UNIVERSITY
OF OSLO

**Master's thesis**

# Virtual LiDAR error models in point cloud compression

**Branislav Jenco**

Programming and System Architecture
60 study points

Department of Informatics
Faculty of Mathematics and Natural Sciences


Spring 2022

**Branislav Jenco**

# Virtual LiDAR error models in point cloud compression

Supervisor:
Prof. Carsten Griwodz

# Abstract

As a common output format of sensors used for scanning real world environments, point clouds are a ubiquitous representation of 3D geometry. A relatively inefficient, unordered data structure for this purpose, point cloud processing and compression has been a target of intense research focus. Notably, this focus has been predominantly targeted on simple object scenes, often with uniform sampling and a lack of noise. This is not usually the case for larger room-scale environments. At the same time, a lot of focus on point clouds in the deep learning space has been in extracting semantic information, which necessitates a large amount of training data which is not present for high density point clouds.

The hypothesis is that compression can be achieved by fitting geometric primitives to the point cloud in a supervised manner. Such a system needs datasets with geometric ground truth information and realistic noise distribution. In this thesis, the noise distribution of room-scale point clouds is explored and a virtual VLP-16 LiDAR scanner with a realistic error model is implemented. Existing point cloud compression approaches are evaluated on noisy room-scale data. The results suggest that a Gaussian noise applied to the distance between the sample and sensor is an adequate approximation with the right parameters. The performance of existing signal processing–based compression approaches does not significantly degrade on noisy room-scale data, however their neural network–based counterparts struggle with performance variability. Exploiting geometric information and advances in deep learning on point clouds for compression is an important area for further study.

# Acknowledgements

I would like to sincerely thank my supervisor Prof. Carsten Griwodz for all the help and guidance throughout my work on this master thesis. It can be challenging to produce a master thesis while working full time and he helped me keep my focus and motivation. Our discussions were always interesting and enjoyable, whether it was about topics relevant to the thesis or not. I would also like to thank my wife, Eirill Strand Hauge, who has been supporting me the whole time, kept my spirits up, and proofread my text.

# Contents


**1 Introduction** **1**
- 1.1 Motivation . . . 1
- 1.2 Goals . . . 2
- 1.3 Outline . . . 2

**2 Background** **5**
- 2.1 Point clouds . . . 5
  - 2.1.1 Types of point clouds . . . 5
  - 2.1.2 Point cloud properties . . . 6
  - 2.1.3 Creating point clouds . . . 8
- 2.2 LiDAR . . . 9
- 2.3 Point cloud compression . . . 10
- 2.4 Metrics for comparing point clouds . . . 11
  - 2.4.1 Asymmetric chamfer distance . . . 11
  - 2.4.2 Chamfer distance . . . 11
  - 2.4.3 Hausdorff distance . . . 12
  - 2.4.4 Point-to-plane metrics . . . 12
- 2.5 Point cloud file formats . . . 12

**3 Related work** **15**
- 3.1 LiDAR simulation . . . 15
  - 3.1.1 Sensor measurement models . . . 16
  - 3.1.2 Sensor error models . . . 18
- 3.2 Deep learning on 3D data . . . 18
  - 3.2.1 Volumetric representations . . . 19
  - 3.2.2 Multi-view based approaches . . . 20
  - 3.2.3 2D point map approaches . . . 20
  - 3.2.4 PointNet architecture . . . 21
  - 3.2.5 Graph-based approaches . . . 21
- 3.3 Point cloud datasets . . . 22
- 3.4 Point cloud compression algorithms . . . 23

**4 Tools** **25**
- 4.1 Velodyne VLP-16 . . . 25
- 4.2 Point Cloud Library . . . 26
- 4.3 CloudCompare . . . 27
- 4.4 Open3D . . . 28

4.5 Indoor reconstruction++ . . . . . 28
4.6 BlenSor . . . . . 29
4.7 PCC Arena . . . . . 31

**5 Simulating VLP-16** **35**
5.1 Collecting error measurements from the real VLP-16 . . . . . 35
5.1.1 Implementation . . . . . 36
5.1.2 Procedure . . . . . 36
5.1.3 Results and discussion . . . . . 38
5.2 Base error model in BlenSor . . . . . 45
5.3 Implementing the virtual VLP-16 sensor . . . . . 46
5.4 Evaluation setup . . . . . 47
5.5 Results and discussion . . . . . 49

**6 Compressing point clouds** **53**
6.1 Implementation . . . . . 54
6.1.1 Preparing the *CAPOD* dataset . . . . . 54
6.1.2 Preparing the raw Stanford Large–Scale 3D Indoor Spaces Dataset (S3DIS) . . . . . 55
6.1.3 Creating the scanned datasets . . . . . 55
6.1.4 Algorithm selection . . . . . 56
6.1.5 Metric selection . . . . . 58
6.1.6 Scaling and compactness . . . . . 58
6.2 Results and discussion . . . . . 59

**7 Preparation for geometric primitive fitting** **67**
7.1 Background and related work . . . . . 67
7.1.1 Hough transform . . . . . 68
7.1.2 Random sample consensus . . . . . 69
7.1.3 Region growing . . . . . 70
7.1.4 Robust statistics . . . . . 70
7.1.5 Local statistics . . . . . 71
7.1.6 Generative modeling . . . . . 71
7.1.7 Shape parsing and parsimonious representation . . . 72
7.2 Approach and discussion . . . . . 73

**8 Conclusion and future work** **75**

# List of Figures


2.1 Object point cloud from the CAPOD dataset . . . . . 7
2.2 Indoor scene point cloud from the S3DIS dataset . . . . . 8

4.1 Velodyne VLP-16 Puck . . . . . 25
4.2 Velodyne VLP-16 laser channels . . . . . 26
4.3 Scanned point cloud rendered within Blender's viewport . . 30
4.4 User interface controls for the scanning configuration . . . . 31

5.1 Single acquisition sweep diagram . . . . . 36
5.2 Full scan at 5 m distance from wall . . . . . 37
5.3 Full scan at 1 m distance from wall . . . . . 37
5.4 Three-quarter view of a cropped scan at 5 m from the wall . 39
5.5 Cropped scan at 5 m distance from the floor . . . . . 39
5.6 Side view of the wall scan . . . . . 40
5.7 Samples from a single laser wall scan . . . . . 40
5.8 Side view of the floor scan . . . . . 41
5.9 Samples from a single laser floor scan . . . . . 41
5.10 Real noise distribution between wall and floor scans . . . . . 41
5.11 Noise distribution per distance (wall) . . . . . 42
5.12 Noise distribution per distance (floor) . . . . . 43
5.15 Spread of plane residuals with increasing incidence . . . . . 44
5.16 Error model comparison at 5 meters (wall) . . . . . 49
5.17 Error model comparison at 5 meters (wall) . . . . . 50
5.18 Noise distributions of the different error models . . . . . 51
5.19 Beam divergence error model evaluation . . . . . 51
5.20 Incidence angle and planar residual relationship . . . . . 52
5.21 Incidence angle to noise relationship diagram . . . . . 52

