Learning to Segment 3D Point Clouds in 2D Image Space

Yecheng Lyu* Xinming Huang Ziming Zhang
Worcester Polytechnic Institute
{ylyu, xhuang, zzhang15}@wpi.edu

Abstract

In contrast to the literature where local patterns in 3D point clouds are captured by customized convolutional operators, in this paper we study the problem of how to effectively and efficiently project such point clouds into a 2D image space so that traditional 2D convolutional neural networks (CNNs) such as U-Net can be applied for segmentation. To this end, we are motivated by graph drawing and reformulate it as an integer programming problem to learn the topology-preserving graph-to-grid mapping for each individual point cloud. To accelerate the computation in practice, we further propose a novel hierarchical approximate algorithm. With the help of the Delaunay triangulation for graph construction from point clouds and a multi-scale U-Net for segmentation, we manage to demonstrate the state-of-the-art performance on ShapeNet and PartNet, respectively, with significant improvement over the literature. Code is available at https://github.com/Zhang-VISLab.

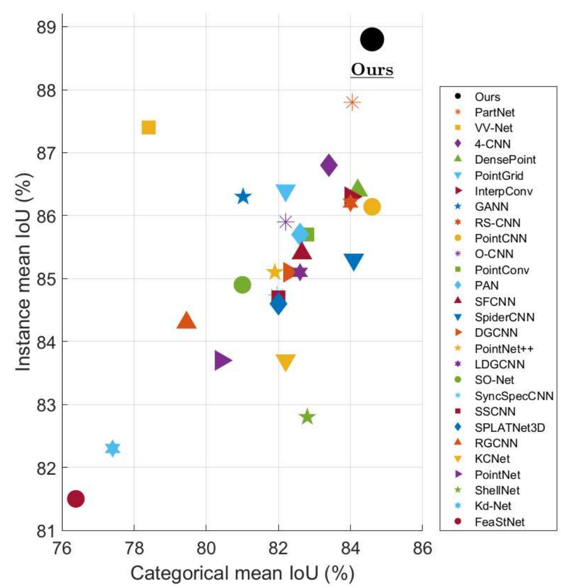

Figure 1: State-of-the-art part segmentation performance comparison on ShapeNet, where IoU denotes intersection-over-union.

1. Introduction

Recently point cloud processing has been attracting more and more attention [45, 44, 17, 46, 10, 61, 34, 25, 57, 29, 66, 30, 73, 72, 33, 71, 18, 39, 27, 38, 60, 28, 53, 47, 70]. As a fundamental data structure to store the geometric features, a point cloud saves the 3D positions of points scanned from the physical world as an orderless list. In contrast, images have regular patterns on 2D grid with well-organised pixels in local neighborhood. Such local regularity is beneficial for fast 2D convolution, leading to well-designed convolutional neural networks (CNNs) such as FCN [35], GoogleNet [54] and ResNet [16] that can efficiently and effectively extract local features from pixels to semantics with state-of-the-art performance for different applications.

Motivation. In fact PointNet[1] [45] for point cloud classification and segmentation can be re-interpreted from the perspective of CNN. In general, PointNet projects each 3D (x, y, z)-point into a higher dimensional feature space using a multilayer perceptron (MLP) and pools all the features from a cloud globally as a cloud signature for further usage. As an equivalent CNN implementation, one can construct an (x, y, z)-image with all the 3D points as the pixels in a random order and $(0, 0, 0)$ for the rest of the image, and apply 1×1 convolutional kernels sequentially to the image, followed by a global max-pooling operator. Different from conventional RGB images, here (x, y, z)-images define a new 2D image space with x, y, z as channels. Same image representation has been explored in [37, 36, 41, 64, 65] for LiDAR points. Unlike CNNs, PointNet lacks of the ability of extracting local features that may limit its performance.

This observation inspires us to investigate whether in the literature there exists a state-of-the-art method that applies conventional 2D CNNs as backbone to image representations for 3D point cloud segmentation. Surprisingly, as we summarize in Table 1, we can only find a few, indicating that currently such integrated methods for point cloud segmen-

*Part of this work was done when the author was an intern at Mitsubishi Electric Research Laboratories (MERL).

[1]For simplicity in our explanation, we assume no bias term in PointNet.

tation may be significantly underestimated. Clearly the key challenge for developing such integrated methods is:

How to effectively and efficiently project 3D point clouds into a 2D image space so that we can take advantage of local pattern extraction in conventional 2D CNNs for point cloud semantic segmentation?

Approach. The question above is nontrivial. A bad projection function can easily lead to the loss of structural information in a point cloud with, for instance, many point collisions in the image space. Such structural loss is fatal as it may introduce so much noise that the local patterns in the original cloud are completely changed, leading to poor performance even using 2D conventional CNNs. Therefore, a good point-to-image projection function is the key to bridge the gap between the point cloud inputs and 2D CNNs.

At the system level, our integrated method is as follows:

Step 1. Construct graphs from point clouds.

Step 2. Project graphs into images using graph drawing.

Step 3. Segment points using U-Net.

We are motivated by the graph visualization techniques in *graph drawing*, an area of mathematics and computer sciences whose goal is to present the nodes and edges of a graph on a plane with some specific properties [7, 49, 21, 11]. Particularly the *Kamada-Kawai (KK)* algorithm [21] is one of the most widely-used undirected graph visualization techniques. In general, the KK algorithm defines an objective function that measures the energy of each graph layout *w.r.t.* some graph distance, and searches for the (local) minimum that gives a reasonably good 2D visualization. Note that the KK algorithm works in a continuous 2D space, rather than 2D grid (*i.e.,* a discrete space).

Therefore, intuitively we propose an integer programming (IP) to enforce the KK algorithm to learn projections on 2D grid, leading to an NP-complete problem [63]. Considering that the computational complexity of the KK algorithm is at least $O(n^2)$ [24] with the number of nodes n in a graph (*e.g.,* thousands of points in a cloud), it would be still too expensive to compute even if we relax the IP with rounding.

In order to accelerate the computation in our approach, we follow the hierarchical strategy in [12, 40, 19] and further propose a novel hierarchical approximation with complexity of $O(n^{\frac{L+1}{L}})$, roughly speaking, where L denotes the number of the levels in the hierarchy. In fact, such a hierarchical scheme can also help us reduce the complexity in graph construction from point clouds using Delaunay triangulation [9] with worst-case complexity of $O(n^2)$ for 3D points [1].

Once we learn the graph-to-grid projection for a point cloud, we accordingly generate an (x, y, z)-image by filling it in with 3D points and zeros. We further feed these image representations to a multi-scale U-Net [48] for segmentation.

Performance Preview. To demonstrate how well our approach works, we summarize 32 state-of-the-art performance on a benchmark data set, ShapeNet [69], in Fig. 1 and compare ours with these results under the same training/testing protocols. Clearly our results are significantly better than all the others with large margins. Similar observations have been made on PartNet [71] as well. Please refer to our experimental section for more details.

Contributions. In summary, our key contributions in this paper are as follows:

- We are the *first*, to the best of our knowledge, to explore the graph drawing algorithms in the context of learning 2D image representations for 3D point cloud segmentation.
- We accordingly propose a novel hierarchical approximate algorithm that accounts for computation to map point clouds into image representations as well as preserving the local information among the points in each cloud.
- We demonstrate the state-of-the-art performance on both ShapeNet and PartNet with significant improvement over the literature for 3D point cloud segmentation, using the integrated method of our graph drawing algorithm with the Delaunay triangulation and a multi-scale U-Net.

