering requires lightweight model structures, therefore simplification algorithms are commonly equipped in rendering pipelines. To this end, the time performance is of crucial importance for the simplification process. To assess time performance, we measured the execution time for each method to simplify 20 meshes under different simplification ratios. Experimental result presented in Table 1 showcase the efficiency of the proposed method, outperforming its baseline counterparts by a large margin. In particular, the proposed method runs up to 10× faster than the optimized QEM method and at least 100× faster than its faster differentiable counterpart. Another important feature of the proposed method is that it remains almost unaffected by the mesh scale, due to the efficient point sampler and the lightweight structure of the face classifier. In summary, the results highlight the fact that the proposed method could be directly plugged into any rendering process without introducing any significant overhead.

4.3. Evaluation of Point Sampler

Point Sampler is responsible for the selection of the vertices to be maintained in the simplified mesh. To assess the performance of the sampling module we measure the structural error in terms of i) the Chamfer distance (CD), ii) the details preserved using the two-sided curvature error as suggested in [28] and iii) the time required to simplify the input point cloud (in seconds) under several simplification ratios. We compare the proposed method with FPS [29], uniform random sampling as utilized in [15], and a recently introduced point cloud simplification module [28]. In Figure 4 we qualitative compare the simplified point clouds. Quantitative results are presented in Table 2 showcasing that the proposed point sampler outperforms the uniform random sampler as proposed in [15] and also demonstrate a balance between the smooth and the sharp results produced by FPS and Potamias *et al*. [28]. Importantly, the proposed method remains unaffected from the size of the input, achieving a sampling of 420%- to 2400%- times faster than FPS and Potamias *et al*. [28]. This result clearly demonstrates that the proposed method can be directly utilized to sample large-scale point clouds with the minimal computational overhead, greatly advancing the naive random sampling approach [15]. Finally, we assess the performance of the proposed sampling module under noisy conditions by training on clean datasets and testing on noisy point clouds by adding zero mean and unit std Gaussian noise to the data. (right block of Table 2). It can be easily observed that the proposed method is less affected by the presence of noise compared to its counterparts by virtue of the DevConv, which encodes points based on the maximum relative features of the neighborhood.

4.4. Curvature-based Simplification

A significant property of the proposed method is that all of its components are fully differentiable. Thus, it can be seamlessly integrated to an arbitrary iterative framework which requires gradients to flow throughout the optimization process. In this experiment, we attempt to exploit the capability of the model to be adapted to a trainable pipeline that generates customized simplification. In particular, we fine-tuned the proposed method to generate simplified meshes that preserve the curvature of the original. To achieve this, we utilized a loss term that measures the curvature difference between the original and the simplified meshes as proposed in [28]. Experimental results revealed that the fine-tuned method achieved an improvement of 40% (in average) to the curvature error compared to the original version. An example is illustrated in Figure 5 where it can be easily observed that the fine-tuned version focuses on preserving the rough details of the original mesh (such as the eyes and the nose-tip) compared to the smooth mesh produced by the untouched version of the proposed method.

4.5. Evaluation of intrinsic distances

To evaluate the distortion of intrinsic properties in the simplified mesh we rely on geodesic and spectral distances, measured between vertices in a spotted area of the surface. The biharmonic spectral distance is calculated as in [21]. A qualitative color-coded comparison of the cat shape simplified by 90% between QEM and the proposed method is illustrated in Figure 6. Although, both methods manage to satisfactorily preserve the geodesic distances, the QEM approach introduces enough distortion to the biharmonic spectral distances. Comparisons with additional baselines are provided in the supplementary material.

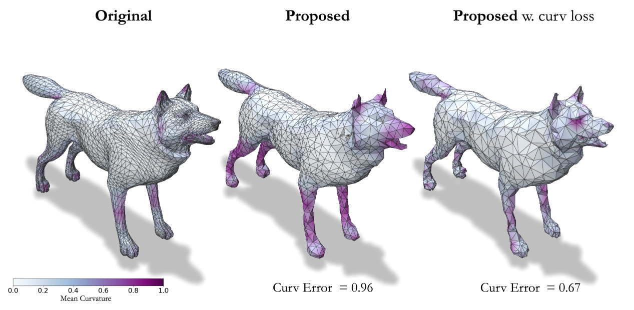

Figure 5. Fine-tuning the proposed method for curvature driven simplification.

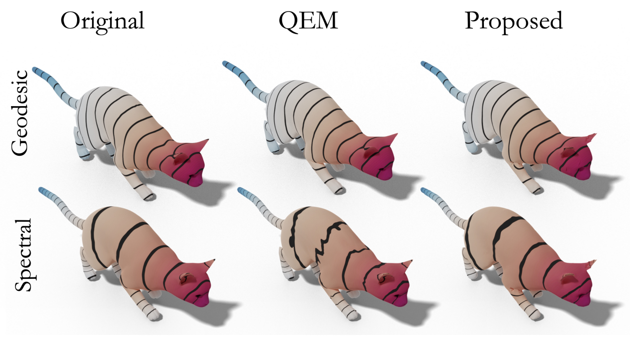

Figure 6. Geodesic and Spectral distance comparison between QEM and the proposed method. Distances are measured from the nose-tip of the cat.

5. Conclusion and Limitations

In this work we attempt to mark a step towards a totally learnable mesh simplification framework. We proposed the first differentiable mesh simplification model based on the advances of graph neural networks. The run-time efficiency and lightweight structure of the proposed model enables its direct use in a wide range of differentiable applications. The proposed method outperforms all of its differentiable counterparts. A limitation of the proposed method is that, although we enforce the model using tailor-made loss function to preserve the topology and the manifoldness of the generated meshes, it can not be explicitly guaranteed. Inevitably, QEM produces smother watertight surfaces with finer details compared to the proposed method. However, QEM comes with a greater computational cost, along with a non-differentiable nature that limits its range of applications. We believe that the rationale underlying the proposed method and the presented findings of this paper will benefit the 3D computer vision community.