6.1 Example point cloud from the CAPOD dataset . . . . . 55
6.2 Example point cloud from the S3DIS dataset . . . . . 56
6.3 Raw and scanned room point cloud comparison . . . . . 57
6.4 Point count distributions . . . . . 59
6.5 Coarse view of the metrics per compression rate . . . . . 61
6.6 Rate–distortion curves for Chamfer distance . . . . . 62
6.7 Rate–distortion curves for Hausdorff distance . . . . . 63
6.8 Encoding time against bpp comparison . . . . . 64
6.9 Decoding time against bpp comparison . . . . . 65

# List of Tables

4.1 VLP-16 LiDAR sensor specifications . . . . . 26
4.2 VLP-16 vertical angles and corrections . . . . . 27

5.1 BlenSor VLP-16 parameters . . . . . 47

# Chapter 1

# Introduction

As computational resources become cheaper and acquisition systems become more precise and easier to use, there is an increasing focus on finding better ways to process 3D data acquired from the real world. While there are several different types of acquisition systems based on different technologies, they predominantly share a common output format, the point cloud. Fundamentally, a point cloud coming out of a 3D acquisition system is a point sampling of the real world.

The crucial difference between a 2D image and a point cloud is the presence of depth information. In combination with information about the sensor field of view and other particular properties, this allows us to reconstruct the $x$, $y$ and $z$ coordinates of each sample in the resolved image in relation to the sensor and obtain a 3D scene. By combining scans from multiple locations, sample density can be improved and occlusion artifacts can be minimized.

Evolved from geographic surveying, point cloud scans are now ubiquitous in the areas of robotics, such as Simultaneous Localization And Mapping (SLAM) approaches; autonomous vehicles, building information modelling, predictive maintenance, architecture, modelling etc. However, point clouds are arguably a rather inefficient and unwieldy representation of 3D geometry. Therefore, techniques for easier processing and compression are being developed. With the advent of deep learning in the last ten years, more and more researchers are trying to apply these techniques to 3D data to try and achieve advances similar to those in the 2D image domain.

## 1.1 Motivation

Most existing point cloud datasets focus on objects and avatars, less so on more complex indoor or outdoor scenes. In particular, most general-purpose focused research that tries to apply machine learning (ML) techniques to point clouds also deals with these types of datasets. Outdoor scenes can be the focus in the autonomous driving field, where real-time but coarse understanding of the environment is crucial. Indoor scenes have been explored within the construction and Building Information Modelling (BIM) fields, but have been mostly limited on traditional (non-

ML) techniques.

At the same time, most of the effort concentrates on data which is noise-free and often uniformly sampled, assumptions that usually do not hold in real-world scans. Even more so, for inside-out scans created using a light detection and ranging (LiDAR) device, it is much harder to control the noise characteristics, compared to a controlled environment of an object scan.

This thesis is a continuation of the work done by Sillerud and Johanssen, 2019, Boulet-Gilly, 2019 and Hansen, 2020. The central question is whether we can find geometric primitives in real-world noisy indoor scene point clouds, in order to simplify the representation and achieve compression of the geometry data. Supervised approaches to this task will need a large of training data with ground truths. Currently, the extent of this data is quite limited, as it is slow to obtain it from the real world. Existing datasets usually focus on semantic labels and not on geometry information. The idea of generating point clouds synthetically, while modelling the real sensor error characteristics is therefore explored. Specifically, the Velodyne VLP-16 sensor, one of the most commonly used LiDAR devices, was used as the target device for this purpose. At the same time, existing point cloud compression approaches needed to be evaluated on noisy room-scale data, to serve as a benchmark. Finally, extending the work of predecessors, a geometric fitting pipeline for primitives other than planes is explored.

## 1.2 Goals

In light of this, the following goals have been stated for this thesis:

1. Study the error characteristics of the Velodyne VLP-16 LiDAR sensor in indoor environments
2. Create a pipeline for generating synthetic point clouds with a realistic error model from virtual environments
3. Evaluate the performance of existing point cloud compression algorithms on noisy indoor scenes
4. Create a compression pipeline using fitting of geometric primitives

## 1.3 Outline

In Chapter 2, the basic characteristics of point clouds, the different kinds and the intricacies of their creation or generation are described. Following is background information on LiDAR sensors and metrics for quantitatively comparing different point clouds.

Chapter 3 then goes through the wide-ranging related work in this area, approaches to deep learning on 3D data, simulating virtual sensors, point cloud compression and public point cloud datasets.

Afterwards, an overview of the hardware and software tools used in the thesis is described in Chapter 4. Chapter 5 follows with the description of building a virtual LiDAR sensor with a realistic error model. This includes studying the error model of a real VLP-16 sensor and evaluating several different error models in the virtual scanner, followed by the comparison of results and discussion. In Chapter 6, point cloud compression using the *PCC Arena* framework on noisy room-scale datasets is performed and evaluated. Likewise, results are presented and discussed.

In Chapter 7, a theoretical exploration and possible directions of inquiry in the area of geometric primitive fitting are laid out, informed by the work done in this direction.

Finally, Chapter 8 concludes this thesis and recommends avenues for future work in this area.

# Chapter 2

# Background

## 2.1 Point clouds

A *point cloud* is a data structure that is made up of an unordered set of points, each point usually representing a position in 3D space. When referring to a point cloud as a result of some 3D data acquisition system, it can be thought of as a point sampling of the real world (Kaiser, Ybanez Zepeda and Boubekeur, 2019). That means that each point, or *sample*, must have at least three properties associated with it, namely its $x$, $y$ and $z$ coordinates. We can therefore more formally define a point cloud $P$ of $n$ points as

$$P = \{\mathbf{p}_i \mid i = 1, \ldots, n\}, \tag{2.1}$$

where each point $\mathbf{p}_i$ is a vector of its three coordinates, as

$$\mathbf{p}_i = (x_i, y_i, z_i). \tag{2.2}$$

Additional properties can be included with each sample, such as custom labels, RGB color values, texture information, reflectivity or information about the normal vector of the surface the point lies on.

### 2.1.1 Types of point clouds

As it is just a set of points, it is a simple data structure for 3D geometry representation compared to something like a *mesh* which is defined by vertices collected into faces that describe surfaces [1]. This points to the fact that a point cloud is more *general*—it does not have to describe 3D geometry with surfaces, it could be a collection of particles or something similar.