2. Related Work

Table 1 summarizes some existing works. In particular,

Representations of 3D Point Clouds. Voxels are popular choices because they can benefit from the efficient CNNs. PointGrid [27], O-CNN [60], VV-Net [39] and InterpConv [38] sample a point cloud in volumetric grids and apply 3D CNNs. Some other works represent a point cloud in specific 2D domains and perform customized network operators [53, 47, 70]. However, these works have difficulty in sampling from a non-uniformly distributed point cloud and result in a serious problem of point collisions. Graph-based approaches [56, 20, 67, 43, 31, 62, 50, 32, 59, 26] construct graphs from point clouds for network processing by sampling all points as graph vertices. However, they often struggle in assigning edges between the graph vertices. There also exist some works [45, 44, 17, 46, 10, 61, 34, 25, 57, 29, 66, 30, 73, 72, 33, 71] that directly use points as network inputs. Though they do not have to consider the sampling and local connections among the points, significant effort has been made to hierarchically partition and extract features from the local point sets. FoldingNet [68] introduced 2D grid as a latent space, rather than the output space, to capture the geometry of a point cloud.

There are some works in the literature of light detection and ranging (LiDAR) point processing that utilize depth images [52] or (x, y, z)-images [37, 36, 41, 64, 65] generated from LiDAR points for training networks. In Table 1 we summarize these works as well, even though they are not developed for point cloud segmentation.

In contrast, we propose an efficient hierarchical approximate algorithm with Delaunay triangulation to map each

Table 1: Summary of state-of-the-art methods for point segmentation.

Method	Raw Data	Data-to-Input Mapping	Network Input	Network Architecture
Ours	**Point Cloud**	**Graph Drawing**	**(x,y,z)-Image**	**Multi-scale U-Net**
Lyu *et al.* [37]	LiDAR Frame	Sphere Mapping	(x,y,z,ϕ,θ,ρ,i)-Image	2D CNN
ChipNet [36]	LiDAR Frame	Sphere Mapping	(x,y,z,ϕ,θ,ρ,i)-Image	2D CNN
LoDNN [6]	LiDAR Frame	2D Grid Samp.	Statistics-Image	2D CNN
RangeNet++ [41]	LiDAR Frame	Sphere Mapping	(r,x,y,z,i)-Image	2D CNN
SqueezeSeg [64]	LiDAR Frame	Sphere Mapping	(x,y,z,i)-Image	2D CNN
SqueezeSegV2 [65]	LiDAR Frame	Sphere Mapping	(x,y,z,i)-Image	2D CNN
PointNet [45]	Point Cloud	-	Point	MLP
JSIS3D [44]	Point Cloud	-	Point	MT-PNet
PCNN [17]	Point Cloud	-	Point	Pointwise Conv.
PointNet ++ [46]	Point Cloud	FPS	Point	MLP
SRN [10]	Point Cloud	FPS	Point	SRN
SGPN [61]	Point Cloud	FPS	Point	MLP
RS-CNN [34]	Point Cloud	Ball Query	Point	RS Conv.
A-CNN [25]	Point Cloud	Ball Query	Point	A-CNN
KPConv [57]	Point Cloud	Ball Query	Point	KPConv
So-Net [29]	Point Cloud	KNN	Point	SOM
PointConv [66]	Point Cloud	KNN	Point	PointConv
PointCNN [30]	Point Cloud	KNN	Point	$\mathcal{X}$-Conv
PointWeb [73]	Point Cloud	KNN	Point	AFA
ShellNet [72]	Point Cloud	KNN	Point	ShellNet
DensePoint [33]	Point Cloud	Rand. Samp.	Point	PConv.
PartNet [71]	Point Cloud	Latent Tree	Point	MLP
Kd-Net [23]	Point Cloud	Kd-Tree	Point	ConvNets
MAP-VAE [15]	Point Cloud	Latent Tree	Point	GRU
RGCNN [56]	Point Cloud	Complete Graph	Graph	Graph Conv.
Point-Edge [20]	Point Cloud	FPS	Graph	Point-Edge Net
SpiderCNN [67]	Point Cloud	KNN	Graph	SpiderConv
PAN [43]	Point Cloud	KNN	Graph	Point Atrous Conv.
GANN [31]	Point Cloud	KNN	Graph	Graph Attention
DG-CNN [62]	Point Cloud	KNN	Graph	Edge-Conv
Kc-Net [50]	Point Cloud	KNN	Graph	MLP
HDGCN [32]	Point Cloud	KNN	Graph	MLP
GAC-Net [59]	Point Cloud	Rand. Samp.	Graph	Graph Attention
SPGraph [26]	Point Cloud	Voronoi Adj. Graph	Graph	GRU
RS-Net [18]	Point Cloud	3D Grid Samp.	Voxel	RNN
VV-Net [39]	Point Cloud	3D Grid Samp.	Voxel	VAE
PointGrid [27]	Point Cloud	3D Grid Samp.	Voxel	3D CNN
InterpConv [38]	Point Cloud	3D Grid Interpolate	Voxel	InterpConv
O-CNN [60]	Point Cloud	Octree	Voxel	MLP
Ψ-CNN [28]	Point Cloud	Spherical 3D Grid	Voxel	Spherical Conv.
SPLATNet [53]	Point Cloud	Lattice Interpolate	Lattice	Bilateral Conv.
SFCNN [47]	Point Cloud	Sphere Mapping	Sphere	Sph. Fractal Conv.
SyncSpecCNN [70]	Point Cloud	3D Grid Samp.	Spectral	SpecTN

point cloud onto a 2D image space.

Network Architectures. Network operations are the key to hierarchically learn the local context and perform semantic segmentation on point clouds. Grid-based approaches usually apply regular 2D or 3D CNNs on the grid representations. Graph-based approaches usually apply customized convolutions on graph representations. For point-based approaches, MLP is the most widely used network. For some other point-based approaches, customized convolution operators are designed as well to support their own network architectures. Recurrent neural networks (RNNs) are applied in some works to handle the unfixed-sized point inputs. Please refer to Table 1 for more references.

In contrast, we apply the classic U-Net to our image representations for point cloud segmentation. In our ablation study later, we also test several alternative 2D CNN architectures and all of them get comparable results to the literature.

Graph Drawing. According to the purposes of graph layout, there exist two families of graph drawing algorithms, in general. N-planar graph [49] focuses on presenting the graph on a plane with least edge intersections regardless the implicit topological features. Force-directed approaches such the KK algorithm, on the other hand, focus on minimizing the difference of graph node adjacency before and after 2D layout. Fruchterman-Reingold (FR) [12], FM^3 [40] and ForceAtlas2 [19] speed up the force-directed layout computation for large-scale graphs by introducing hierarchical schemes and optimized iterating functions.

Note that graph drawing can be considered as a subdiscipline of network embedding [14, 8, 5] whose goal is to find a low dimensional representation of the network nodes in some metric space so that the given similarity (or distance) function is preserved as much as possible. In summary, graph drawing focuses on the 2D/3D visualization of graphs, while network embedding emphasizes the learning of low dimensional graph representations.

In this paper, we propose a hierachical graph drawing algorithm based on the KK algorithm, where we apply the FR method as layout initialization and then apply a novel discretization method to achieve grid layout.