Acknowledgements. Dr. Stefanos Zafeiriou acknowledges support from EPSRC fellowship Deform (EP/S010203/1).

References

[1] Kanchan Bahirat, Chengyuan Lai, Ryan P. Mcmahan, and Balakrishnan Prabhakaran. Designing and evaluating a mesh simplification algorithm for virtual reality. *ACM Trans. Multimedia Comput. Commun. Appl.*, 14(3s), June 2018. 2

[2] Alexander M Bronstein, Michael M Bronstein, and Ron Kimmel. *Numerical geometry of non-rigid shapes*. Springer Science & Business Media, 2008. 6

[3] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018. 3

[4] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019. 3

[5] Angela Dai and Matthias Nießner. Scan2mesh: From unstructured range scans to 3d meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5574–5583, 2019. 2

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. 2

[7] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11):1944–1957, 2007. 2

[8] Xiang Feng, Wanggen Wan, Richard Yi Da Xu, Haoyu Chen, Pengfei Li, and J Alfredo Sánchez. A perceptual quality metric for 3d triangle meshes based on spatial pooling. *Frontiers of Computer Science*, 12(4):798–812, 2018. 2

[9] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *International Conference on Machine Learning*, pages 2083–2092, 2019. 3

[10] 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. 1, 2, 6

[11] Michael Garland and Paul S Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings Visualization'98 (Cat. No. 98CB36276)*, pages 263–269. IEEE, 1998. 2

[12] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *arXiv preprint arXiv:2110.05292*, 2021. 3

[13] 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):90:1–90:12, 2019. 2

[14] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on computer graphics and interactive techniques*, pages 71–78, 1992. 2

[15] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11108–11117, 2020. 2, 4, 7, 8

[16] SJ Kim, WK Jeong, and CH Kim. Lod generation with discrete curvature error metric. In *Proceedings of Korea Israel Bi-National Conference*, pages 97–104. Citeseer, 1999. 2

[17] Sun-Jeong Kim, Chang-Hun Kim, and David Levin. Surface simplification using a discrete curvature norm. *Computers & Graphics*, 26(5):657–663, 2002. 2

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[19] Hyunho Lee and Min-Ho Kyung. Parallel mesh simplification using embedded tree collapsing. *The Visual Computer*, 32(6):967–976, 2016. 2

[20] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3734–3743. PMLR, 09–15 Jun 2019. 3

[21] Thibault Lescoat, Hsueh-Ti Derek Liu, Jean-Marc Thiery, Alec Jacobson, Tamy Boubekeur, and Maks Ovsjanikov. Spectral mesh simplification. *Computer Graphics Forum*, 39(2):315–324, 2020. 2, 8

[22] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2916–2925, 2018. 3

[23] Hsueh-Ti Derek Liu, Alec Jacobson, and Maks Ovsjanikov. Spectral coarsening of geometric operators. *ACM Trans. Graph.*, 38(4), July 2019. 2

[24] Minghua Liu, Xiaoshuai Zhang, and Hao Su. Meshing point clouds with predicted intrinsic-extrinsic ratio guidance. In *European Conference on Computer Vision*, pages 68–84. Springer, 2020. 3

[25] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 3

[26] Alexandros Papageorgiou and Nikos Platis. Triangular mesh simplification on the gpu. *The Visual Computer*, 31(2):235–244, 2015. 2

[27] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019. 3

[28] Rolandos Alexandros Potamias, Giorgos Bouritsas, and Stefanos Zafeiriou. Revisiting point cloud simplification: A learnable feature preserving approach. *arXiv preprint arXiv:2109.14982*, 2021. 2, 6, 7, 8

[29] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on

point sets in a metric space. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 3, 4, 7

[30] Marie-Julie Rakotosaona, Noam Aigerman, Niloy Mitra, Maks Ovsjanikov, and Paul Guerrero. Differentiable surface triangulation. *arXiv preprint arXiv:2109.10695*, 2021. 2, 3, 6

[31] Marie-Julie Rakotosaona, Paul Guerrero, Noam Aigerman, Niloy J Mitra, and Maks Ovsjanikov. Learning delaunay surface elements for mesh reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22–31, 2021. 2, 3

[32] Ekagra Ranjan, Soumya Sanyal, and Partha Pratim Talukdar. ASAP: Adaptive structure aware pooling for learning hierarchical graph representations. *arXiv preprint arXiv:1911.07979*, 2019. 3

[33] Rémi Ronfard and Jarek Rossignac. Full-range approximation of triangulated polyhedra. In *Computer Graphics Forum*, volume 15, pages 67–76. Wiley Online Library, 1996. 2

[34] Jarek Rossignac and Paul Borrel. Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in computer graphics*, pages 455–465. Springer, 1993. 1, 2

[35] William J Schroeder. A topology modifying progressive decimation algorithm. In *Proceedings. Visualization'97 (Cat. No. 97CB36155)*, pages 205–212. IEEE, 1997. 2

[36] William J Schroeder, Jonathan A Zarge, and William E Lorensen. Decimation of triangle meshes. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70, 1992. 2

[37] Nicholas Sharp and Maks Ovsjanikov. Pointtrinet: Learned triangulation of 3d point sets. In *European Conference on Computer Vision*, pages 762–778. Springer, 2020. 2, 3, 6, 7

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 4

[39] Chao Wen, Yinda Zhang, Zhuwen Li, and Yanwei Fu. Pixel2mesh++: Multi-view 3d mesh generation via deformation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1042–1051, 2019. 3

[40] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. 3, 4