While it is a very simple data structure, individual point clouds can significantly vary in their characteristics. These properties are often dictated by the process with which the point cloud was created and have a large impact on the kinds of applications that the data can be used for.

[1] Another often used term is a Computer Aided Design (CAD) model. Used in manufacturing or construction, CAD models are usually made up of parametrized solids and boolean operations between them.

The type of scene captured by the point cloud is an important characteristic. Across most applications, we can distinguish three types of scenes:

- **Object scenes**, where the goal is to create a highly detailed scan of an object. In cases where the object is a human, these are sometimes called *avatars*. These point clouds represent an *outside-in* approach, where one or more sensors scan the object from multiple angles in a controlled environment. In case of small objects, these can be placed on a rotating plate to minimize inaccuracies. Capturing both the correct *volume* and *surface* information is essential for these use cases. Object scenes are also the most common in synthetic point cloud datasets, usually created by uniformly or randomly sampling a mesh. An example can be seen on Figure 2.1.

- **Outdoor scenes**, in which a sensor captures its surrounding environment, an *inside-out* approach. Use cases include robotics and autonomous localization and navigation, but also construction. In robotics, deriving real-time but coarse understanding of the environment around the sensor is most valuable, whereas in construction, accuracy and precision is more important.

- **Indoor scenes**, where the sensor similarly uses an *inside-out* approach to capture the surrounding room or a different environment that is assumed to be closed around the device. In these types of scenes, the most important information to be captured is the geometry and structure of the visible surfaces. When working on indoor and outdoor scenes, research papers usually hold the *Manhattan world assumption*, which states that most surfaces are aligned with the three principal directions in 3D space (Furukawa et al., 2009). An example of an indoor scene point cloud is shown on Figure 2.2.

### 2.1.2 Point cloud properties

The type of scene and the acquisition process then influence properties like:

- **Scale**, which specifies the relative size of the point cloud. Depending on the acquisition method, this property is often unknown. When creating point clouds from RGB images, some kind of reference is needed to get the correct scale (this can be challenging as the lens distortion has to be taken into account). Time-of-flight based sensors are able to infer the correct scale based on the speed of light.

- **Density distribution**, describing the number of samples per unit of measure. In synthetic point clouds, often created by sampling a mesh, the distribution is usually uniform, but this is rarely the case in real world scans.

- **Accuracy distribution**, measuring the error compared to the real scanned environment. This property is usually important in larger

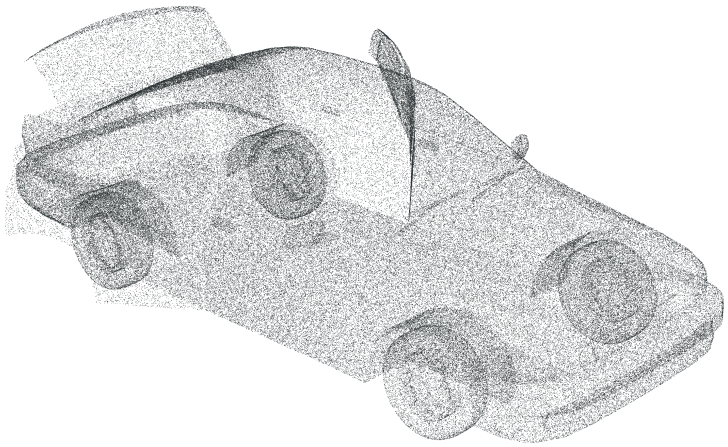


Figure 2.1: An example of a noise-free object point cloud from the CAPOD dataset. The point cloud was generated by randomly sampling a 3D mesh. (Papadakis, 2014)

scenes acquired using light-based sensors, whose accuracy decreases with distance that the light rays travel.

- **Noise distribution**, which can be caused by the generation process or by the noise inherent in the scanned environment. By default, noise is not present in synthetic scans, unless it is artificially added based on some predefined distribution. For real-world scans, the noise comes from the physical characteristics of the sensor, which can be compounded when multiple scans are merged into one. Moving objects and presence of fog or rain contribute to noise as well.

- **Occlusion**, or presence of holes. Unless artificially modeled, this problem is also not present in synthetic scans[2]. Holes created by occlusion are a natural consequence of the fact that the sensor technology depends on line-of-sight light detection.

Another important property is the storage requirement for storing point cloud data. This depends on the size and density of the point cloud. For highly dense point clouds, storage and processing power requirements are very high as point clouds can be thought of as an inefficient representation of 3D geometry. The reason is that, while a triangle surface in 3D space can be most simply described by the coordinates of its three vertices along with the normal vector, in a point cloud, it is likely represented with a variable amount of samples that are usually non-uniformly distributed on the triangle surface.

[2]One of the uses of adding holes artificially is when training deep learning algorithms for inpainting applications. Inpainting describes the process of automatically filling missing data.

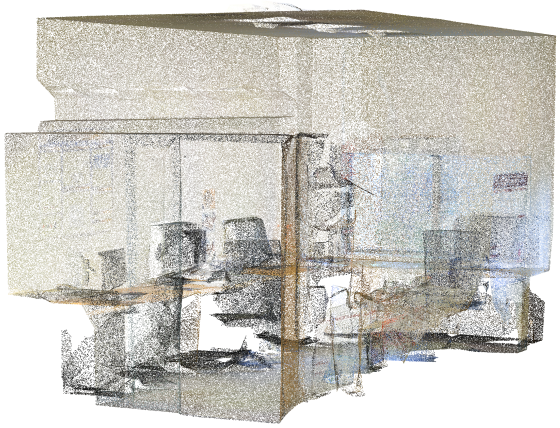


Figure 2.2: An example of an indoor scene point cloud from the Stanford Large-Scale 3D Indoor Spaces Dataset (Armeni, Sener et al., 2016). The point cloud is a result of sampling a mesh generated using the Matterport Camera sensor.

### 2.1.3 Creating point clouds

A point cloud can be generated from digital assets or from real-world environments. Digital 3D objects are usually "scanned" by the means of uniform sampling on a 3D voxel grid. This technique creates point clouds with a predictable and uniform density distribution and accuracy that is controlled by the grid scale. Other techniques include simply sampling the vertices of the object mesh, which creates a sparse point cloud with high accuracy, but low density in flat areas with low frequency information. Unless added artificially, point clouds generated from 3D assets are devoid of noise. A technique employed in this thesis is to take virtual scenes and simulate a scan with a virtual sensor. This allows for more realistic modelling of occlusion and noise.

