MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

Graph Spectral Point Cloud Processing

Hu, Wei; Chen, Siheng; Tian, Dong

TR2021-085 July 23, 2021

Abstract

In this book chapter, we present how graphs and graph signal processing contribute from low-level to high-level point cloud processing and understanding. The processing of point clouds addresses denoising and enhancement, as well as resampling techniques. The analysis of point clouds considers segmentation, classification, and recognition (e.g., skeleton-based action recognition). These are just some of the main areas of work related to GSP-based point cloud processing, which are unique relative to the use of GSP techniques for other data in terms of the signal model and processing methods. In practice, GSP has already demonstrated great benefits for processing point clouds and the main purpose of this chapter is to bring the underlying concepts together within a common framework.

ISTE Ltd

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Chapter 8

Graph Spectral Point Cloud Processing

Wei HU, Siheng CHEN, Dong TIAN

Peking University, Mitsubishi Electric Research Labs, InterDigital

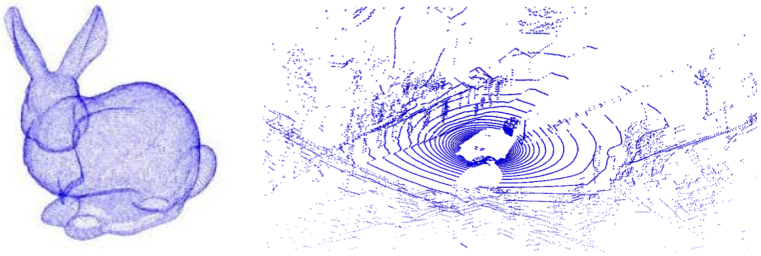
8.1. Introduction

Geometric aspects of the physical world are captured, digitized, and then reorganized as different forms of geometric data, among which a representative form is 3D point cloud Rusu and Cousins (2011). A point cloud is a set of 3D points sampled from the surfaces of objects or scenes. Each point usually contains geometry information (*i.e.*, 3D coordinates) and other attribute information, such as RGB colors and reflection intensity. The maturity of depth sensing and laser scanning techniques[1] makes the acquisition of 3D point clouds much convenient. Point clouds are gradually becoming a standardized solution for 3D representation and playing a critical role in a wide range of applications, including immersive tele-presence Mekuria *et al*. (2016), autonomous driving Chen *et al*. (2020), and heritage reconstruction Gomes *et al*. (2014).

Different from many other representation formats, point clouds record the precise geometry of objects or scenes. Conventional images could have geometric information embedded implicitly (*e.g.,* stereo images). On the contrary, point clouds, as a collection of 3D sampling points, form a more explicit representation of geometric information on the 3D world. The collected 3D points are *irregularly*

1. Commercial products include Microsoft Kinect, Velodyne LiDAR, Intel RealSense, *etc.*

scattered in the 3D space. Therefore, many traditional lattice-based methods cannot be directly applied to 3D points, while graphs provide a *natural* representation of point clouds: we can define each 3D point as a node, represent the signal (*e.g.*, coordinates, attributes) of each point as graph signal, and capture the relationship between a pair of 3D points by graph edges[2]. Hence, compared to the popularity of GSP in image applications, graph and GSP provide a more unique and prevailing contributor for point cloud processing tasks.



(a) The "Stanford Bunny" . (b) LiDAR sweep.

Figure 8.1. *Examples of 3D point clouds. The 3D point cloud in Plot (a) shows a 3D bunny model obtained by range scanners with some postprocessing. The 3D point cloud in Plot (b) shows one LiDAR sweep directly collected by Velodyne HDL-64E for autonomous driving.*

In this chapter, we present how graphs and GSP contribute from low-level to high-level point cloud processing and understanding. The processing of point clouds addresses denoising and enhancement, as well as resampling techniques. The analysis of point clouds considers segmentation, classification, and recognition (*e.g.*, skeleton-based action recognition). These are just some of the main areas of work related to GSP-based point cloud processing, which are unique relative to the use of GSP techniques for other data in terms of the signal model and processing methods. In practice, GSP has already demonstrated great benefits for processing point clouds and the main purpose of this chapter is to bring the underlying concepts together within a common framework.

The outline of this chapter is as follows. We start from the definition of graphs and graph-signals in point cloud applications. Then we introduce methodologies for point cloud processing, which include model-based GSP methods and data-driven geometric deep learning methods. Both families of approaches can be classified into spectral-domain methods and nodal-domain methods, depending on the domain

2. For example, we can connect two 3D points when their Euclidean distance is smaller than some threshold.

where the processing tools are designed. Finally, we discuss various applications of point cloud processing, including point cloud restoration, resampling, segmentation, classification and generation.

8.2. Graph and graph-signals in point cloud processing

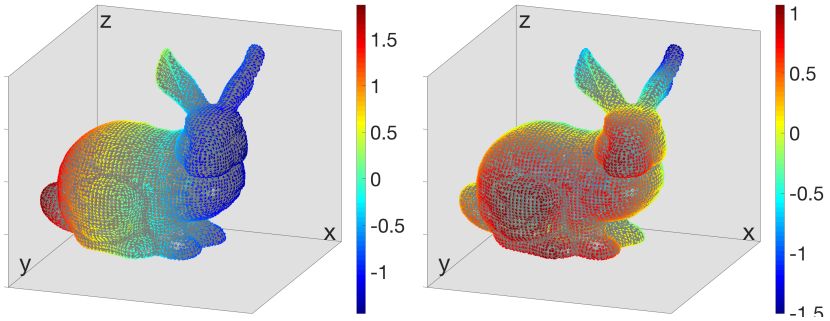
Point clouds describe the geometry of objects or scenes with a set of irregularly sampled 3D points. To denote a 3D point cloud, we consider a set, which ignores any order of 3D points. Let $\S = \{(\mathbf{p}_i, \mathbf{a}_i)\}_{i=1}^{N}$ be a set of N 3D points, whose ith element $\mathbf{p}_i = [x_i, y_i, z_i] \in \mathbb{R}^3$ represents the 3D coordinate of the ith point and $\mathbf{a}_i$ represents other attributes of the ith point, such as RGB colors, reflection intensity and surface normals. To ease mathematical computation, we could also represent a 3D point cloud in a matrix,

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} \in \mathbb{R}^{N \times d}, \qquad [8.1]$$

whose ith row vector $\mathbf{x}_i^T = [\mathbf{p}_i^T \ \mathbf{a}_i^T] \in \mathbb{R}^{1 \times d}$ records the information about the ith point.

To capture the relationships among 3D points, one can introduce a graph topology, where each node is a 3D point and each edge reflects the pairwise relationship of 3D points. This graph topology can be viewed as discrete approximations of Riemannian manifolds that describe the continuous surface of an original object or a scene. One mathematical representation of a graph topology with N nodes is via an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, whose (i, j)th element indicates the pairwise relationship between the ith and the jth 3D points. As a 3D point, each node is associated with several values, such as the 3D coordinates or attributes. The collection of those values form signals supported on this graph topology $\mathbf{A}$—called *graph signals*. In other words, each column vector in $\mathbf{X}$ can be considered as a graph signal; see Figure 8.2.

There are two typical methods to construct a graph topology: 1) model-based graph construction, which builds graphs with models from domain knowledge; 2) learning-based graph construction, which infers/learns the underlying graph from geometric data. Model-based graph construction for point clouds often assumes edge weights are inversely proportional to the affinities in coordinates, such as a K-nearest-neighbor graph and an ϵ-graph. A K-nearest-neighbor graph is a graph in which two nodes are connected by an edge, when their Euclidean distance is among the K-th smallest Euclidean distances from one 3D point to all the other 3D points.



(a) The "x-coordinate" graph signal .

(b) The "y-coordinate" graph signal.

(c) The "z-coordinate" graph signal .

(d) Zoom-in plot of the ear.