3. Our Method: A System Overview

3.1. Graph Construction from Point Clouds

In the literature a graph from a point cloud is usually generated by connecting the K nearest neighbours (KNN) of each point. However, such KNN approaches suffer from selecting a suitable K. When K is too small, the points are intended to form small subgraphs (*i.e.,* clusters) with no guarantee of connectivity among the subgraphs. When K is too large, points are densely connected, leading to much more noise in local feature extraction.

In contrast, in this work we employ the Delaunary triangulation [9], a widely-used triangulation method in computational geometry, to create graphs based on the positions of points. The triangulation graph has three advantages: (1) The connection of all the nodes in the graph is guaranteed; (2) All the local nodes are directly connected; (3) The total number of graph connections is relatively small. In our experiments we found that the Delaunary triangulation gives us slightly better segmentation performance than the best one using KNN ($K = 20$) with margin of about 0.7%.

The worst-case computational complexity of the Delaunay triangulation is $O(n^{\lceil \frac{d}{2} \rceil})$ [1] where d is the feature dimension and $\lceil \cdot \rceil$ denotes the ceiling operation. Thus in the 3D space the complexity is $O(n^2)$ with $d = 3$. In our experiments we found that its running time on 2048 points is about 0.1s (CPU:Intel Xeon E5-2687W@3.1GHz), on average.

3.2. Graph Drawing: from Graphs to Images

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph with a vertex set $\mathcal{V}$ and an edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. $s_{ij} \geq 1, \forall i \neq j$ is the graph-theoretic distance such as shortest-path between two vertices $v_i, v_j \in \mathcal{V}$ on the graph that encodes the graph topology.

Now we would like to learn a function $f : \mathcal{V} \to \mathbb{Z}^2$ to map the graph vertex set to a set of 2D integer coordinates on the grid so that the graph topology can be preserved as

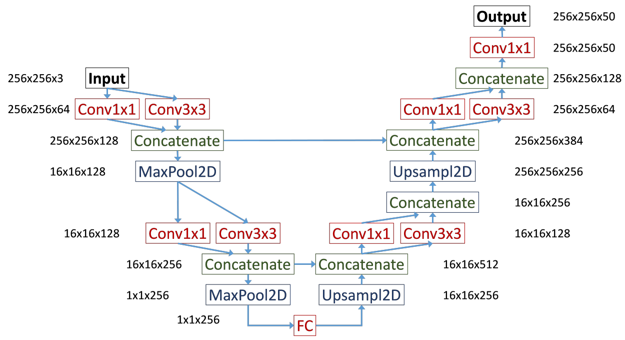

Figure 2: Illustration of our multi-scale U-Net architecture.

much as possible given a metric $d : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ and a loss $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. As a result, we are seeking for f to minimize the objective $\min_f \sum_{i \neq j} \ell(d(f(v_i), f(v_j)), s_{ij})$. Letting $\mathbf{x}_i = f(v_i) \in \mathbb{Z}^2$ as reparametrization, we can rewrite this objective as an *integer programming* (IP) problem $\min_{\mathcal{X} \subseteq \mathbb{Z}^2} \sum_{i \neq j} \ell(d(\mathbf{x}_i, \mathbf{x}_j), s_{ij})$, where the set $\mathcal{X} = \{\mathbf{x}_i\}$ denotes the *2D grid layout* of the graph, *i.e.,* all the vertex coordinates on the 2D grid.

For simplicity we set ℓ and d above to the least-square loss and Euclidean distance to preserve topology, respectively. This leads us to the following objective for learning:

$$\min_{\mathcal{X} \subseteq \mathbb{Z}^2} \sum_{i \neq j} \frac{1}{2} \left(\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{s_{ij}} - 1 \right)^2, \text{ s.t. } \mathbf{x}_i \neq \mathbf{x}_j, \forall i \neq j. \quad (1)$$

In fact the KK algorithm shares the same objective as in Eq. 1, but with different feasible solution space in $\mathbb{R}^2$, leading to relatively faster solutions that are used as the initialization in our algorithm later (see Alg. 2). Once the location of a point on the grid is determined, we associate its 3D feature as well as label, if available, with the location, finally leading to the (x, y, z)-image representation and a label mask with the same image size for network training.

In general an IP problem is NP-complete [63] and thus finding exact solutions is challenging. Relaxation and rounding is a widely used heuristic for solving IP due to its efficiency [3], where rounding is applied to the solution from the relaxed problem as the solution for the IP problem. However, considering that the computational complexity of the KK algorithm is at least $O(n^2)$ [24] with the number of nodes n in a graph (*i.e.,* thousands of points in a cloud for our case), it would be still too expensive to compute even if we relax the IP with rounding. Empirically we found that the running time of the KK algorithm on 2048 points is about 38s (CPU:Intel Xeon E5-2687W@3.1GHz), on average, which is considerably long. To accelerate the computation in practice, we propose a novel hierarchical solution in Sec. 4.

3.3. Multi-Scale U-Net for Point Segmentation

Eq. 1 enforces our image representations for the point clouds to be compact, indicating that the local structures in a

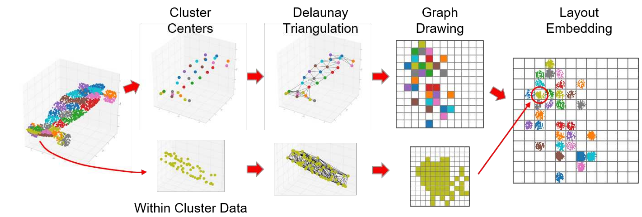

Figure 3: Illustration of hierarchical approximation for a point cloud. Each color represents a cluster where all the points share the same color.

point cloud are very likely to be preserved as local patches in its image representation. This is crucial for 2D CNNs to work because as such small convolutional kernels (*e.g.,* 3×3) can be used for local feature extraction.

To capture these local patterns in images, multi-scale convolutions are often used in networks such as the inception module in GoogLeNet [55]. U-Net [48] was proposed for biomedical image segmentation, and its variants are widely used for different image segmentation tasks. As illustrated in Fig. 2, in this paper we propose a multi-scale U-Net that integrates the inception module with U-Net, where FC stands for the fully connected layer, ReLU activation is applied after each Inception module and FC layer, and the softmax activation is applied after the last Conv1×1 layer.

Single-Scale *vs.* Multi-Scale. We only consider two sizes of 2D convolution kernels, *i.e.,* 1×1 and 3×3, because in our experiments we found that larger sizes of kernels do not bring significant improvement but heavier computational burden. We also compare the performance using single *vs.* multiple scales in Table 2. As we see the multi-scale U-Net with the inception module significantly outperforms the other single scale U-Nets.

Table 2: Performance comparison on ShapeNet using different U-Nets.

Scales in U-Net	1x1	3x3	Inception
Instance mIoU (%)	83.1	82.5	88.8

U-Net *vs.* CNNs. We also compare our U-Net with some other CNN architectures in Table 3. A baseline is an autoencoder-decoder network with similar architecture in Fig. 2 but no multi-scales and skip connections. We test it with 1×1 and 3×3 kernels, respectively, as shown in Table 3. A second baseline is SegNet [2], a much more complicated autoencoder-decoder. Again our U-Net works the best. By comparing Table 3 and Table 2, we can see that the skip connections in U-Net really help improve the performance. Note that our simple baselines can achieve comparable performance with the literature already.