A multitude of acquisition techniques exist for scanning real-world environments. Small and simple objects can be put on a turntable and scanned by cameras and laser sensors from all angles in a highly controlled environment. This creates a dense point cloud with high accuracy. Purely image-based (photogrammetry) techniques exist as well, such as the structure-from-motion (SfM) and multiview stereo (MVS) algorithms. Robotics approaches using visual simultaneous localization and mapping (also known as SLAM) create fast but low-accuracy point clouds to estimate the path of a camera through space.

Structured light sensors use a projector to emit a known pattern onto the scene that is then captured by a camera. The distortions in the pattern are then used to discern the depth information. This technique was used in the first generation *Microsoft Kinect* device and in the *Matterport Camera*

(*Matterport Pro2 3D Camera* 2022). Time-of-flight sensors (also known as range imaging sensors) measure the distance between the sensor and points in the environment based on the round-trip time of a light signal to create samples of points. These can either based on visible or infrared light such as the later generation *Microsoft Kinect* or use lasers, in the case of LiDAR devices, such as the *Leica BLK360*, *Velodyne HDL-32E*, *HDL-64E* and the sensor used in this thesis, *VLP-16* (Bi et al., 2021).

## 2.2 LiDAR

LiDARs (light detection and ranging) are time-of-flight sensor systems that are used to scan real-world environments into point clouds by measuring the round trip time of laser rays emitted from the device that reflect back from the environment. The generated datasets have applications in e.g. photogrammetry, digital twin construction or autonomous vehicles.

The simplest LiDAR devices scan the environment using only a single ray, and have been historically used in land surveying and earth observation satellites. Probably the most common handheld sensors today use a rotating mirror to scan the entire environment around the device, in one or more planes. Solid-state LiDARs without moving parts also exist, with a field of view similar to a camera (Intel, 2020).

The physical characteristics of a spinning LiDAR dictate that the spatial resolution quickly decreases with distance and so does the accuracy. The sensor is scanning one or more strips of vertical space by rotating around an axis, which means that the density between these strips (orthogonal to the rotation axis) is especially low. Reflectivity and translucency of the scanned materials also plays a huge role in the quality of data gathered. The sensor will fail to detect objects made up of transparent materials and will have problems with specular surfaces. Diffuse surfaces and retro-reflective surfaces work best. An obvious limitation of LiDARs is occlusion, where surfaces that are not directly visible to the sensor cannot be detected.

Because of these issues, creating high-density point clouds with affordable LiDAR systems today means they must stand still or follow a highly exact timing along a very well-known movement path and take several scans of the same environment. However, this technique requires the addition of an inertial measurement unit (IMU), unless it is already present in the LiDAR device. For sensors traversing larger distances outdoors, a GPS unit might need to be used as well. These units have their own noise distribution and accuracy characteristics that compound the problem of creating accurate scans.

Merging these scans into a single, larger point cloud, also called *densification*, is a challenging task as points in two respective scans do not have a correspondence. Sequential scanning also requires that no objects in the scene are moved between the data collection steps. However a successfully merged high-density point cloud can alleviate the aforementioned limitations of occlusion, spatial resolution and noise.

## 2.3 Point cloud compression

As the use of point clouds increases, the need for a more efficient representation becomes more and more pressing. In general, data compression simply means the process of encoding information in a more efficient manner than the original representation. This can be quantified by the fact that a compressed representation uses fewer bits to store the data. Compression can be lossy or lossless. Compressing and decompressing data always consumes computational resources and this must always be weighed against the achieved compression ratio.

Compressing point clouds is challenging, as it is a data structure that is sparse and unstructured yet often needs high-precision. Over the last decade, research on point cloud compression algorithms expanded along several axes. First, there exist several algorithms based on a more traditional signal processing (SP)–based approach. More recently, neural network (NN)–based algorithms became the focus much like in the image domain. Second, the research has focused on both static point clouds and dynamic point clouds with a temporal aspect, especially important in streaming virtual reality (VR) and augmented reality (AR) applications. Third, the algorithms differ in how much of the available information they can work with. Several of the SP–based approaches take into account the sample color information, whereas the NN–based techniques usually only work on the coordinate data (Wu, Hsu, Hung et al., 2022).

Point cloud compression algorithms usually adhere to a basic structure that is composed of three steps:

1. **Pre-processing**

   In the first step, the point cloud might undergo a *scaling* or *normalization* operation, e.g. for the purposes of easier comparison between different point clouds or datasets. A *partitioning* step is sometimes performed to divide the point cloud into smaller blocks. This might be used to decrease the search space, in the case of neural network–based algorithms.

2. **Redundancy elimination**

   The second step is usually performed separately for the point coordinates and for their attributes (if the attributes are handled at all). The operations here can be divided into *transformations* and various types of *quantization*. Transformations might include projection onto 2D planes (to then use familiar 2D algorithms), or employing a tree partitioning data structure like KD-trees or octrees [3]. Quantization refers to reducing the number of points in the

[3] These techniques fall under the umbrella of spatial point cloud partitioning. A *KD-tree* is a general space-partitioning data structure operating in a $k$-dimensional space. A particular variation used in 3D point clouds, called an *octree*, is a tree where each node has eight children. It can be built up from a point cloud by way of recursively subdividing the 3D space, starting with a single node surrounding all the points of the point cloud. The subdivision stops when a condition is met, such as the minimal size of a node, or the minimum number of points that fall into a node.

cloud along a voxel grid, optionally merging points where necessary, or interpolating between their attributes. Deep learning based algorithms usually employ an *Autoencoder* or *Variational Autoencoder*, which integrates the redundancy elimination into a single step.

3. **Entropy coding**

   Finally, the results of the previous step are fed into some kind of an *entropy code* that exploits redundancies in data, e.g. arithmetic coding, where finding frequency of patterns in the data allows for use of fewer bits to store repetitive symbols. Neural network–based algorithms usually call this step an *entropy bottleneck layer* (Jianqiang Wang, Ding, Z. Li and Ma, 2021).

## 2.4 Metrics for comparing point clouds

To be able to quantitatively compare point clouds, e.g. for the purposes of evaluating a compression algorithm, several metrics can be used. For some use cases, a valid approach would be to project the point clouds onto 2D images, which are then compared. This however does not work well to detect geometric distortions. A number of 3D based metrics have therefore been developed.

### 2.4.1 Asymmetric chamfer distance

Asymmetric chamfer distance (ACD) is a directional metric that describes the average of the minimal distances between points from point cloud 1 to point cloud 2. It is defined as

$$\mathrm{ACD}(\mathrm{P}_1, \mathrm{P}_2) = \frac{1}{|P_1|} \sum_{\mathbf{p}_i \in \mathrm{P}_1} \min_{\mathbf{p}_j \in \mathrm{P}_2} \|\mathbf{p}_i - \mathbf{p}_j\|_2^2, \tag{2.3}$$

where $\mathrm{P}_1$ represents the set of points in point cloud 1 and $\mathrm{P}_2$ represents the set of points in point cloud 2. $\mathbf{p}_i$ and $\mathbf{p}_j$ represent a single sample (point) from $\mathrm{P}_1$ and $\mathrm{P}_2$ respectively. Each sample is a vector that consists of the three $x$, $y$ and $z$ coordinates.