Figure 8.2. *Graph and graph signals for 3D point clouds. A K-nearest-neighbor graph is constructed to capture the pairwise spatial relationships among 3D points. The values of graph signals are reflected via color.*

An ϵ-nearest-neighbor graph is a graph in which two nodes are connected by an edge, when their Euclidean distance is smaller than a given threshold ϵ. Both K-nearest-neighbor graphs and ϵ-graphs can be efficiently implemented by using efficient data structures, such as Octree Peng and Kuo (2005), Kammerl (n.d.), Hornung *et al*. (2013). In learning-based graph construction, the underlying graph topology is inferred or optimized from point clouds in terms of certain optimization criterion. For example, edge weights could be trainable in an end-to-end learning architecture Wang, Sun, Liu, Sarma, Bronstein and Solomon (2019*b*).

8.3. Graph Spectral Methodologies for Point Cloud Processing

In this section, we elaborate on graph spectral methods for point cloud processing. Based on how we infer graph filter parameters, we classify graph spectral methods into two classes: *model-based graph spectral methods* and *learning-based graph spectral methods*. Model-based methods are built upon a deep understanding and assumptions of the characteristics of point clouds, such as *sparsity* models in the spectral domain or *smoothness* models in the nodal domain. The learning-based paradigm (*i.e.*, geometric deep learning Bronstein *et al*. (2017)) learns filter parameters end-to-end in a data-driven fashion, such as the recently developed Graph Neural Networks (GNNs) that extend classical neural networks to graph data via graph convolution in the spectral domain or nodal domain.

8.3.1. *Model-based Graph Spectral Methods for Point Clouds*

Detailed description of graph signal processing has been introduced in Part 1. Here we revisit them in the context of point clouds and stress how we deploy model-based graph spectral methods to process point clouds.

Spectral-domain Graph Filtering for Point Clouds. As introduced in Chapter 1 and Chapter 2, graph spectral filtering is defined in the graph frequency domain such as the Graph Fourier Transform (GFT) domain. Given a point cloud $\mathbf{X}$, suppose we have constructed a graph over $\mathbf{X}$, whose connectivity is encoded in a graph Laplacian $\mathbf{L}$ computed from $\mathbf{A}$. Let us denote $\mathbf{U}$ as the eigenvector matrix of $\mathbf{L}$ and $\{\lambda_0 = 0 \leq \lambda_1 \leq ... \leq \lambda_{N-1}\}$ as the set of eigenvectors of $\mathbf{L}$. Assuming the graph frequency response of the graph spectral filtering is $\hat{h}(\lambda_k)$ $(k = 0, \ldots, N-1)$, then this filtering can be written as

$$\mathbf{Y} = \mathbf{U} \begin{bmatrix} \hat{h}(\lambda_0) & & \\ & \ddots & \\ & & \hat{h}(\lambda_{N-1}) \end{bmatrix} \mathbf{U}^\top \mathbf{X}. \qquad [8.2]$$

This graph spectral filtering essentially first converts the point cloud data into the GFT domain, performs filtering on each eigenvalue (*i.e.*, spectrum of the graph), and finally projects back to the spatial domain via the inverse GFT. The key is to specify N graph frequency responses in $\hat{h}(\lambda_k)$ tailored for each point cloud processing task. Next, we discuss *low-pass* graph spectral filtering and *high-pass* graph spectral filtering separately.

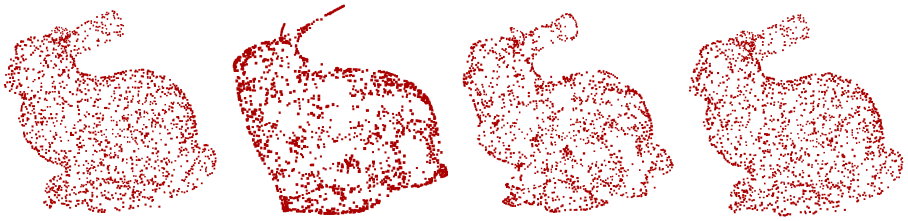
As in image processing, we can use a low-pass graph filter to capture rough shape of a point cloud and attenuate noise/outliers under the assumption that signals on point

clouds are smooth. In practice, a point cloud signal (*e.g.*, coordinates, normals, RGB colors) is naturally smooth with respect to the graph we construct from the signal. Thus, we can perform point cloud smoothing by a low-pass graph filter, where high-frequency components are likely to be generated by noise, as the energy of a smooth point cloud concentrates on low-frequency components.

One intuitive realization is an ideal low-pass graph filter, which completely eliminates all graph frequencies above a given bandwidth while keeping those below unchanged. The graph frequency response of an ideal low-pass graph filter with bandwidth b is

$$\hat{h}(\lambda_k) = \begin{cases} 1, k \leq b, \\ 0, k > b, \end{cases} \quad [8.3]$$

which projects the input point cloud onto a bandlimited subspace by removing eigenfunction components corresponding to large eigenvalues (*i.e.*, high-frequency components). The smoothed result provides a bandlimited approximation of the original point cloud. Fig. 8.3 demonstrates an example of the bandlimited approximation of the 3D coordinates of a sampled point cloud of *Bunny* with 10, 100 and 1000 graph frequencies respectively. We see that the first 10 low frequency components are able to represent the rough shape of *Bunny*, while the shape gets finer with more graph frequencies. See Rosman *et al*. (2013) and Chen *et al*. (2017) for more examples.



(a) Original. (b) 10 frequencies. (c) 100 frequencies. (d) 1000 frequencies.

Figure 8.3. *Low-pass approximation of the point cloud Bunny. Plot (a) is the original sampled point cloud with 1797 points. Plot (b), (c) and (d) show the low-pass approximations with 10, 100 and 1000 graph frequencies.*

Another simple choice is Haar-like low-pass graph filter as discussed in Chen *et al*. (2017), where the graph frequency response is

$$\hat{h}(\lambda_k) = 1 - \frac{\lambda_k}{\lambda_{max}}, \quad [8.4]$$

where $\lambda_{max} = \lambda_{N-1}$ is the maximum eigenvalue for normalization. As $\lambda_{k-1} \leq \lambda_k$, we have $\hat{h}(\lambda_{k-1}) \geq \hat{h}(\lambda_k)$. As such, low-frequency components are preserved while high-frequency ones are attenuated.

In contrary to the low-pass filtering, high-pass filtering eliminates low-frequency components, which enables the detection of large variations in a point cloud, such as geometric contours or texture variations. A simple design is a Haar-like high-pass graph filter with the following graph frequency response

$$\hat{h}(\lambda_k) = \frac{\lambda_k}{\lambda_{max}}. \qquad [8.5]$$

As $\lambda_{k-1} \leq \lambda_k$, we have $\hat{h}(\lambda_{k-1}) \leq \hat{h}(\lambda_k)$. This indicates that lower frequency responses are attenuated while high frequency reponse is preserved. An application example is contour detection as discussed in Chen *et al*. (2017).

In addition, we can design a desired spectral distribution and then use graph filter coefficients to fit this distribution. For example, an K-length graph filter is

$$\hat{h}(\Lambda) = \begin{bmatrix} \sum_{k=0}^{K-1} \hat{h}_k \lambda_1^k & \cdots & 0 \\ & & \ddots \\ 0 & \cdots & \sum_{k=0}^{K-1} \hat{h}_k \lambda_N^k \end{bmatrix},$$

where Λ is the eigenvalue matrix of the graph Laplacian $\mathbf{L}$, and $\hat{h}_k$ is the filter coefficients. If a desirable response of the ith graph frequency is c_i, we set

$$\hat{h}(\lambda_i) = \sum_{k=0}^{K-1} \hat{h}_k \lambda_i^k = c_i,$$

and solve a set of linear equations to obtain the graph filter coefficients $\hat{h}_k$. It is also possible to use the Chebyshev polynomial to construct graph filter coefficients Hammond *et al*. (2011*c*).