Table 3: Instance mIoU comparison on ShapeNet using different CNNs.

CNNs	Conv1x1	Conv3x3	SegNet [2]	U-Net
mIoU (%)	81.6	78.1	86.9	88.8

All the comparisons above are based on the same image representations under the same protocols. Please refer to our experimental section for more details.

Algorithm 1 Balanced KMeans for Clustering

Input : point cloud $\mathcal{P} = \{\mathbf{p}\}$, number of clusters K, parameter α, distance metric s, cluster center computing function c

Output : balanced point clusters $\mathcal{H}$

$\mathcal{H} \leftarrow \text{KMeans}(\mathcal{P}, K)$;

while $\exists h^* \in \mathcal{H}, |h^*| > \alpha \frac{|\mathcal{P}|}{K}$ **do**

  $h' \in \arg\min_{\tilde{h}:|\tilde{h}|<\frac{|\mathcal{P}|}{K}} \left\{ s(c(h^*), c(\tilde{h})) \right\}$;

  $\mathbf{p}' \in \arg\min_{\mathbf{p}\in h^*} \{ s(\mathbf{p}, c(h')) \}$;

  $h^* \leftarrow h^* \setminus \{\mathbf{p}'\}$; $h' \leftarrow h' \cup \{\mathbf{p}'\}$;

end

Algorithm 2 Fast Graph-to-Image Drawing Algorithm

Input : Graph $\mathcal{G}$, 2D grid $\mathcal{S} \subseteq \mathbb{Z}^2$

Output : Graph layout $\mathcal{X} \subseteq \mathbb{Z}^2$

$\mathcal{X} \leftarrow \text{KK\_2D\_layout}(\mathcal{G})$; $\mathbf{a} \leftarrow \text{mean}(\mathcal{X})$; $\mathbf{b} \leftarrow \text{std}(\mathcal{X})$;

foreach $\mathbf{x} \in \mathcal{X}$ **do** $\mathbf{x} \leftarrow \text{round}((\mathbf{x} - \mathbf{a})./\mathbf{b} * \sqrt{|\mathcal{X}|})$;

while $\exists \mathbf{x}_i = \mathbf{x}_j, i \neq j, \mathbf{x}_i \in \mathcal{X}, \mathbf{x}_j \in \mathcal{X}$ **do**

  $\mathbf{x}^* \in \arg\min_{\mathbf{x}\in\mathcal{S}\setminus\mathcal{X}} \|\mathbf{x}_i - \mathbf{x}\|$; $\mathbf{x}_i \leftarrow \mathbf{x}^*$;

end

return $\mathcal{X}$;

4. Efficient Hierarchical Approximation

4.1. Two-Level Graph Drawing

For simplicity, in this section we will use the example in Fig. 3 to explain the key components in our hierarchical approximation. All the operations here can be easily extended to hierarchical cases with no change.

Given a point cloud, we first cluster these points hierarchically. We then apply the Delaunay triangulation and our graph drawing algorithms sequentially to the cluster centers as well as the within-cluster points per cluster, respectively, producing higher and lower-level graph layouts. Finally we embed all the lower-level graph layouts into the higher-level layout (recursively along the hierarchy) to produce the 2D image representation. For instance, we cluster a 2048-point cloud from ShapeNet into 32 clusters, and build a higher-level grid with size 16×16 using these 32 cluster centers. Within each cluster we build a lower-level grid with size 16×16 as well using the points belonging to the cluster. We finally construct the image representation for the cloud with size 256×256.

4.1.1 Balanced KMeans for Clustering

The key to accelerate computation in graph construction from point clouds is to reduce the number of points that the triangulation and graph drawing algorithms process at a time. Therefore, without loss of information we introduce hierarchical clustering, following the strategy in [12, 40, 19].

Recall that the complexity of the Delaunay triangulation and KK algorithms is $O(n^2)$, roughly speaking. Now consider the problem where given n points how we should determine K clusters so that the complexity in our graph construction from point clouds is minimize. The solution of this problem is that, ideally, all the clusters should have equal size of $\frac{n}{K}$, *i.e.,* balancing. Some algorithms such as normalized cut [51] are developed for learning balanced clusters, however, suffering from high complexity. Fast algorithms such as KMeans, unfortunately, do not provide such balanced clusters by nature.

We thus propose a heuristic post-processing step on top of KMeans to approximately balance the clusters with condition $|h| \leq \alpha\frac{|\mathcal{P}|}{K}, \forall h \in \mathcal{H}$ where $\mathcal{P} = \{\mathbf{p}\}$ denotes a point cloud with size $|\mathcal{P}|$, $\mathcal{H} = \{h\}$ denotes a set of clusters (*i.e.,* point sets) with size K, $|h|$ denotes the size of cluster h, and $\alpha \geq 1$ is a predefined constant. We list our algorithm in Alg. 1.

We first apply Kmeans to generate the cluster initials. We then target on one of the oversized clusters, h^*, at each iteration and change the cluster association for only one point. We determine the target cluster h' as the closest not-full cluster to $h*$ to receive a point. To send a point from h^* to h', the selected point is a boundary point that is closest to the center of h'. By default we set $\alpha = 1.2$, although we observed that higher values has little impact on either running time or performance.

4.1.2 Fast Graph-to-Image Drawing Algorithm

Recall that our graph drawing algorithm in Eq. 1 is an IP problem with complexity of NP-complete. Even though we use hierarchical clustering to reduce the number of points for processing, solving the exact problem is still challenging. To overcome this problem, we propose a fast approximate algorithm in Alg. 2, where $|\mathcal{X}|$ denotes the number of points.

Layout Discretization. After the layout initialization with the KK algorithm, we discrete the layouts onto the 2D grid. We first normalize the layout to a Gaussian distribution with a zero mean and an identity standard deviation (std). Then we rescale each 2D point in the layout with a scaling factor $\sqrt{|\mathcal{X}|}$, followed by a rounding operator. The intuition behind this is to organize the layout within a $\sqrt{|\mathcal{X}|} \times \sqrt{|\mathcal{X}|}$ patch as tightly as possible while minimizing the topological change. We finally replace each collided point with its nearest empty cell on the grid sequentially as our final graph layout.

Point Collision. In order to control the running time and image size in practice, we make a trade-off to predefine the maximum number of iterations as well as the maximum size of the 2D grid in Alg. 2. This may incur that some 3D points will collide at the same location on the grid. Such point collision scenarios, however, are very rare in our experiments. For instance, using our implementation for ShapeNet we observe 26 collisions with $2 \times 26 = 52$ points (*i.e.,* 2 points per collision) among 5,885,952 points in the testing set when projected onto 2D grid, leading to a 8.8×10^{-6} point collision ratio.