### 2.4.2 Chamfer distance

Chamfer distance (CD) is the average of the Asymmetric chamfer distance (defined in Equation (2.3)) from both directions, defined as

$$\mathrm{CD}(\mathrm{P}_1, \mathrm{P}_2) = \frac{1}{2} \left[ \mathrm{ACD}(\mathrm{P}_1, \mathrm{P}_2) + \mathrm{ACD}(\mathrm{P}_2, \mathrm{P}_1) \right]. \tag{2.4}$$

Unlike the Assymetric chamfer distance, the Chamfer distance is symmetric, meaning that the order of point clouds does not matter ($\mathrm{CD}(\mathrm{P}_1, \mathrm{P}_2) = \mathrm{CD}(\mathrm{P}_2, \mathrm{P}_1)$).

### 2.4.3 Hausdorff distance

The Hausdorff distance (HD) is the maximum distance among all pairs of the nearest points. As such it is very sensitive to outliers (Wu, Hsu, Kuo et al., 2020). Just like the Chamfer distance, it is symmetric and it is defined as

$$\mathrm{HD}(\mathrm{P}_1, \mathrm{P}_2) = \max(\max_{\mathbf{p}_i \in \mathrm{P}_1}(\min_{\mathbf{p}_j \in \mathrm{P}_2} \|\mathbf{p}_i - \mathbf{p}_j\|_2^2), \max_{\mathbf{p}_j \in \mathrm{P}_2}(\min_{\mathbf{p}_i \in \mathrm{P}_1} \|\mathbf{p}_i - \mathbf{p}_j\|_2^2)). \quad (2.5)$$

In other words, for each point in cloud $\mathrm{P}_1$, it first computes the minimal distance between it and all the points in cloud $\mathrm{P}_2$. Then, it takes the maximum of these computed values from all the points in $\mathrm{P}_1$ and subsequently performs the same operation from the other direction. Finally, it takes the maximum of these two values.

### 2.4.4 Point-to-plane metrics

The above metrics can be thought of as point-to-point metrics, looking only at the points by themselves. As noted by Tian et al. (2017), point-to-point metrics fail to account for the fact that points in a cloud mostly represent surfaces. This has been alleviated partly by constructing a mesh from one of the point clouds and measure a *point-to-mesh* metric, but this approach is dependent on the technique used to generate the mesh. The authors therefore propose a middle ground approach called *point-to-plane* metrics.

The difference between point-to-point and point-to-plane metrics is that the latter emphasize the errors when a point is straying further from the assumed surface. This is because instead of just measuring the distance between two points, they measure it along the normal vectors. This aligns better with the way humans perceive errors naturally. A requirement of this approach is to therefore have the normal information present in the point cloud.

All three of the mentioned point-to-point metrics can be adjusted to a point-to-plane metrics by incorporating the normal to the distance measure, as given by

$$\|\mathbf{p}_i - \mathbf{p}_j\|_2 \cdot \mathbf{N}_{\mathbf{p}_i}, \quad (2.6)$$

where $\mathbf{N}_{\mathbf{p}_i}$ is the normal vector of the surface at the location of point $\mathbf{p}_i$.

## 2.5 Point cloud file formats

A point cloud is a relatively simple, essentially columnar, data structure and over time, several different file formats to encode it have evolved. These can be divided into ASCII-based and binary file formats.

The simplest way to store a point cloud is to use a plain textfile, where every line corresponds to one point. Within one line, the individual properties of the point can be separated by a whitespace or other delimiter. Usually the first three correspond to the $x$, $y$ and $z$ values of the point's coordinates, with other properties like color, or normal information

following. If the delimiter is a comma, and a header is added with a name of each column, it becomes a `.csv` file, though rarely used for point clouds.

More established formats for 3D computer graphics are also often used to store point clouds. Formats like `.ply`, `.obj`, `.x3d` have the capability to store point coordinates, and subsequently which faces the points belong to, along with texture and other information. To store a point cloud in these means simply omitting storing any information apart from the point positions.

Created as a part of the Point Cloud Library (described in Section 4.2), the `.pcd` format was created specifically for storing point clouds. It has a textual and a binary version, with an initial header storing metadata about the point cloud, with rows of point data following afterwards (*The PCD File Format* 2022).

# Chapter 3

# Related work

This thesis is part of a larger project and builds on previous work done by Sillerud and Johanssen (2019), Boulet-Gilly (2019) and Hansen (2020). Sillerud and Johanssen built a data acquisition system combining the Velodyne VLP-16 LIDAR sensor with an inertial measurement unit, which allows them to construct high-density point clouds from multiple scans using pairwise registration of the scan fragments and pose graphs. They also systematically measure the errors of their setup, which is an important step in trying to simulate the sensor behaviour in virtual environments. Boulet-Gilly explores the possibilities of building more compact representations of the computationally intensive high-density point clouds. Compression can be achieved by detecting planes and using high or low polygon models of the scanned environment along with a height map. Hansen improved the work of Sillerud and Johanssen (2019) with *Indoor reconstruction++*, a framework using sampling and several variants of the ICP algorithm. He looked at using machine learning techniques for flat surface segmentation using spherical kernels, combining it with the height map approach of Boulet-Gilly (2019). Finally, using autoencoder networks for point cloud compression was explored. The combined work has also been released as a research paper (Hansen et al., 2020).

## 3.1 LiDAR simulation

Applying supervised deep learning methods on point clouds requires a large amount of labeled training data. Obtaining the necessary volumes of point clouds with ground truths in the real world is often a tedious and ineffective process. Researchers have therefore been exploring possibilities of generating labeled point clouds programmatically. This can be done by simulating a sensor gathering data from a virtual environment, or by using generative approaches based on the statistics of real LiDAR point clouds. As mentioned by Manivasagam et al. (2020), simulating ranging sensors goes back to the work of NASA and JPL in the 1970s in the field of aerospace research. The 1990s and 2000s showed a surge of interest in the context of aerial (often military) applications as first LiDAR sensors became

available. Since the proliferation of machine learning approaches in the last ten years, research in this area has been focused on robotics and especially autonomous driving, spurred also by a higher availability of cheaper and more robust sensors.