However, since spectral-domain graph filtering entails high computational complexity due to the full eigen-decomposition of the graph Laplacian, this class of methods are either dedicated to *small-scale* point clouds or applied to point clouds in a divide-and-conquer manner. For instance, one may divide a point cloud into regular cubes, and perform graph spectral filtering for each cube separately. Also, one may deploy a fast algorithm of GFT implementation (*e.g.*, fast GFT in Le Magoarou *et al*. (2017)) to accelerate the spectral filtering process.

Nodal-domain Graph Filtering for Point Clouds. This class of methods perform filtering on point clouds in the nodal domain, which is often computationally efficient and thus amenable to *large-scale* data. Intuitively, the functionality of nodal-domain graph filtering is to replace the feature value (attributes) at each 3D point with a weighted linear combination of feature values at its neighboring 3D points in the graph vertex domain.

The most elementary nontrivial graph filter is called a *graph shift operator*. A common option for a graph shift operator is the graph Laplacian matrix $\mathbf{L}$. Every linear, shift-invariant graph filter is a polynomial in the graph shift,

$$h(\mathbf{L}) = \sum_{k=0}^{K-1} h_k \mathbf{L}^k = h_0\mathbf{I} + h_1\mathbf{L} + \ldots + h_{K-1}\mathbf{L}^{K-1}, \quad [8.6]$$

where h_k, $k = 0, 1, \ldots, K-1$ are filter coefficients and K is the graph filter length. A higher order corresponds to a larger receptive field on the graph vertex domain. The output of graph filtering is given by the matrix-vector product. For example, let $\mathbf{X}_{\text{in}} \in \mathbb{R}^{N\times 3}$ be the RGB values of a 3D point cloud, the filtering response $\mathbf{X}_{\text{out}} = h(\mathbf{L})\mathbf{X}_{\text{in}} \in \mathbb{R}^{N\times 3}$ is the weighted averaged RGB values. The output of the ith point, $\sum_{k=0}^{K-1} h_k \left(\mathbf{L}^k\mathbf{X}\right)_i$ is a weighted average of the attributes of points that are within K hops away from the ith point. The kth graph filter coefficient, h_k, quantifies the contribution from the kth-hop neighbors. We design the filter coefficients to change the weights in local averaging.

Similarly to spectral-domain graph filtering, we now consider two typical types of nodal-domain graph filters.

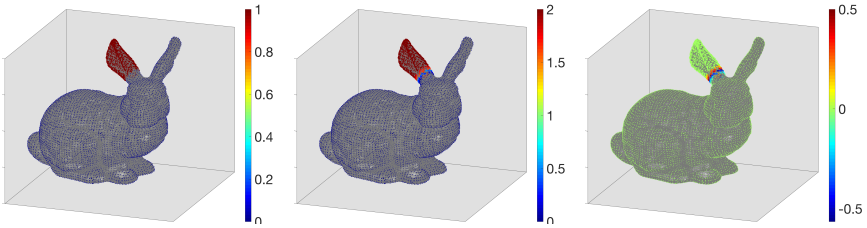
– Low-pass graph filtering. Similarly to classical signal processing, we can use a low-pass graph filter to smooth feature values over a 3D point cloud. Figure 8.4 shows the effect of low-pass graph filtering, where Plot (a) shows a graph signal activating the right ear of the bunny and Plot (b) shows that the boundary gets blurred after low-pass graph filtering. This indicates the functionality of low-pass graph filtering is to reduce transient change and capture the main shape.

– High-pass graph filtering. In image processing, a high-pass filter is used to extract edges and contours; similarly, we can use a high-pass graph filter to extract contours in a point cloud. Figure 8.4 (c) shows that the boundary gets highlighted after high-pass graph filtering. This indicates the functionality of high-pass graph filtering is to detect transient changes of feature values over a 3D point cloud.

Graph filtering can be used in various processing tasks. For example, Chen, Tian, Feng, Vetro and Kovavcević (2018*a*) designs a high-pass graph filter to extract contour-related features from a 3D point cloud with applications to 3D point cloud downsampling; Zeng, Cheung, Ng and J. Pang (2018) designs a graph filter to solve the graph-Laplacian-regularized optimization problem with applications to 3D point cloud denoising. Graph filtering also lays the mathematical foundation for graph convolutional neural networks, whose key operation is nothing but a graph filter.

8.3.2. *Learning-based Graph Spectral Methods for Point Clouds*

Learning-based graph spectral methods infer filter parameters end-to-end, such as the recently developed geometric deep learning (see Bronstein *et al*. (2017) and



(a) Original graph signal. (b) Low-pass graph filtering. (c) High-pass graph filtering.

Figure 8.4. *Graph filtering for 3D point clouds. Low-pass graph filtering smooths the sharp transition in a graph signal, while high-pass graph filtering highlights the sharp transition.*

references therein). While some previous works represent irregular point clouds with regular 3D voxel grids or collections of 2D images before feeding them to a neural network or impose certain symmetrizations in the net computation (*e.g.*, PointNet Qi, Su, Mo and Guibas (2017)), many recent efforts naturally represent point clouds on graphs and develop Graph Neural Networks (GNNs) that are generalizations of classical neural networks to graph data.

As the first such work on point clouds, Wang, Sun, Liu, Sarma, Bronstein and Solomon (2019*b*) introduces two useful techniques: the edge convolution operation and learnable graphs. The edge convolution is a convolution-like operation to extract geometric features on a graph. The edge convolution exploits local neighborhood information and can be stacked to learn global geometric properties. Let $\mathbf{H} \in \mathbb{R}^{N\times d}$ be a local-feature matrix, where the ith row $\mathbf{H}_i$ represents the features for the ith point. A basic computational block works as:

$$\mathbf{H}_i = \|_{(i,j)\in\mathcal{E}} g(\mathbf{X}_i, \mathbf{X}_j) \in \mathbb{R}^d, \qquad [8.7]$$

where $\mathcal{E}$ is the edge set and $g(\cdot,\cdot)$ is a generic mapping, implemented by some neural networks, and $\|$ is a generic aggregation function, which could be the summation or maximum operation. The edge convolution is also similar to nodal-domain graph filtering: both aggregate neighboring information; however, the edge convolution specifically models each pairwise relationship by a nonparametric function. Wang, Sun, Liu, Sarma, Bronstein and Solomon (2019*b*) also suggests to dynamically learn a kNN graph at each layer of the network. The distance metric of the learnable kNN graph is the Euclidean distance in the high-dimensional feature space, not the original 3D space.

Subsequent research has proposed novel graph neural networks for point cloud understanding, such as recognition, segmentation and classification. For example, Wang, Samari and Siddiqi (2018*a*) proposes local spectral graph

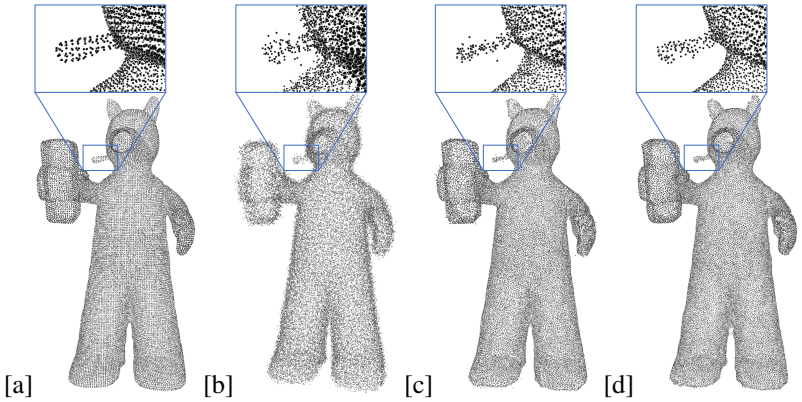


Figure 8.5. *A synthetic noisy point cloud with Gaussian noise $\sigma = 0.04$ for `Quasimoto` and denoised results: (a) The ground truth; (b) The noisy point cloud; (c) The denoised result by GLR Zeng, Cheung, Ng, Pang and Cheng (2019); (d) The denoised result by Hu* et al. *(2020).*

convolution, where filter coefficients represented in the graph spectral domain are learnable; Wang, Huang, Hou, Zhang and Shan (2019) proposes graph attention convolution, which selects the most relevant points from all the neighbors by trainable attentional weights. It also uses graph coarsening to promote multiscale analysis on graphs. As one of the most recent works in this area, Li, Müller, Thabet and Ghanem (2019) constructs the deepest yet graph convolution network (GCN) architecture, which has 56 layers. It transplants a series of techniques from CNNs, such as residual and dense connections, as well as dilated graph convolutions, to the graph domain. Some other works do not rely on graphs, but have based their computations on local neighbors defined by either kNN graphs Zhao *et al*. (2019), Liu *et al*. (2019*a*), Pan *et al*. (2019) or ϵ-graphs Thomas *et al*. (2019).