Once point collision occurs, we randomly select a point

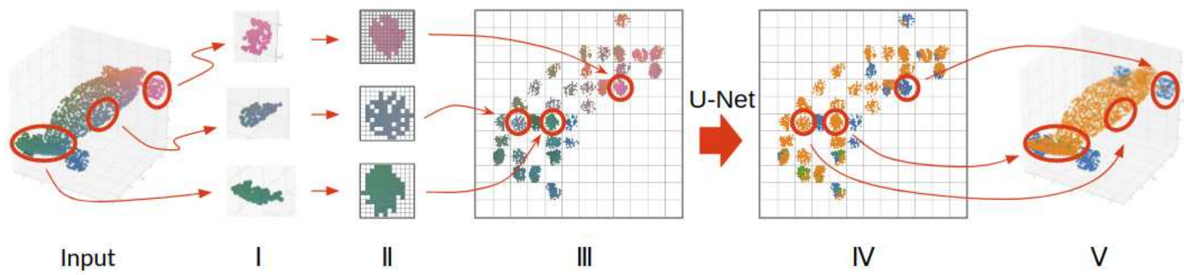

Figure 4: Illustration of our pipeline for point cloud semantic segmentation. Input: point cloud of a skateboard from ShapeNet. (I): point cloud clustering, (II): within-cluster image representation from graph drawing, (III): image embedding to generate a representation for the cloud, (IV): image segmentation using U-Net, (V): prediction reversion from the image representation to the point cloud. Here colors indicate either (x, y, z) features or the predicted labels.

from the collided points and put the selected point at the location with its 3D feature (x, y, z) and label, if available, for training U-Net. We observe that max pooling or average pooling is not appropriate to be applied here, because the labels of collided points can vary, *e.g.,* points at the boundary of different parts, leading to confusion for training U-Net.

At test time, we propagate the predicted label of the selected point to all its collided points. We observe only 4 out of 52 points mislabelled on ShapeNet due to point collision.

4.2. Generalization

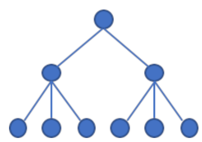

Figure 5: Full-tree illustration for our hierarchical clustering.

Recall that we would like to achieve balanced clusters in our hierarchical method for computational efficiency. Therefore, as generalization we propose using the full tree data structure, as illustrated in Fig. 5, to organize the hierarchical clusters, where at each cluster a higher-level graph is built using the Delaunay triangulation on the cluster centers, following by graph drawing to generate an image patch. Then we embed all the patches hierarchically to produce an image representation for a point cloud, and apply the remaining steps in Fig. 4 for segmentation.

Complexity. For simplicity and without loss of generality, assume that the full tree has $L \geq 1$ levels, and each cluster at the same level contains the same number of points. Let a_i, b_i be the numbers of clusters and sub-clusters per cluster at the i-th level, respectively, and n be the total number of points. For instance, in Fig. 5 we have $L = 3, a_1 = 1, b_1 = 2, a_2 = 2, b_2 = 3, a_3 = 6, b_3 = 1, n = 6$. Then it holds that $\prod_{j=i}^{L} b_i = \frac{n}{a_i}, \forall i$. We observe that in practice the running time of our hierarchical approximation is dominated by the KK initialization in Alg. 2 (see Table 4 for more details).

Proposition 1 (Complexity of Hierarchical Approximation)**.** *Given a full tree with $(a_i, b_i), \forall i \in [L]$ as above, the complexity of our hierarchical approximation is dominated by $O\left(n^{\frac{L+1}{L}}\right)$, at least.*

Proof. Here we focus on the complexity of the KK algorithm as it dominates the whole. Since for each cluster this complexity is $O(b_i^2)$, the total complexity of our approach is $O(\sum_{i=1}^{L} a_i b_i^2)$. Because

$$\sum_{i=1}^{L} a_i b_i^2 = n \sum_{i=1}^{L} \frac{b_i}{\prod_{j=i+1}^{L} b_j} \geq nL \left[\prod_i \left(\frac{b_i}{\prod_{j=i+1}^{L} b_j}\right)\right]^{\frac{1}{L}}$$
$$= nL \left(\frac{n}{\prod_{i=2}^{L} b_i^{i-1}}\right)^{\frac{1}{L}} = O\left(n^{\frac{L+1}{L}}\right), \quad (2)$$

we can complete the proof accordingly. □

5. Experiments

We evaluate our works on two benchmark data sets for point cloud segmentation: ShapeNet [69] and PartNet [42]. We follow exactly the same experimental setups as in PointNet [45] for ShapeNet and [42] for PartNet, respectively.

ShapeNet contains 16,881 CAD shape models (14,007 and 2,874 for training and testing, respectively) from 16 categories with 50 part categories. From each shape model 2048 points are scanned and labeled with their part categories. The shapes come from the same object category share the same part label sets, while shapes from different object categories have no shared part category. For performance evaluation there are two mean intersection-over-union (mIoU) metrics, namely, class mIoU and instance mIoU. Class mIoU is the average over points in each shape category, while instance mIoU is the average over all shape instances.

PartNet is a semantic segmentation benchmark focusing on fine-grained part-level 3D object understanding. Compared with ShapeNet, it has 24 shape categories and 26,671 shape instances. In addition, PartNet samples 10,000 points from each shape instance and defines up to 82 part semantics in one shape category, which calls for better local context learning to recognize them. Different from training a single network for all shape categories as done in ShapeNet, PartNet defines three segmentation levels in each shape category where a network is trained and tested for each category at each level separately.

5.1. Our Pipeline for Point Cloud Segmentation

In all of our experiments, we utilize the pipeline as illustrated in Fig. 4 for point cloud segmentation. As we expect,

Table 4: Running time of each component in our pipeline on ShapeNet. CPU: Intel Xeon E5-2687W@3.1GHz, GPU: NVidia RTX 2080Ti

Component	KMeans Clustering	Delaunay Tribulation	Graph Drawing	Network Inference	Total
Time (ms)	65.0±7.2	41.0±7.1	696.5±30.0	18.6±2.6	1054.8
Device	CPU	CPU	CPU	GPU	-

the 3D points are mapped to the 2D image space smoothly following their relative distances in the 3D space, leading to similar distributions in the neighborhood among image pixels to those in the local regions of the point cloud.

Implementation. We use the KMeans solver in the Scikit-learn library [4] with 100 iterations in maximum, the Delaunay triangulation implementation in the Scipy library [58], and the spring-layout implementation for graph drawing in the Networkx library [13]. In the mask image, we ignore the pixels with no point association that do not contribute to the loss of network training.

In our pipeline there are there hyper-parameters: the number of clusters K, the maximum ratio α, and the grid sizes for both lower and higher-level graph drawing. By default, we set $K = 32$ and $K = 100$ on ShapeNet and PartNet, respectively. On both data sets we use $\alpha = 1.2$, and set both lower and higher-level grid size to 16×16, leading to a 256×256 image representation per cloud.

We implement our multi-scale U-Net in Keras with Tensorflow backend on a desktop machine with an Intel Xeon E5-2687W@3.1GHz CPU and an NVidia RTX 2080Ti GPU. During training we follow PointNet [45] to rotate and jitter the shape models as input. We use the Adam [22] optimizer and set the learning rate to 0.0001. We train the network for 100 epochs with single batch in each iteration.

Running Time. We also list the average running time of each component in our pipeline in Table 4 for analysis. Compared with the running time on 2048 points, both Delaunay triangulation and graph drawing algorithms are accelerated significantly (recall 0.1s and 38s, respectively). Still graph drawing dominates the overall running time. Further acceleration will be considered in our future work.

5.2. State-of-the-art Performance Comparison

5.2.1 Ablation Study

In this section we evaluate the effects of different factors on our segmentation performance using ShapeNet. We keep using the default parameters and components in our pipeline, unless we explicitly mention what to change accordingly.