The dichotomy of point cloud simulation versus generation is analogously summarized by Hanke et al. (2017). The authors distinguish two types of sensor models in the context of LiDAR error simulation:

1. *sensor measurement models* - based on a *physical* description of the measurement process. The goal is to generate low level measurement data based on the virtual scene.

2. *sensor error models* - these aim to reproduce *statistical* characteristics of errors, i.e. deviations between the perceived and true values, of the measurement and perception performed by sensors.

Manivasagam et al. (2020) also touch on this area, distinguishing between *physics-based* and *learning-based* simulation. They argue these approaches can be combined for a better performance and smaller sim-to-real domain gap.

### 3.1.1 Sensor measurement models

Simulating LiDAR sensors for point cloud generation can also be thought of as a synthetic data generation problem. The starting point for this area of research has been the extensive work done in the domain of synthetic ground-truth data generation for images, such as in *SYNTHIA* (Ros et al., 2016), where authors create a synthetic collection of diverse urban images with automatically generated labels for use in semantic segmentation. Another inspiration has been the use of games to obtain the ground truths (Richter et al., 2016). As 3D open world games and their engines are essentially simulation packages, they can be used to generate synthetic visual data of varying quality and with varying amount of tweaking needing to be done to the game itself. An example of this is *CARLA*, a driving simulator based on the Unreal Engine 4, used in (Dosovitskiy et al., 2017). The simulator has an extensible architecture, so that subsequent researchers have been able to improve its performance, accuracy and features.

The general approach of the last several years in the field of robotics and autonomous driving has been to use a simulation package (like *CARLA*), a game engine (like *Unreal* or *Unity*) or a video game (most often *GTA V*) and apply the method of ray casting for a physical simulation of the LiDAR sensor. This method is then sometimes augmented by adding varying degrees of error and noise based on the observations from real world LiDAR scans.

Hanke et al. (2017) propose a sensor measurement model which utilizes ray casting in the *Vires VTD* driving simulator. They also simulate beam divergence (the phenomenon were laser rays slowly widen as they travel through space) using multiple rays to sample the area illuminated by a

laser beam. To simulate dropping returns which are not strong enough, they apply an intensity threshold and to model the distance noise they add a white noise to the measured distance which is also affected by the reflectivity of the material. To compare virtual and real scans, they apply an overall error, and measure the correlation coefficients on occupancy grids formed by the point clouds.

In the area of semantic point cloud segmentation, Yue et al. (2018) try the ray casting approach in the video game GTA V, where they fetch coordinates of every ray hit, along with the object id and category. On the other hand, F. Wang et al. (2019) base their work on *CARLA*, with a similar ray casting approach. However, neither of these papers explores the idea of generating noisy point clouds. Fang et al. (2020) acknowledge that it is not enough to simply render the scene depth without considering sensor characteristics. They work with a physical model of the Velodyne HDL-64E S3 sensor which takes into account the incident angle and air attenuation rate. Their findings suggest that the real sensor has some noise in the vertical angles that the lasers are pointing at on the order of 1° to 3°. They approximate this noise by using a Gaussian distribution, setting the distance noise variance to 0.5 cm and the azimuth angular noise variance 0.05°. Additionally, they simulate missing returns by randomly dropping some points in the resulting cloud.

The already–mentioned Manivasagam et al. (2020) also utilize ray casting over the virtual environment, however the authors create an entirely new simulation package called *LiDARsim*. To produce realistic deviations, the authors augment this approach with a deep neural network. They argue that existing driving simulators like *CARLA* and *AirSim* use handcrafted assets and simplified physics, which results in a large domain gap. A large part of their work is building 3D static maps and dynamic object meshes by driving around cities with a vehicle fleet and accumulating information. A deep neural network is used for simulating a phenomenon called *raydrop*. According to their findings, real-world scans have less points than the virtual ones, because in a virtual environment, all cast rays that hit something will return. In a real world, atmospheric effects and other disturbances will make some rays not register in the detector. The network is trained on the real world data and applied to the synthetic ones to simulate the raydrop effect. Simulating this effect is also a recurring theme, found in the work of Fang et al. (2020) and Hanke et al. (2017) under different names.

In Sobczak et al. (2021), the authors simulate the Velodyne VLP-16 sensor in the Unity game engine, with a familiar method of ray casting. They utilize a real-world setup with a small robot driving in a labyrinth and its virtual equivalent. Similarly to other works, they compare the real and virtual point clouds by removing unnecessary points and simply finding distances to nearest points between the clouds, which gives them about 6–6.5 mm average distance. This error goes down to 1–2 mm after manually aligning the point clouds. An interesting consideration not seen in other research has been the problem of rolling shutter in moving vehicles, which they simulate in their model. This effect often seen on DSLR cameras is also

visible in LiDAR sensors mounted on fast moving vehicles. Additionally, an unspecified type of noise is added in their simulation.

### 3.1.2 Sensor error models

Unlike the previous physics-based approaches, the work of Caccia et al. (2019) explores using generative models on LiDAR scans. Instead of using a virtual environment and simulating the sensor, the authors use a LiDAR scan dataset to teach their model how to reconstruct point clouds with noisy or missing data, or generate entirely new ones. This puts their models squarely into the aforementioned sensor error model category as described by Hanke et al. (2017). It builds on the work of B. Li, T. Zhang and Xia (2016) mentioned in the previous section, also utilizing the 2D point map representation of a point cloud (described in Section 3.2.3). Instead of object detection, the authors focus on point cloud generation using variational autoencoders (VAE) and generative adversarial networks (GAN) as well as traditional convolutional neural networks (CNNs).

Diverging from the approach of B. Li, T. Zhang and Xia (2016), this work proposes a different transformation from a point cloud to a 2D grid. Points are first clustered by their elevation angle into a number of clusters and then the 360°plane is divided into a different number of bins which store the average coordinates of the points that fell within. They argue that this transformation, which they call *Cartesian*, provides better performance on noise or incomplete data than the transformation of the predecessors (which they call *Polar*). The authors use the *KITTI* dataset (Geiger et al., 2013) as their input and the *Earth Mover's Distance* as their evaluation criterion [1].

Anderson et al. (2019) explore creating annotated data from virtual grassland plots for the purpose of segmenting vegetation from the ground. The authors state that key disadvantage of existing deep learning implementations is the dependence on annotated data for supervised learning. Much like in this thesis, they use the open source 3D creation suite *Blender* to create annotated data from grassland. This data is then sent to *PointNet++* (described in more detail in Section 3.2.4) for semantic segmentation. They find that the main hurdle to overcome is minimizing the differences between the synthetic and the real domain and they propose the use of an adversarial framework.