8.4. Low-level Point Cloud Processing

In this section, we discuss the applications of graph spectral methods in low-level point cloud processing, including restoration and resampling.

Similar to image restoration introduced in Chapter 7, point cloud restoration is an inverse problem to reconstruct point clouds from degraded versions. Due to various causes such as the inherent limitations of acquisition sensors and capturing viewpoints, point clouds often suffer from noise, holes, compression artifacts, *etc*. Consequently, many attempts have been made to point cloud restoration, with representative problems as point cloud denoising and point cloud inpainting.

8.4.1. *Application in point cloud denoising*

Point clouds are often perturbed by noise, which comes from hardware, software or environmental causes. Hardware wise, noise occurs due to the inherent limitations of the acquisition equipment. Software wise, in the case of generating point clouds with existing algorithms, points may locate somewhere completely wrong due to imprecise triangulation (*e.g.*, a false epipolar matching). Environmentally, outliers may appear due to the surrounding contamination, such as dust in the air. Noise corruption directly affects the subsequent applications of point clouds. We show a point cloud corrupted by Gaussian noise and its denoised results from two recent graph spectral methods in Fig. 8.5.

Assuming an additive noise model for a point cloud, namely

$$\mathbf{P} = \mathbf{Y} + \mathbf{E}, \qquad [8.8]$$

where $\mathbf{P} \in \mathbb{R}^{N\times 3}$ denotes the observed noise-corrupted 3D coordinates of the target point cloud with N points, $\mathbf{Y} \in \mathbb{R}^{N\times 3}$ is the ground truth 3D coordinates of the point cloud, and $\mathbf{E} \in \mathbb{R}^{N\times 3}$ is an additive noise. Then the problem of point cloud denoising is to reconstruct $\mathbf{Y}$ from $\mathbf{P}$.

Modeling of $\mathbf{E}$ depends on the actual point cloud acquisition mechanism. There exist a wide range of point cloud acquisition systems at different price points—from consumer-level depth sensors like Intel RealSense[3] costing 150 USD to high-end outdoor scanners like Teledyne Optech[4] that cost up to 250,000 USD—and defining accurate noise models for all of them is difficult. One may select the most common Gaussian noise model, which has been shown to be reasonably accurate for popular depth cameras like Microsoft Kinect Nguyen *et al*. (2012), Sun *et al*. (2008). Hence, $\mathbf{E}$ in [8.8] represents zero-mean *additive white Gaussian noise* (AWGN) with standard deviation σ, i.e.,

$$\mathbf{E} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}), \qquad [8.9]$$

where $\mathbf{I}$ is an identity matrix.

Standard experimental setup. There is no standard benchmark for point cloud denoising. Commonly adopted datasets include

– Synthetic noisy point cloud datasets created by adding Gaussian noise to public benchmark, such as five models from the surface reconstruction `benchmark` Berger *et al*. (2013), 100 models from the ShapeNetCore dataset Chang, Funkhouser, Guibas, Hanrahan, Huang, Li, Savarese, Savva, Song, Su *et al*. (2015), *etc.*

– Real-world noisy point cloud datasets, such as `Face` model Huang *et al*. (2009) and `Iron Vise` model Huang *et al*. (2013*b*), which are raw scans from laser scanners.

3. https://www.intelrealsense.com/

4. https://www.teledyneoptech.com/en/products/static-3d-survey/

Popular evaluation metrics include mean squared error (MSE) and signal-to-noise ratio (SNR). Th MSE between the ground truth point cloud $\S$ and the denoised one $\widehat{\S}$ is defined as:

$$\text{MSE}(\S, \widehat{\S}) = \frac{1}{N} \sum_{\mathbf{x}_i \in \S, \widehat{\mathbf{x}}_i \in \widehat{S}} \|\mathbf{x}_i - \widehat{\mathbf{x}}_i\|_2^2 . \quad [8.10]$$

The SNR between $\S$ and $\widehat{\S}$ follows as

$$\text{SNR}(\S, \widehat{\S}) = 20 \log\left(\frac{\sum_{\widehat{\mathbf{x}}_i \in \widehat{S}} \|\widehat{\mathbf{x}}_i\|_2^2}{\sum_{\mathbf{x}_i \in \S, \widehat{\mathbf{x}}_i \in \widehat{S}} \|\mathbf{x}_i - \widehat{\mathbf{x}}_i\|_2^2}\right). \quad [8.11]$$

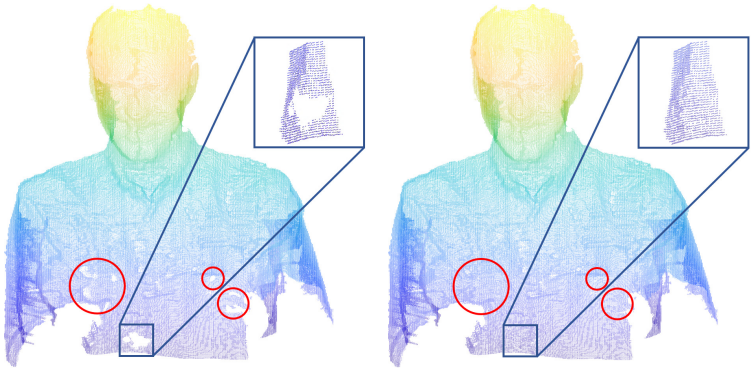
Standard methods. Existing point cloud denoising methods can be categorized into five main families: moving least squares (MLS)-based methods, locally optimal projection (LOP)-based methods, sparsity-based methods, non-local similarity-based methods, and graph-based methods. Please refer to Zeng, Cheung, Ng, Pang and Cheng (2019) for a thorough description of previous point cloud denoising approaches. Here we focus on the graph spectral methods to address the point cloud denoising problem.

Graph-based methods interpret a point cloud as a graph signal, and perform denoising via chosen spectral/spatial graph filters. Hard thresholding of GFT coefficients was proposed in Rosman *et al*. (2013) to remove spectrum components with small energy assuming they were introduced by noise. In Schoenenberger *et al*. (2015), the input point cloud was represented as a signal on a k-nearest-neighbor graph and then denoised via a convex optimization problem regularized by the gradient of the point cloud on the graph. In Dinesh *et al*. (2018), a reweighted graph Laplacian regularizer for surface normals was designed, with a general l_p-norm fidelity term that modeled two types of additive noise. Moreover, they established a linear relationship between normals and 3D point coordinates via bipartite graph approximation for ease of optimization.

Zeng, Cheung, Ng, Pang and Cheng (2019) proposed graph Laplacian regularization (GLR) of a low dimensional manifold model (LDMM), and sought self-similar patches to denoise them simultaneously. Instead of directly smoothing the coordinates or normals of 3D points, Duan *et al*. (2018) estimated a local tangent plane at each 3D point based on a graph and then reconstructed each 3D point by weighted averaging of its projections on multiple tangent planes. Instead of constructing the underlying graph with pre-defined edge weights from hand-crafted parameters, Hu *et al*. (2020) proposed feature graph learning by minimizing GLR

using the Mahalanobis distance metric matrix as variable, assuming feature vector per node is available. A fast algorithm was presented and applied to point cloud denoising, where the graph for each set of self-similar patches is computed from 3D coordinates and surface normals as features.