Graph Distance s_{ij} in Eq. 1. There are multiple choices to compute s_{ij} that our algorithm aims to preserve. We demonstrate three ways in Table 5 to verify their effects on performance. Note that for the 3D distance method, we do not construct graphs from point cloud. Rather we directly compute the (x, y, z)-distance between pairs of points. As we see, different s_{ij}'s do have impact on our segmentation performance, but relatively small. Compared with the results in Fig. 1, even using the 3D distance our pipeline can still outperform all the competitors.

Table 5: Instance mIoU results (%) under different settings for s_{ij}.

s_{ij}	Triangulation + Shortest Path [59]	KNN (K=20) + Shortest Path	3D Distance [56]
mIoU	88.8	87.1	86.4

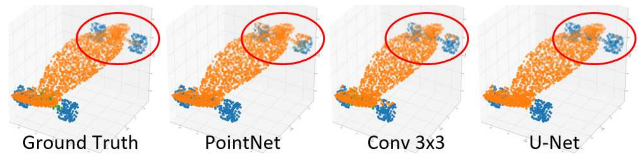

Figure 6: Visual comparison among different methods. **Ours: U-Net**.

Grid Size in Graph Drawing. Such grid size affects not only our segmentation performance but also the inference time of our pipeline. As demonstration we list three grid sizes in Table 6. As we expect, larger image sizes lead to significantly longer inference time, but marginal change in performance. Using smaller sizes it may be difficult to preserve the topological information among points, leading to performance degradation but faster inference time.

Table 6: Result comparison with different sizes of image representation.

Grid Size	10×10	16×16	24×24
Image Size	100×100	256×256	576×576
mIoU (%)	82.4	88.8	87.5
Time (ms)	13.6	18.6	63.3

Number of Clusters. Similar to grid size, the number of clusters also has an impact on both segmentation performance and inference time. To verify this, we show a result comparison in Table 7. With larger K, the performance decreases. This is probably because the higher-level graph loses more local context in the cloud so that even after pooling such loss cannot be recovered in learning. For timing, the numbers fluctuate due to different hierarchies as we prove in Prop. 1.

Table 7: Result comparison with different numbers of clusters, K, in KMeans.

K	32	64	128
mIoU (%)	88.8	86.7	85.9
Time (ms)	821.2	775.4	1164.5

5.2.2 Comparison Results

We first illustrate some visual results on ShapeNet in Fig. 6. Clear differences within the circulated regions can be observed and our result is much closer to the ground-truth.

We then list more detailed comparison on ShapeNet with some recent publications in 2018-2019 on ShapeNet in Table 8 that are also included in our summary in Fig. 1 before. Clearly our approach achieves the best and second best in 4 and 6 out of 16 categories, respectively. Our class-mIoU performance is on par with the state-of-the-art and instance-mIoU result improves the state-of-the-art by 1.4%.

Table 8: Result comparison (%) with recent works on ShapeNet. Numbers in red are the best in the column, and numbers in blue are the second best.

Method	class mIoU	instance mIoU	air plane	bag	cap	car	chair	ear phone	guitar	knife	lamp	laptop	motor bike	mug	pistol	rocket	skate board	table
DGCNN [62]	82.3	85.1	84.2	83.7	84.4	77.1	90.9	78.5	91.5	87.3	82.9	96.0	67.8	93.3	82.6	59.7	75.5	82.0
RS-CNN [34]	84.0	86.2	83.5	84.8	88.8	79.6	91.2	81.1	91.6	88.4	**86.0**	96.0	73.7	94.1	83.4	60.5	77.7	83.6
DensePoint [33]	84.2	86.4	84.0	85.4	**90.0**	79.2	91.1	**81.6**	91.5	87.5	84.7	95.9	74.3	94.6	82.9	64.6	76.8	83.7
SpiderCNN [67]	84.1	85.3	83.5	81.0	87.2	77.5	90.7	76.8	91.1	87.3	83.3	95.8	70.2	93.5	82.7	59.7	75.8	82.8
PointGrid [27]	82.2	86.4	85.7	82.5	81.8	77.9	**92.1**	**82.4**	**92.7**	85.8	84.2	95.3	65.2	93.4	81.7	56.9	73.5	84.6
VV-Net [39]	84.2	**87.4**	84.2	**90.2**	72.4	**83.9**	88.7	75.7	92.6	87.2	79.8	94.9	73.4	94.4	**86.4**	**65.2**	**87.2**	**90.4**
PartNet [71]	84.1	**87.4**	**87.8**	86.7	**89.7**	80.5	91.9	75.7	91.8	85.9	83.6	**97.0**	74.6	**97.3**	83.6	**64.6**	78.4	85.8
Ψ-CNN [28]	83.4	86.8	84.2	82.1	83.8	80.5	91.0	78.3	91.6	86.7	84.7	95.6	**74.8**	94.5	83.4	61.3	75.9	85.9
SFCNN [47]	82.7	85.4	83.0	83.4	87.0	80.2	90.1	75.9	91.1	86.2	84.2	**96.7**	69.5	94.8	82.5	59.9	75.1	82.9
PAN [43]	82.6	85.7	82.9	81.3	86.1	78.6	91.0	77.9	90.9	87.3	84.7	95.8	72.9	95.0	80.8	59.6	74.1	83.5
SRN [10]	82.2	85.3	82.4	79.8	88.1	77.9	90.7	69.6	90.9	86.3	84.0	95.4	72.2	94.9	81.3	62.1	75.9	83.2
PointCNN [30]	**84.6**	86.1	84.1	**86.5**	86.0	80.8	90.6	79.7	92.3	**88.4**	85.3	96.1	**77.2**	**95.3**	84.2	64.2	80.0	83.0
Ours	**84.6**	**88.8**	**86.5**	78.9	83.4	**80.9**	**92.6**	77.6	**93.3**	**91.6**	**90.0**	**96.7**	70.0	87.2	**84.5**	58.8	**83.0**	**88.1**

Table 9: Result comparison on PartNet using part-category mIoU (%). P, P^+, S and C refer to PointNet [45], PointNet++ [46], SpiderCNN [67] and PointCNN [30]. 1, 2 and 3 refer to three tasks: coarse-, middle- and fine-grained. Short lines denote the undefined levels. Numbers are cited from [42].