## 3.2 Deep learning on 3D data

Over the last decade, computer vision has seen advances in tasks such as object recognition (He et al., 2016), classification (Krizhevsky, Sutskever

[1] The Earth Mover's Distance (also known as the Wasserstein metric) measures the distance between two probability distributions. The term Earth Mover's Distance comes from the fact that the metric can be interpreted as the minimum cost of making one distribution into another—if we think of distributions as piles of dirt, as the cost of moving dirt between them. The cost then depends on how much dirt has to be moved and how far away (Rubner, Tomasi and Guibas, 1998).

and Hinton, 2017) and segmentation (Shelhamer, Long and Darrell, 2017), achieved with the use deep convolutional neural networks. Applications of image reconstruction and generation have also been explored with architectures, such as autoencoders (Rumelhart and McClelland, 1987), generative adversarial networks (Goodfellow et al., 2014) and other generative models. In the last several years, approaches using deep learning on point clouds have been explored from several angles. These are usually inspired by their already much more established counterparts in the 2D image domain, generalizing the idea of convolutions to point clouds. A recent development in this area is applying Transformer architectures to images and point clouds, which originated in natural language processing. In these architectures, convolution is replaced by a self-attention mechanism.

Traditional CNN architectures process two-dimensional, structured, Euclidean data. However, as Liu et al. (2019) sum up, there are three main challenges when learning from point clouds:

1. they are unordered sets, thus requiring the learned representation being permutation invariant

2. they distribute in 3D geometric space, thus demanding the learned representation being robust to rigid transformation (e.g., rotation and translation)

3. they form an underlying shape, therefore the learned representation should be of discriminative shape awareness

While the first point can be resolved by using symmetric functions (exemplified by the *PointNet* architecture (Qi, Hao Su, Mo et al., 2017)), the other two are not yet fully explored. Some of the aformentioned transformer architectures are able to achieve equivariance under rotations, translations and permutations (Fuchs et al., 2020). A significant part of research into machine learning on point clouds is therefore spent on trying to come up with representations that are more amenable to existing (whether that's 2D or graph) ML approaches, while not losing important spatial information contained in the point set. Central to this effort in many applications is the extension of the convolution operator to work on these unordered sets, often using graph theory.

### 3.2.1 Volumetric representations

The most straightforward way of bridging the 2D-3D divide is to represent the 3D space with volumetric approaches, effectively voxelizing the input data (whether that's point clouds or meshes). This transformation gives us back the rigid ordered Euclidean structure that is exploited in images, and 3D convolutions can be applied. However, these representations are severely constrained by the computational cost of 3D convolution as well as by the data sparsity and noise characteristics in the inputs. Most importantly, they are only useful when the volume of shapes and objects

is important, and not so much the surface description. For detecting primitives in indoor point clouds, representing entire objects with their inner volume voxels is very inefficient.

*3D ShapeNets* by Zhirong Wu et al. (2015) was one of the first applications of this approach. The authors represent geometric 3D shapes as a probabilistic distribution of binary variables on a 3D voxel grid and use a Convolutional Deep Belief Network for Computer Aided Design (CAD) model classification. As part of their work, they also created the widely used *ModelNet* object dataset. To get around the problems with data sparsity, *OctNet* by Riegler, Ulusoy and Geiger (2017) uses a hierarchical partitioning of the space using a set of unbalanced octrees where each leaf node stores a pooled feature representation. While network operations such as convolution and pooling have to be reformulated on this new data structure, the result is a significant reduction in computational and memory requirements. The network can therefore work on the space with a higher resolution than traditional voxelization approaches. *VoxelNet* on the other hand uses equally spaced 3D voxels, which are encoded using a novel voxel feature encoding layer, and then uses a Region Proposal Network (RPN) for final detection (Y. Zhou and Tuzel, 2018).

### 3.2.2 Multi-view based approaches

To try to avoid the computational cost of moving from 2D to 3D, earliest approaches to learning on 3D data utilized the traditional CNN architectures by taking pictures of 3D models to be recognized from several angles and focused on learning from the resulting images. Hang Su et al. (2015) outperformed many existing classifiers that directly work on 3D data by using a fixed set of rendered views of a 3D object. The resulting images were fed to a traditional CNN pre-trained on the widely available *ImageNet* dataset. Subsequently, Qi, Hao Su, Niebner et al. (2016) improved this approach using improved training data augmentation and a new multi-resolution filtering component.

Much like the volumetric approaches, multi-view techniques have been more suited to shape classification tasks in the object scene type as defined in Section 2.1.1. They work with the idea that the scanned object is fully detectable from all positions, an assumption which often does not hold in scanned real-world indoor environments.

### 3.2.3 2D point map approaches

Another potential path to working effectively with point clouds is to represent the data structure as a 2D point map and apply traditional CNN architectures built for image data. In B. Li, T. Zhang and Xia (2016), the authors tackle the task of detecting bounding boxes in point clouds captured by the Velodyne HDL-64E sensor. They project the points from the scan and discretize them into pixels on a 2D point map, which can be thought of as a cylindrical image projection from the point of view of the

sensor. In case multiple samples are projected onto the same 2D position, the nearest sample is used.

Pixels in the resulting image contain two channels ($d$, $z$), representing their original position in the point cloud. The original $x$ and $y$ coordinates are coupled together in distance $d = \sqrt{x^2 + y^2}$ to maintain rotation invariance around $z$. The network finds points which are on vehicles (foreground) and which are the background. A bounding box corner is translated to ensure the CNN works on a smaller range and rotated to ensure rotation invariance of the encoding. The output feature maps represent the proposed bounding boxes as a 24-dimensional vector encoding the positions of the 8 corner points. This approach is adjacent to research on deep learning on RGBD images, where the depth channel can be thought of as a range scan.

### 3.2.4 PointNet architecture

Instead of transforming point clouds into different representations, the *PointNet* (Qi, Hao Su, Mo et al., 2017) architecture is made to directly consume unordered point clouds for use in many different applications. This architecture needs to be invariant to permutations of the samples and to rigid motions. In the first stage, it processes each sample independently, learns point-wise features with shared Multilayer Perceptrons (MLPs) and, in the second stage, extracts a global feature with max pooling instead of using the convolution operator. As the authors state, it turns out that the network learns to summarize an input point cloud by a sparse set of key points, which roughly corresponds to the skeleton of objects. A limitation of this approach is that it does not capture local structures that the points make up.

The authors address this in *PointNet++* (Qi, Yi et al., 2017), a hierarchical neural network that first partitions the set of points into overlapping local regions and applies the original *PointNet* network to these regions. The obtained local features are then grouped into larger units to produce higher-level features. In other words, *PointNet++* is *PointNet* applied recursively on a nested partitioning of the input set.