8.4.2. *Application in point cloud inpainting*



(a) Original point cloud. (b) Inpainted result by Hu *et al*. (2019).

Figure 8.6. *The point cloud Phili with real holes due to scanning and its inpainted result by Hu* et al. *(2019).*

Point clouds often exhibit several holes of missing data inevitably. Fig. 8.6 shows an example of real holes in the point cloud *Phili* and its inpainted result by Hu *et al*. (2019). This is mainly due to incomplete scanning views and inherent limitations of the acquisition equipments[5]. Besides, there may lack some regions in the data itself (*e.g.*, dilapidated heritage).

Let $\S = \{\mathbf{x}_i \in \mathbb{R}^d\}_{i=1}^N$ be a 3D point cloud with N 3D points and h be an inpainting operator that restores $\S$ to a complete point cloud $\widehat{\S}$ with M 3D points, where $M > N$. The inpaiting process works as

$$\widehat{\S} = h(\S),$$

where $\S \subset \widehat{\S}$. One often need to detect holes in $\S$ prior to inpainting the missing regions.

5. For example, 3D laser scanning is less sensitive to the objects in dark colors. This is because the darker it is, the more wavelengths of light the surface absorbs and the less light it reflects. Thus the laser scanning devices are unable to receive enough reflected light for dark objects to recognize.

Standard experimental setup. Similar to point cloud denoising, there is no standard benchmark for point cloud inpainting. Commonly adopted datasets include four MPEG models (*Longdress*, *Loot*, *Redandblack* and *Soldier*) from Eugene dâEon and Chou (2017), five MSR models (*Andrew9*, *David9*, *Phil9*, *Ricardo9* and *Sarah9*) from Loop *et al*. (2016), and the Stanford 3D Scanning Repository M. Levoy and Pull (https://graphics.stanford.edu/data/3Dscanrep/). There exist real holes in some of the datasets, such as the MPEG models and MSR models that are acquired from 3D sensors. One can further generate synthetic holes on point clouds for quantitative comparison with the ground truth. The evaluation metrics are the same as those for point cloud denoising.

Standard methods. Point clouds inpainting methods can be classified into two main categories according to the cause of holes: (1) restore holes in the object itself such as heritage and sculptures, and (2) inpaint holes caused by the limitation of scanning devices. For the first class of methods, the main hole-filling data source is online database, as the holes are often large. The other class of methods focus on holes generated due to the limitations of scanning devices. This kind of holes is smaller than the aforementioned ones in general, thus the information of the data itself is often enough for inpainting. Please refer to Hu *et al*. (2019) for detailed description, while we focus on graph spectral methods as follows.

Lozes *et al*. (2014) proposed Partial Differential Equation (PDE) based graph signal processing to solve an optimization equation for point cloud inpainting, which refers to the neighborhood of the hole to compute the geometric structure. A k-nearest-neighbor graph is constructed from the union of the input point cloud and generated points. Then a graph-based PDE solver is employed to deform the generated points to fit into the hole. This method is suitable for small holes or smooth surfaces, but it faces problems if the hole boundary contains folds or twist. In Chinthaka Dinesh (2018), hole filling was performed iteratively using templates near the hole boundary to find the best matching regions elsewhere in the cloud, from where existing points are transferred to the hole. An undirected k-nearest-neighbor graph was constructed on the input point cloud to define a smoothness term. Hu *et al*. (2019) proposed to exploit both the local smoothness and the non-local self-similarity in point clouds for inpainting. They first presented the smoothing and denoising properties of a graph-signal smoothness prior in order to describe the local smoothness of point clouds. Secondly, the characteristics of non-local self-similarity were explored, by globally searching for the most similar area to the missing region. The similarity metric between two areas was defined based on the direct component and the anisotropic graph total variation of normals in each area. Finally, the hole-filling step was formulated as an optimization problem based on the selected most similar area and regularized by the graph-signal smoothness prior. Besides, an efficient automatic hole detection approach was proposed for the point cloud prior to inpainting.

8.4.3. *Application in point cloud resampling*

The resampling task considers changing the point density of a 3D point cloud, which includes downsampling and upsampling.

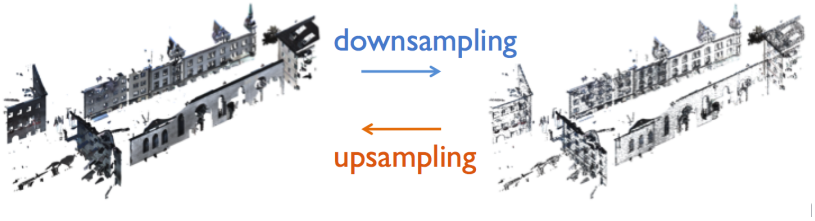


Figure 8.7. *Dual problems of 3D point cloud downsampling and upsampling.*

8.4.3.1. *Downsampling*

The goal of downsampling is to select a subset of 3D points in an original 3D point cloud while peserving representative information; see Figure 8.7. Handling a large number of 3D points is challenging and expensive. Therefore, a 3D point cloud is often sampled to a size that can be processed more easily. As a typical task of 3D point cloud processing, downsampling is potentially useful to data storage. To represent a 3D scene, one can select representative 3D points from an HD map through downsampling, leading to faster and better localization performances Chen, Tian, Feng, Vetro and Kovavcević (2018*a*).

Let $\S = \{\mathbf{x}_i \in \mathbb{R}^d\}_{i=1}^N$ be a 3D point cloud with N 3D points and h be a downsampling operator that selects M 3D points from $\S$, where $M < N$. The downsampling process works as

$$\S_{\mathbf{M}} = h(\S),$$

where $\S_{\mathbf{M}} = \{\mathbf{x}_{\mathbf{M}_i} \in \mathbb{R}^d\}_{i=1}^M$ is a downsampled 3D point cloud, where the downsampled set $\mathbf{M} = (\mathbf{M}_1, \ldots, \mathbf{M}_M)$ denotes the sequence of downsampled indices, $\mathbf{M}_i \in \{1, \ldots, N\}$ with $|\mathbf{M}| = M$.

Standard experimental setup. As a processing task, it is difficult to directly evaluate the performance of downsampling. Researchers usually input downsampled 3D point clouds to some subsequent tasks and test their performance. Chen, Tian, Feng, Vetro and Kovavcević (2018*a*) evaluates downsampling on 3D point cloud registration. The evaluation metric is the localization error, such as the mean square error; and Dovrat *et al*. (2018) suggests evaluating downsampling on the tasks of

classification and reconstruction. For classification, the dataset is ModelNet40 Wu, Song, Khosla, Yu, Zhang, Tang and Xiao (2015*a*) and the evaluation metric is classification accuracy. For reconstruction, the dataset is ShapeNet Chang, Funkhouser, Guibas, Hanrahan, Huang, Li, Savarese, Savva, Song, Su, Xiao, Yi and Yu (2015) and the evaluation metric is the reconstruction error. To evaluate the quality of reconstruction, two distance metrics are usually considered. The Earth mover's distance is the objective function of a transportation problem, which moves one point set to the other with the lowest cost; that is,

$$d_{\mathrm{EMD}}(\S, \widehat{\S}) = \min_{\phi:\S\to\widehat{\S}} \sum_{\mathbf{x}\in\S} \|\mathbf{x} - \phi(\mathbf{x})\|_2 \,, \qquad [8.12]$$

where ϕ is a bijection. The Chamfer distance measures the total distance between each point in one set to its nearest neighbor in the other set; that is,

$$d_{\mathrm{CH}}(\S, \widehat{\S}) = \frac{1}{N} \sum_{\mathbf{x}\in\S} \min_{\widehat{\mathbf{x}}\in\widehat{S}} \|\mathbf{x} - \widehat{\mathbf{x}}\|_2 + \frac{1}{M} \sum_{\widehat{\mathbf{x}}\in\widehat{S}} \min_{\mathbf{x}\in\S} \|\widehat{\mathbf{x}} - \mathbf{x}\|_2 \,.$$

Both the Earth mover's distance and the Chamfer distance enforce the underlying manifold of the reconstruction to stay close to that of the original point cloud. Reconstruction with the Earth mover's distance usually outperforms that with the Chamfer distance; however, it is more efficient to compute the Chamfer distance.