	Avg	Bag	Bed	Bott	Bowl	Chair	Clock	Dish	Disp	Door	Ear	Fauc	Hat	Key	Knife	Lamp	Lap	Micro	Mug	Frid	Scis	Stora	Table	Trash	Vase
P1	57.9	42.5	32.0	33.8	58.0	64.6	33.2	76.0	86.8	64.4	53.2	58.6	55.9	**65.6**	62.2	29.7	96.5	49.4	80.0	49.6	86.4	51.9	50.5	55.2	54.7
P2	37.3	—	20.1	—	—	38.2	—	55.6	—	38.3	—	—	—	—	—	27.0	—	41.7	—	35.5	—	44.6	34.3	—	—
P3	35.6	—	13.4	29.5	—	27.8	28.4	48.9	76.5	30.4	33.4	47.6	—	—	32.9	18.9	—	37.2	—	33.5	—	38.0	29.0	34.8	44.4
Avg	51.2	42.5	21.8	31.7	58.0	43.5	30.8	60.2	81.7	44.4	43.3	53.1	55.9	**65.6**	47.6	25.2	96.5	42.8	80.0	39.5	86.4	44.8	37.9	45.0	49.6
P^+1	**65.5**	**59.7**	51.8	53.2	67.3	68.0	**48.0**	80.6	89.7	59.3	**68.5**	64.7	62.4	62.2	**64.9**	**39.0**	**96.6**	55.7	83.9	51.8	**87.4**	58.0	**69.5**	64.3	**64.4**
P^+2	44.5	—	38.8	—	—	43.6	—	55.3	—	49.3	—	—	—	—	—	**32.6**	—	48.2	—	41.9	—	49.6	**41.1**	—	—
P^+3	42.5	—	30.3	41.4	—	39.2	**41.6**	**50.1**	80.7	32.6	38.4	**52.4**	—	—	**34.1**	**25.3**	—	48.5	—	36.4	—	40.5	**33.9**	46.7	49.8
Avg	58.1	**59.7**	40.3	47.3	67.3	50.3	**44.8**	**62.0**	85.2	47.1	53.5	58.6	62.4	62.2	**49.5**	**32.3**	**96.6**	50.8	83.9	43.4	**87.4**	49.4	**48.2**	55.5	57.1
S1	60.4	57.2	**55.5**	**54.5**	**70.6**	67.4	33.3	70.4	90.6	52.6	46.2	59.8	**63.9**	64.9	37.6	30.2	**97.0**	49.2	**83.6**	50.4	75.6	**61.9**	50.0	62.9	63.8
S2	41.7	—	**40.8**	—	—	39.6	—	**59.0**	—	48.1	—	—	—	—	—	24.9	—	47.6	—	34.8	—	46.0	34.5	—	—
S3	37.0	—	36.2	32.2	—	30.0	24.8	50.0	80.1	30.5	37.2	44.1	—	—	22.2	19.6	—	43.9	—	**39.1**	—	**44.6**	20.1	42.4	32.4
Avg	53.6	57.2	44.2	43.4	**70.6**	45.7	29.1	59.8	85.4	43.7	41.7	52.0	**63.9**	64.9	29.9	24.9	**97.0**	46.9	**83.6**	41.4	75.6	50.8	34.9	52.7	48.1
C1	64.3	**66.5**	**55.8**	49.7	61.7	**69.6**	42.7	**82.4**	**92.2**	**63.3**	64.1	**68.7**	**72.3**	**70.6**	62.6	21.3	**97.0**	**58.7**	**86.5**	**55.2**	**92.4**	61.4	17.3	**66.8**	63.4
C2	**46.5**	—	**42.6**	—	—	**47.4**	—	**65.1**	—	**49.4**	—	—	—	—	—	22.9	—	**62.2**	—	**42.6**	—	**57.2**	29.1	—	—
C3	**46.4**	—	**41.9**	**41.8**	—	**43.9**	36.3	**58.7**	**82.5**	**37.8**	**48.9**	**60.5**	—	—	**34.1**	20.1	—	**58.2**	—	**42.9**	—	**49.4**	21.3	**53.1**	**58.9**
Avg	**59.8**	**66.5**	**46.8**	**45.8**	61.7	**53.6**	39.5	**68.7**	**87.4**	**50.2**	**56.5**	**64.6**	**72.3**	**70.6**	48.4	21.4	**97.0**	**59.7**	**86.5**	**46.9**	**92.4**	**56.0**	22.6	**60.0**	**61.2**
Ours1	**72.3**	51.9	42.3	**58.9**	**90.7**	**77.8**	**72.0**	**89.9**	**92.5**	**82.3**	**85.6**	**77.8**	43.8	55.0	**64.3**	**51.3**	67.5	**95.7**	75.8	**97.6**	50.5	**70.1**	**88.6**	**65.2**	**88.4**
Ours2	**55.6**	—	37.7	—	—	**45.2**	—	51.1	—	**57.3**	—	—	—	—	—	**44.4**	—	**87.6**	—	**48.1**	—	**61.1**	**68.2**	—	—
Ours3	**52.3**	—	**38.6**	**57.1**	—	**43.2**	**57.8**	36.3	**93.0**	**68.5**	**42.9**	39.5	—	—	**61.3**	**33.1**	—	**83.4**	—	34.2	—	39.0	**40.5**	**59.1**	**62.1**
Avg	**63.0**	51.9	**39.5**	**58.0**	**90.7**	**55.4**	**64.9**	59.1	**92.8**	**69.4**	**64.3**	**58.7**	43.8	55.0	**62.8**	**42.9**	67.5	**88.9**	75.8	**60.0**	50.5	**56.7**	**65.8**	**62.2**	**75.3**

On PartNet our approach improves the state-of-the-art much more significantly, as listed in Table 9, with marginals of 6.8%, 9.1%, 5.9% on the three different segmentation levels, leading to an average improvement of 3.2%. Notice that PartNet is much more challenging given the number of categories as well as shapes. For instance, PointNet achieves 83.7% on ShapeNet but only 57.9% on PartNet. Our method, however, is much more robust and reliable with only 16.5 % decrease. Taking into account all the categories in the three levels, *i.e.,* 50 in total, we achieve the best in 31 out of 50.

6. Conclusion

In this paper we address the problem of point cloud semantic segmentation by taking advantage of conventional 2D CNNs. To this end, we propose a novel segmentation pipeline, including graph construction from point clouds, graph-to-image mapping using graph drawing, and point segmentation using U-Net. The computational bottleneck in our pipeline is the graph drawing algorithm, which is essentially an integer programming problem. To accelerate the computation, we further propose a novel hierarchical approximate algorithm with complexity dominated by $O(n^{\frac{L+1}{L}})$, leading to a save of about 97% running time. To better capture the local context embedded in our image presentations from point cloud, we also propose a multi-scale U-Net as our network. We evaluate our pipeline on ShapeNet and PartNet, achieving new state-of-the-art performance on both data sets with significantly large margins compared with the literature.

References

[1] Nina Amenta, Dominique Attali, and Olivier Devillers. Complexity of delaunay triangulation for points on lower-dimensional˜ polyhedra. In *18th ACM-SIAM Symposium on Discrete Algorithms*, pages 1106–1113, 2007. 2, 3

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017. 4

[3] Stephen P Bradley, Arnoldo C Hax, and Thomas L Magnanti. Applied mathematical programming. 1977. 4

[4] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013. 7

[5] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018. 3

[6] Luca Caltagirone, Samuel Scheidegger, Lennart Svensson, and Mattias Wahde. Fast lidar-based road detection using fully convolutional neural networks. In *2017 ieee intelligent vehicles symposium (iv)*, pages 1019–1024. IEEE, 2017. 3

[7] Marek Chrobak and Thomas H Payne. A linear-time algorithm for drawing a planar graph on a grid. *Information Processing Letters*, 54(4):241–246, 1995. 2

[8] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 31(5):833–852, 2018. 3

[9] Boris Delaunay et al. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934. 2, 3

[10] Yueqi Duan, Yu Zheng, Jiwen Lu, Jie Zhou, and Qi Tian. Structural relational reasoning of point clouds. In *CVPR*, June 2019. 1, 2, 3, 8

[11] Yaniv Frishman and Ayellet Tal. Multi-level graph layout on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1310–1319, 2007. 2

[12] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991. 2, 3, 5

[13] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008. 7

[14] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017. 3

[15] Zhizhong Han, Xiyang Wang, Yu-Shen Liu, and Matthias Zwicker. Multi-angle point cloud-vae: Unsupervised feature learning for 3d point clouds from multiple angles by joint self-reconstruction and half-to-half prediction. In *ICCV*, October 2019. 3