Standard methods. There are three common approaches: farthest point sampling, learning-based sampling and nonuniformly random sampling. A simple and popular downsampling technique is the farthest point sampling (FPS) Qi, Yi, Su and Guibas (2017*a*). It randomly chooses the first 3D point and then iteratively chooses the next 3D point that has the largest distance to all the points in the downsampled set. It is nothing but the deterministic version of K-means++ Arthur and Vassilvitskii (2007). Compared with uniformly random sampling, it has better coverage of the entire 3D point point given the same number of samples; however, FPS is agnostic to a subsequent application, such as localization and recognition.

S-NET Dovrat *et al*. (2018) is a deep-neural-network-based downsampling system. It takes a 3D point cloud and produces a downsampled 3D point cloud that is optimized for a subsequent task. The architecture is similar to latentGAN used for 3D point cloud reconstruction Achlioptas *et al*. (2018). The difference is that S-NET does not reconstruct all the 3D points, but only reconstruct a fixed number of 3D points. The loss function include a reconstruction loss, such as the Earth mover's distance [8.12] and Chamfer distance [8.13], and a task-specific loss, such as classification loss. Since the reconstructed 3D point cloud is not a subset of the original 3D point cloud any more, S-NET matches each reconstructed 3D point to its

nearest neighbor in the original 3D point cloud; however, it is not trivial to apply S-NET to train and operate on large-scale 3D point clouds, which makes it less practical in autonomous driving.

To make the downsampling process more efficient and adaptive to subsequent tasks, Chen, Tian, Feng, Vetro and Kovavcević (2018*a*) considers a randomized downsampling strategy by choosing downsampled indices from a nonuniform distribution. Let $\pi \in \mathbb{R}^N$ be a downsampling distribution, where π_i denotes the probability of selecting the ith sample in each random trial. Chen, Tian, Feng, Vetro and Kovavcević (2018*a*) designs an optimal downsampling distribution by solving a reconstruction-based optimization problem. It turns out that the optimal downsampling distribution is $\pi_i^* \propto \|\mathbf{H}_i\|_2$, where $\mathbf{H}_i \in \mathbb{R}^D$ is task-specific features of the ith point, which could be obtained by graph filtering.

8.4.3.2. *Upsampling*

The goal of upsampling is to generate a dense (high-resolution) 3D point cloud from a sparse (low-resolution) 3D point cloud to describe the underlying geometry of an object or a scene; see Figure 8.7. 3D point cloud upsampling is similar in nature to the super resolution of 2D images and is essentially an inverse procedure of downsampling.

Let $\S = \{\mathbf{x}_i \in \mathbb{R}^d\}_{i=1}^N$ be a 3D point cloud with N 3D points and h be a upsampling operator that generate N' 3D points from $\S$, where $N' > N$. The upsampling process works as

$$\widehat{\S} = h(\S),$$

where $\S \subset \widehat{\S}$. Intuitively, a 3D point cloud $\S$ is sampled from some surface and we aim to use the high-resolution 3D point cloud $\widehat{\S}$ to capture the same surface, but provide a higher density.

Standard experimental setup. There is no standard benchmark for 3D point cloud upsampling. Researchers create their own training and testing datasets based on the VisionAir repository *Visionair* (n.d.), ModelNet40 Wu, Song, Khosla, Yu, Zhang, Tang and Xiao (2015*a*), ShapeNet Chang, Funkhouser, Guibas, Hanrahan, Huang, Li, Savarese, Savva, Song, Su, Xiao, Yi and Yu (2015), or SHREC15 Lian *et al*. (2015). Some common evaluation metrics are the Earth mover's distance [8.12] and Chamfer distance [8.13].

Standard methods. Classical 3D point cloud upsampling algorithms are based on image super resolution algorithms. For example, Alexa *et al*. (2003) constructs surfaces with the moving least squares algorithm and generates new points at the vertices of the Voronoi diagram to upsample a 3D point cloud; to avoid over-smoothing, Huang *et al*. (2013*a*) applies an anisotropic locally optimal

projection operator to preserve sharp edges by pushing 3D points away from the edges, and achieves the edge-aware 3D point cloud upsampling; Wu, Huang, Gong, Zwicker and Cohen-Or (2015) combines the smoothness of surfaces and the sharpness of edges through an extracted meso-skeleton. The meso-skeleton consists of a mixture of skeletal curves and sheets to parameterize the underlying surfaces. It then generates new 3D points by jointly optimizing both the surface and 3D points residing on the meso-skeleton; however, these classical upsampling algorithms usually depend heavily on local geometry priors, such as the normal vectors and the curvatures. Some algorithms also suffer from multiscale structure preservation due to the assumption of global smoothness Wang, Wu, Huang, Cohen-Or and Sorkine-Hornung (2018).

With the development of deep neural networks, more upsampling algorithms adopt the learning-based approach. PU-Net Yu *et al*. (2018*b*) is the first end-to-end 3D point cloud upsampling network, which extracts multi-scale features based on PointNet++. The architecture is similar to latentGAN for 3D point cloud reconstruction, but reconstructs many more 3D points than the original 3D point cloud. The loss function includes a reconstruction loss and a repulsion loss, pushing a more uniform distribution for the generated points. Inspired by the recent success of neural-network-based image super-resolution, patch-based progressive Wang, Wu, Huang, Cohen-Or and Sorkine-Hornung (2018) proposes a patch-based progressive upsampling architecture for 3D point clouds. The multi-step upsampling strategy breaks an upsampling network into several subnetworks, where each subnetwork focuses on a specific level of details. To emphasize edge preservation, EC-Net designs a novel edge-aware loss function Yu *et al*. (2018*a*). During the reconstruction, EC-Net is able to attend to the sharp edges and provide more precise 3D reconstructions. Note that all those deep-neural-network-based methods are trained based on well-selected patches, which cover a rich variety of shapes.

8.5. High-level Point Cloud Understanding

In this section, we discuss the applications of graph spectral methods in high-level point cloud understanding, including segmentation, classification and generation.

8.5.1. *Application in point cloud segmentation*

The goal is to group points into subsets (referred as segments) with one or more characteristics in common (geometry, semantic, *etc.*), which exploits in-depth the informative value of point clouds. Depending on whether semantic labels are required as the output, point cloud segmentation includes part segmentation and semantic segmentation. Fig. 8.8 demonstrates part segmentation of point clouds by Gao *et al*. (2020). Due to the irregular structure and varying density of point clouds, point cloud segmentation is quite challenging and has attracted much attention recently.

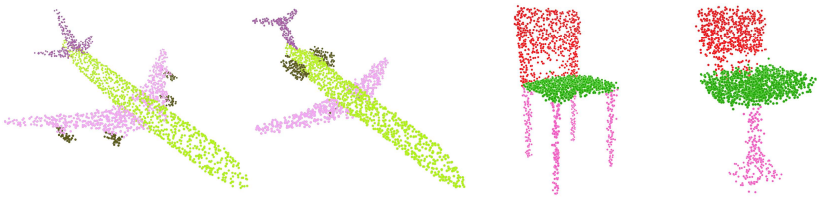


Figure 8.8. *Demonstration of part segmentation for point clouds by Gao* et al. *(2020).*

Let h be an operator that maps each point $\S_i$ in a point cloud $\S$ to a confidence vector $\mathbf{y}_i$ indicating the segment, that is,

$$\mathbf{y}_i = h(\S_i) \in [0,1]^C, \qquad [8.13]$$

where C is the number of segments. The cth element of $\mathbf{y}_i$, $\mathbf{y}_i^c$, indicates the likelihood of the ith point belonging to the cth segment of the 3D point cloud.

Standard experimental setup. Commonly employed benchmarks include ShapeNet part dataset Yi *et al*. (2016), Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS) , and ScanNet Dai, Chang, Savva, Halber, Funkhouser and Nießner (2017). ShapeNet part dataset contains 16881 shapes from 16 categories, annotated with 50 labels in total. Each 3D point cloud contains $2,048$ points, most of which are labeled with fewer than six parts. S3DIS dataset includes 3D scan point clouds for 6 indoor areas including 272 rooms in total. Each point belongs to one of 13 semantic categories, *e.g.*, board, bookcase, chair, ceiling, and beamâplus clutter. ScanNet is an instance-level indoor RGB-D dataset that includes both 2D and 3D data. Point cloud segmentation is usually evaluated by mean Intersection of Union (mIoU). IoU is widely used in semantic segmentation to measure the ratio of the ground truth and prediction, and mean IoU is the average of IoU for each label appearing in the model categories.

Standard methods. Previous point cloud segmentation works can be classified into model-driven segmentation and data-driven segmentation. Model-driven methods include edge-based Rabbani *et al*. (2006), region-growing Rusu *et al*. (2008) and model-fitting Tarsha-Kurdi *et al*. (2007), which are based on the prior knowledge of the geometry but sensitive to noise, uneven density and complicated structure. Data-driven segmentation, on the other hand, learns the semantics from data, such as deep learning methods. Qi, Su, Mo and Guibas (2017) came up with PointNet, a neural network which consumes point clouds directly. The key breakthrough is the proposed symmetric function that is invariant to the order of points. However, PointNet processes each point identically and independently. Hence, Qi, Yi, Su and Guibas (2017*b*) proposed PointNet++ to introduce hierarchical grouping, which achieves better performance. Please refer to Te *et al*. (2018) for detailed discussion.

Te *et al*. (2018) proposed a Regularized Graph Convolutional Neural Network (RGCNN), which is the first to design graph neural networks for point cloud segmentation. They regularize each layer of RGCNN by adding graph-signal smoothness prior in the loss function, and prove the spectral smoothing property of this prior. The graph Laplacian matrix is updated in each layer of RGCNN in order to adaptively capture the structure of dynamic graphs. Experiments show that RGCNN significantly reduces the computation complexity while achieving competitive results with the state of the art. It is also much more robust to both low density and noise in comparison with other methods. Wang, Sun, Liu, Sarma, Bronstein and Solomon (2019*c*) proposed a new neural network module dubbed EdgeConv suitable for CNN-based high-level tasks on point clouds including classification and segmentation. EdgeConv acts on graphs dynamically computed in each layer of the network. It is differentiable and can be plugged into existing architectures. Compared to existing modules operating in extrinsic space or treating each point independently, EdgeConv has several appealing properties: It incorporates local neighborhood information; it can be stacked applied to learn global shape properties; and in multi-layer systems affinity in feature space captures semantic characteristics over potentially long distances in the original embedding. Gao *et al*. (2020) proposed a novel unsupervised learning of Graph Transformation Equivariant Representations (GraphTER), aiming to capture intrinsic patterns of graph structure under both global and local transformations. The learned GraphTER is then applied to graphs of 3D point cloud data for segmentation.

8.5.2. *Application in point cloud classification*

We consider the classification task for two types of point clouds: static point clouds and dynamic point clouds.

8.5.2.1. *Static point cloud classification*

The goal is to classify a static 3D point cloud to a predefined category.

Let h be a classifier that maps a static point cloud $\S$ to a confidence vector $\mathbf{y}$ indicating the category belongings, that is,

$$\mathbf{y} = h(\S) \in [0,1]^C, \qquad [8.14]$$

where C is the number of classes. The cth element of $\mathbf{y}$, $\mathbf{y}_c$, indicates the likelihood of the static 3D point cloud belonging to the cth class.

Standard experimental setup. A widely employed benchmark is ModelNet40 Wu, Song, Khosla, Yu, Zhang, Tang and Xiao (2015*b*). This dataset contains $12,311$ meshed CAD models from 40 categories, where $9,843$ models are used for training and $2,468$ models are for testing. For each model, $1,024$ points are sampled from the original mesh.

Standard methods. Early attempts of point cloud classification adapted ideas from deep learning on images, *e.g.*, converting point clouds to regular 3D voxel grids Maturana and Scherer (2015) or multiple view images Su *et al*. (2015) before feeding them into typical convolutional neural networks (CNNs). However, this introduces quantization error in the conversion process and renders the resulting data unnecessarily voluminous. As in point cloud segmentation, Qi, Su, Mo and Guibas (2017) addressed this problem by learning global features of point clouds using a symmetric function that is order-invariant. Other non-graph-based methods include PointNet++ Qi, Yi, Su and Guibas (2017*b*), PointCNN Li, Bu, Sun, Wu, Di and Chen (2018), SpiderCNN Xu *et al*. (2018), RS-CNN Liu *et al*. (2019*b*), *etc*.

Along the thread of Graph Neural Networks, Wang, Samari and Siddiqi (2018*b*) proposed to carry out graph convolution on a nearest-neighbor graph constructed from a pointâs local neighborhood, combined with a novel graph pooling strategy. In FoldingNet Yang, Feng, Shen and Tian (2018), a novel end-to-end deep auto-encoder is proposed to address unsupervised feature learning for point cloud classification. The folding-based decoder deforms a canonical 2D grid onto the underlying 3D object surface of a point cloud, achieving low reconstruction errors even for objects with delicate structures. By extracting global features via pooling, previous approaches for point cloud segmentation can be extended to classification, such as DGCNN Wang, Sun, Liu, Sarma, Bronstein and Solomon (2019*c*), RGCNN Te *et al*. (2018) and GraphTER Gao *et al*. (2020).

non-graph point cloud classification: PointNet, PointNet++, FoldingNet, PointCNN, PCNN, VoxNet...

to add: Yue Wang's paper and Siheng's TIP paper...

8.5.2.2. *Dynamic point cloud classification*

to add: 3D object detection in dynamic point clouds? ...

The goal is to classify a dynamic 3D point cloud to a predefined category. A dynamic point cloud specifically means a temporal sequence of 3D point clouds. Here we motivate this setting through 3D skeleton-based human action recognition. Given dynamic 3D positions of human body joints, we aim to recognize the corresponding action; see Figure 8.9. Note that each 3D point cloud indicates the human pose at a time stamp, where each 3D point is the position of one body joint. Such a 3D point cloud is usually obtained through either depth sensors or pose estimation algorithms based on videos. The overall dynamic point cloud is the sequence of the human pose, reflecting the human action.

Let h be a classifier that maps a sequence of 3D point cloud $\{\S_t\}_{t=1}^T$ to a confidence vector $\mathbf{y}$ indicating the category belongings, that is,

$$\mathbf{y} = h(\{\S_t\}_{t=1}^T) \in [0,1]^C, \qquad [8.15]$$

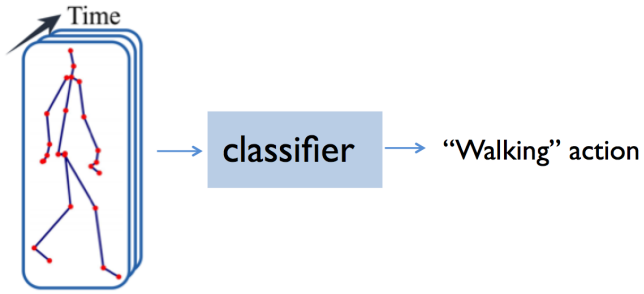


Figure 8.9. *3D skeleton-based Human action recognition as dynamic 3D point cloud classification.*

where C is the number of classes. The cth element of $\mathbf{y}$, $\mathbf{y}_c$, indicates the likelihood of the dynamic 3D point cloud belonging to the cth class.