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 1

[17] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise convolutional neural networks. In *CVPR*, pages 984–993, 2018. 1, 2, 3

[18] Qiangui Huang, Weiyue Wang, and Ulrich Neumann. Recurrent slice networks for 3d segmentation of point clouds. In *CVPR*, pages 2626–2635, 2018. 1, 3

[19] Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann, and Mathieu Bastian. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PloS one*, 9(6):e98679, 2014. 2, 3, 5

[20] Li Jiang, Hengshuang Zhao, Shu Liu, Xiaoyong Shen, Chi-Wing Fu, and Jiaya Jia. Hierarchical point-edge interaction network for point cloud semantic segmentation. In *ICCV*, October 2019. 2, 3

[21] Tomihisa Kamada, Satoru Kawai, et al. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989. 2

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7

[23] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 863–872, 2017. 3

[24] Stephen G Kobourov. Spring embedders and force directed graph drawing algorithms. *arXiv preprint arXiv:1201.3011*, 2012. 2, 4

[25] Artem Komarichev, Zichun Zhong, and Jing Hua. A-cnn: Annularly convolutional neural networks on point clouds. In *CVPR*, June 2019. 1, 2, 3

[26] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *CVPR*, pages 4558–4567, 2018. 2, 3

[27] Truc Le and Ye Duan. Pointgrid: A deep network for 3d shape understanding. In *CVPR*, pages 9204–9214, 2018. 1, 2, 3, 8

[28] Huan Lei, Naveed Akhtar, and Ajmal Mian. Octree guided cnn with spherical kernels for 3d point clouds. In *CVPR*, June 2019. 1, 3, 8

[29] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-net: Self-organizing network for point cloud analysis. In *CVPR*, pages 9397–9406, 2018. 1, 2, 3

[30] 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. 1, 2, 3, 8

[31] Zongmin Li, Jun Zhang, Guanlin Li, Yujie Liu, and Siyuan Li. Graph attention neural networks for point cloud recognition. In *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pages 387–392. IEEE, 2019. 2, 3

[32] Zhidong Liang, Ming Yang, Liuyuan Deng, Chunxiang Wang, and Bing Wang. Hierarchical depthwise graph convolutional neural network for 3d semantic segmentation of point clouds. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8152–8158. IEEE, 2019. 2, 3

[33] Yongcheng Liu, Bin Fan, Gaofeng Meng, Jiwen Lu, Shiming Xiang, and Chunhong Pan. Densepoint: Learning densely contextual representation for efficient point cloud processing. In *ICCV*, October 2019. 1, 2, 3, 8

[34] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *CVPR*, June 2019. 1, 2, 3, 8

[35] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440, 2015. 1

[36] Yecheng Lyu, Lin Bai, and Xinming Huang. Chipnet: Real-time lidar processing for drivable region segmentation on an fpga. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(5):1769–1779, 2018. 1, 2, 3

[37] Yecheng Lyu, Lin Bai, and Xinming Huang. Real-time road segmentation using lidar data processing on an fpga. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018. 1, 2, 3

[38] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. Interpolated convolutional networks for 3d point cloud understanding. In *ICCV*, October 2019. 1, 2, 3

[39] Hsien-Yu Meng, Lin Gao, Yu-Kun Lai, and Dinesh Manocha. Vv-net: Voxel vae net with group convolutions for point cloud segmentation. In *ICCV*, October 2019. 1, 2, 3, 8

[40] Henning Meyerhenke, Martin Nöllenburg, and Christian Schulz. Drawing large graphs by multilevel maxent-stress optimization. In *International Symposium on Graph Drawing*, pages 30–43. Springer, 2015. 2, 3, 5

[41] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. Rangenet++: Fast and accurate lidar semantic segmentation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019. 1, 2, 3

[42] Kaichun Mo, Shilin Zhu, Angel X. Chang, Li Yi, Subarna Tripathi, Leonidas J. Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *CVPR*, June 2019. 6, 8

[43] Liang Pan, Pengfei Wang, and Chee-Meng Chew. Pointatrousnet: Point atrous convolution for point cloud analysis. *IEEE Robotics and Automation Letters*, 4(4):4035–4041, 2019. 2, 3, 8

[44] Quang-Hieu Pham, Thanh Nguyen, Binh-Son Hua, Gemma Roig, and Sai-Kit Yeung. Jsis3d: Joint semantic-instance segmentation of 3d point clouds with multi-task pointwise networks and multi-value conditional random fields. In *CVPR*, June 2019. 1, 2, 3

[45] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 652–660, 2017. 1, 2, 3, 6, 7, 8

[46] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017. 1, 2, 3, 8

[47] Yongming Rao, Jiwen Lu, and Jie Zhou. Spherical fractal convolutional neural networks for point cloud recognition. In *CVPR*, pages 452–460, 2019. 1, 2, 3, 8

[48] 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. Springer, 2015. 2, 4

[49] Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 138–148. Society for Industrial and Applied Mathematics, 1990. 2, 3

[50] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining point cloud local structures by kernel correlation and graph pooling. In *CVPR*, pages 4548–4557, 2018. 2, 3

[51] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Departmental Papers (CIS)*, page 107, 2000. 5

[52] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760. Springer, 2012. 2

[53] 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 *CVPR*, pages 2530–2539, 2018. 1, 2, 3

[54] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015. 1

[55] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015. 4

[56] Gusi Te, Wei Hu, Amin Zheng, and Zongming Guo. Rgcnn: Regularized graph cnn for point cloud segmentation. In *2018 ACM Multimedia Conference on Multimedia Conference*, pages 746–754. ACM, 2018. 2, 3, 7

[57] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, Francois Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *ICCV*, October 2019. 1, 2, 3

[58] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *arXiv e-prints*, page arXiv:1907.10121, Jul 2019. 7

[59] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. Graph attention convolution for point cloud semantic segmentation. In *CVPR*, June 2019. 2, 3, 7

[60] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):72, 2017. 1, 2, 3

[61] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation. In *CVPR*, pages 2569–2578, 2018. 1, 2, 3

[62] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):146, 2019. 2, 3, 8

[63] Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*. John Wiley & Sons, 2014. 2, 4

[64] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. *ICRA*, 2018. 1, 2, 3

[65] Bichen Wu, Xuanyu Zhou, Sicheng Zhao, Xiangyu Yue, and Kurt Keutzer. Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. In *ICRA*, 2019. 1, 2, 3

[66] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, June 2019. 1, 2, 3

[67] 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 (ECCV)*, pages 87–102, 2018. 2, 3, 8

[68] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *CVPR*, pages 206–215, 2018. 2

[69] Li Yi, Vladimir G Kim, Duygu Ceylan, I Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, Leonidas Guibas, et al. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (TOG)*, 35(6):210, 2016. 2, 6

[70] Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *CVPR*, pages 2282–2290, 2017. 1, 2, 3

[71] Fenggen Yu, Kun Liu, Yan Zhang, Chenyang Zhu, and Kai Xu. Partnet: A recursive part decomposition network for fine-grained and hierarchical shape segmentation. In *CVPR*, June 2019. 1, 2, 3, 8

[72] Zhiyuan Zhang, Binh-Son Hua, and Sai-Kit Yeung. Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics. In *ICCV*, October 2019. 1, 2, 3

[73] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. In *CVPR*, June 2019. 1, 2, 3