Standard experimental setup. There are two standard datasets: NTU-RGB+D and Kinetics. NTU-RGB+D, containing $56,880$ skeleton action sequences completed by one or two performers and categorized into 60 classes, is one of the largest data sets for skeleton-based action recognition Shahroudy *et al*. (2016). It provides the 3D spatial coordinates of 25 joints for each human in an action. For evaluating the models, two protocols are recommended: Cross-Subject and Cross-View. In Cross-Subject, $40,320$ samples performed by 20 subjects are separated into training set, and the rest belong to test set. Cross-View assigns data according to camera views, where training and test set have $37,920$ and $18,960$ samples, respectively. Kinetics is another large data set for human action analysis, containing over $240,000$ video clips **?**. There are 400 types of actions. Due to only RGB videos are provided, 3D skeleton data can be obtained by estimating joint locations on certain pixels with OpenPose toolbox Cao *et al*. (2017). The toolbox generates 2D pixel coordinates (x, y) and confidence score c for totally 18 joints from the resized videos with resolution of 340×256. Each body joint is represented as a three-element feature vector: $[x, y, c]^{\mathrm{T}}$. For the multiple-person cases, we can select the body with the highest average joint confidence in each sequence. Therefore, one clip with T frames is transformed into a skeleton sequence with dimension of $18 \times 3 \times T$).

Standard methods. Some early attempts of skeleton action recognition often encode all the body joint positions in each frame to a feature vector for pattern learning Vemulapalli *et al*. (2014), Fernando *et al*. (2015), Du *et al*. (2015), Liu *et al*. (2017). These models rarely explore the internal dependencies between body joints, resulting to miss abundant actional information. To capture joint dependencies, recent

methods construct a skeleton graph whose vertices are joints and edges are bones, and apply graph convolutional networks (GCN) to extract correlated features Kipf and Welling (2017*b*). The spatio-temporal GCN (ST-GCN) is further developed to simultaneously learn spatial and temporal features Yan *et al*. (2018). ST-GCN though extracts the features of joints directly connected via bones, structurally distant joints, which may cover key patterns of actions, are largely ignored. For example, while walking, hands and feet are strongly correlated. While ST-GCN tries to aggregate wider-range features with hierarchical GCNs, node features might be weaken during long diffusion Li, Han and Wu (2018). Instead of fully relying on the body skeleton structure, some recent works start to learn a data-dependent graph topology to to capture richer dependencies among body joints Si *et al*. (2018, 2019), Shi, Zhang, Cheng and Lu (2019*b*,*a*), Li, Chen, Chen, Zhang, Wang and Tian (2019), Li, Li, Zhang and Wu (2019), Wen *et al*. (2019). Based on the learnt graph topology, one can still apply graph convolution. The experimental results show that with a trainable graph topology, the system achieves significantly better performances.

8.5.3. *Application in point cloud generation*

The goal is to find a compact representation of a 3D point cloud that preserves the ability to reconstruct the original 3D point cloud. Here we only consider the 3D coordinate of each 3D point. Let $\S = \{\mathbf{x}_i\}_{i=1}^N$ be a set of N 3D points, whose ith element $\mathbf{x}_i \in \mathbb{R}^3$ is the 3D coordinate. We aim to design a pair of an encoder $\Psi(\cdot)$ and a decoder $\Phi(\cdot)$, such that $\Psi(\cdot)$ compresses a 3D point cloud $\S$ to a low-dimensional code $\mathbf{c}$ and $\Phi(\cdot)$ that decompress the code back to a reconstruction $\widehat{\S} = \{\widehat{\mathbf{x}}_i\}_{i=1}^M$ that approximates $\S$; that is,

$$\mathbf{c} = \Psi\left(\S\right) \in \mathbb{R}^C \qquad [8.16a]$$

$$\widehat{\S} = \Phi\left(\mathbf{c}\right), \qquad [8.16b]$$

where the code $\mathbf{c}$ with $C \ll 3N$ summarizes the original point cloud. We aim to optimize over $\Psi(\cdot)$ and $\Phi(\cdot)$ to push $\widehat{\S}$ to be close to $\S$. Note that the number of points in the reconstruction may be different from the number of points in the original 3D point cloud. To evaluate the quality of reconstruction, we use either the Earth mover's distance or the Chamfer distance.

Standard experimental setup. A standard dataset is ShapeNet Chang, Funkhouser, Guibas, Hanrahan, Huang, Li, Savarese, Savva, Song, Su, Xiao, Yi and Yu (2015). It contains more than 3,000,000 3D models, 220,000 models out of which are classified into 3,135 categories (WordNet synsets). For each 3D model, one can sample 3D points from the surfaces of these 3D models by using the Poisson-disk sampling algorithm Bridson (2007) and rescale the points into a unit cube centered at

the origin. The evaluation metric is either the Earth mover's distance or the Chamfer distance.

Standard methods. An encoder should extract global features that preserve as much information as possible from an original 3D point cloud. To design an encoder, one can leverage a graph topology to accelerate information diffusion and feature learning. For example, latentGAN Achlioptas *et al*. (2018) directly uses the global feature vector from PointNet to encode the overall geometry information of a 3D point cloud.

A decoder should translate information from the feature space to the original 3D space as much as possible. To design a decoder, the simplest approach is to use fully-connected neural networks, which work well for a small-scale 3D point cloud, but require a huge number of training parameters Achlioptas *et al*. (2018).

To improve efficiency, FoldingNet Yang, Feng, Shen and Tian (2018) and AtlasNet Groueix *et al*. (2018) consider the decoding as a warping process that folds a 2D lattice to a 3D surface. Let $\mathbf{Z} \in \mathbb{Z}^{M\times 2}$ be a matrix representation of nodes sampled uniformly from a fixed regular 2D lattice and the ith row vector $\mathbf{z}_i \in \mathbb{R}^2$ be the 2D coordinate of the ith node in the 2D lattice. Note that $\mathbf{Z}$ is fixed and is used as a canonical base for the reconstruction, which does not depend on an original 3D point cloud. We can then concatenate each 2D coordinate with the code from the encoder [8.16a] to obtain a local feature; and then, use MLPs to implement the warping process. Mathematically, the ith point after warping is

$$\widehat{\mathbf{x}}_i \;=\; g_{\mathbf{c}}(\mathbf{z}_i) = \mathrm{MLP}\left(\left[\mathrm{MLP}\left(\left[\mathbf{z}_i, \mathbf{c}\right]\right), \mathbf{c}\right]\right) \in \mathbb{R}^3,$$

where the code $\mathbf{c}$ is the output of the encoder and $[\cdot,\cdot]$ denotes the concatenation of two vectors. The warping function $g_{\mathbf{c}}(\cdot)$ consists of two-layer MLPs and the code is introduced in each layer to guide the warping process. We collect all the 3D points $\widehat{\mathbf{x}}_i$ to form the reconstruction $\widehat{\S} = \{\widehat{\mathbf{x}}_i \in \mathbb{R}^3, i = 1, \cdots, M\}$.

Intuitively, introducing a 2D lattice provides a smoothness prior; in other words, when two points are close in the 2D lattice, their correspondence after warping are also close in the 3D space. This design makes the networks easy to train and save a huge amount of training parameters.

8.6. Summary and further reading

This chapter overviews the recent progress on 3D point cloud processing via graph-based techniques. 3D points are irregularly scattered in the 3D space, which cannot be directly handled by traditional lattice-based methods. As a general representation format, graphs bring spatial relationships for those points and enable the subsequent processing. In this chapter, we introduce two groups of techniques, including graph

signal processing and geometric deep learning. We also consider their applications to restoration, resampling, segmentation, classification and generation of 3D point clouds.