### 3.2.5 Graph-based approaches

Graph Neural Networks (GNNs) are types of networks that are applied to data that can be represented as a graph. This might be social network graphs, molecules and other graphs in biology and chemistry, scene graphs in computer vision and others. Point clouds by themselves are not a graph data structure but can converted into one by employing the notion of a *point neighbourhood*. Each point in the cloud can be thought of as a vertex in the graph, and all the points within it is neighbourhood are connected to it.

Most of GNN research then focuses on coming up with efficient ways to adapt the convolution operation to the graphs. The benefits of using convolutions include localization and parameter reduction. Neighbourhood–based graphs constructed from point clouds exhibit similar locality prop-

erties as in images—points close to each other have a shared influence between each other.

As an example, Lei, Akhtar and Mian (2019) use a spherical kernel for learning on point clouds. A 3D sphere imitates the role of a 2D convolution kernel in image learning. The spherical kernel is partitioned into a set of bins each with its own weight matrix. For each target point, the kernel defines its neighbourhood and thus the points that fall within the individual partitions of the sphere. Additionally, they use octrees as a data structure to base the neural network on. This approach was also studied in more detail in the work of Hansen (2020).

Landrieu and Simonovsky (2018) conclude that LiDAR point clouds are too massive to employ deep learning methods directly. They take inspiration from the approach of superpixels used in images, and partition the point clouds into a graph of superpoints using a global energy model. The superpoints are geometrically simple, homogeneous shapes which makes them computationally efficient. At the same time, they can have varying scale so that they are adaptive to local geometry. Several features that geometrically determine the local neighbourhood of every point are computed. The superpoints are connected with superedges with features describing the relationships between them.

The Transformer architecture has been making strides in the domain of images and and now point clouds. One of the most important is the work of Fuchs et al. (2020). The authors use a variant of the self-attention module present in most Transformer architectures, adapted to 3D point clouds and graphs, making the module one of the first examples of a layer that is equivariant under rotations, translations and permutations. According to the authors, using self-attention in contrast to extending the convolution operator allows for more natural handling of edge features.

## 3.3 Point cloud datasets

Several often–used point cloud datasets have been created within the last decade. In this thesis, the Stanford 2D-3D-Semantics Dataset is used to evaluate point cloud compression algorithms. It is a large scale dataset of indoor spaces, spanning 6 indoor areas. The spaces include offices, hallways, auditoriums and other types of rooms that can be usually found on a university campus. To acquire this data, researchers used the Matterport Camera, which uses the structured light approach with 3 light sensors to capture 18 RGB and depth images during a 360°rotation. Scans were performed at many positions within the areas and combined together by the camera itself. The outputs from the device are already reconstructed 3D textured meshes, so in order to get point cloud data, the authors performed a dense uniform sampling of these meshes (Armeni, Sax et al., 2017).

The dataset was first presented in Armeni, Sener et al. (2016) as the *Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS)* in which authors focused on semantic parsing of the resulting point clouds. As part of

Armeni, Sax et al. (2017) the dataset was extended with a new area and more modalities, namely 2D RGB and depth images, as well as the textured and semantically labeled meshes, and called the *Stanford 2D-3D-Semantics Dataset (2D-3D-S)*.

Among the most used object datasets for benchmarking point cloud processing systems, *ModelNet* and *ShapeNet* have become almost ubiquitous. ModelNet contains 3D CAD models of various different classes, along with a reduced dataset of 10 most popular classes. The models have been manually classified (Zhirong Wu et al., 2015). Similarly, ShapeNet's goal was to establish a richly-annotated, large-scale dataset of 3D shapes with alignment information (Chang et al., 2015).

## 3.4 Point cloud compression algorithms

Work on point cloud compression (PCC) is similarly lagging behind its 2D image counterparts. As mentioned in Section 2.3, one of the main differentiators between the techniques is whether they are based on a more traditional signal processing approach or a neural network approach.

In the area of signal processing–based approaches, *Draco* is an open source algorithm from Google, created initially specifically for the browser, having a C++ based encoder and a C++/JavaScript based decoder. It has both lossy and lossless modes and can compress point coordinates as well as any attributes. It first performs a quantization followed by a KD-tree encoding for the point coordinates and predictive coding for their attributes. At the end standard entropy coding tools are applied (Wu, Hsu, Hung et al., 2022). *Draco* is meant to be used both for point clouds and meshes and the compression parameters allow for controlling the overall compression rate as well as individual quantization rates for coordinates, texture coordinates, point normals and other generic attributes separately (*Draco 3D Graphics Compression* 2022).

The Moving Picture Expert Group (MPEG) has also been focusing on point cloud compression for the past several years. The results of their efforts include the V-PCC codec, meant for point cloud videos and G-PCC, for static point clouds. V-PCC (Video Point Cloud Compression) uses 2D projections, whereas G-PCC (Geometry Point Cloud Compression) utilizes the octree data structure (*MPEG Point Cloud Compression* 2022). G-PCC supports both lossy and lossless compression and shares some of the structure with *Draco*. The first step is voxelization based on rounding floating point numbers to integers. Unlike in *Draco*, duplicate points in a voxel are merged. The octree structure then encodes the geometry, followed by an arithmetic coding (Wu, Hsu, Hung et al., 2022).

Among the NN-based counterparts, *GeoCNNv1* by Quach, Valenzise and Frederic Dufaux (2019) uses shallow 3D convolutional layers and a 3D autoencoder paired with an entropy bottleneck layer. It supports only lossy compression and works only on the point coordinates, similar to most of the NN-based algorithms. *GeoCNNv1* uses a loss function which is a weighted sum of the *focal loss* and the bitrate. The focal loss compensates

for the fact that in a voxelized point clouds, the vast majority of the voxels are empty. Their main criticism of SP-based approaches is that tree-based structures like the octree and KD-tree produce blocky results. *GeoCNNv1* was trained on the ModelNet40 dataset and to use different compression rates, different pre-trained models need to be used (Wu, Hsu, Hung et al., 2022).

In Quach, Valenzise and Frédéric Dufaux (2020), the previous technique is improved by using a 3D Variational Autoencoder (VAE) and a deeper network. Even though its use was criticized in the first version, *GeoCNNv2* also uses the octree structure to partition the point clouds into blocks. These improvements allow for processing bigger point clouds, reducing the memory requirements and compression time.

Jianqiang Wang, Zhu et al. (2021) introduce *PCGCv1* which also uses a 3D VAE along with layers of an architecture called VoxceptionNet. These layers are based on several stacked blocks based on the *ResNet* architecture. *PCGCv1* performs the traditional steps of scaling, voxelization and partitioning before data is sent to the NN layers. It works only on point coordinates and was trained on the *ShapeNet* dataset.

*PCGCv1* was improved by Jianqiang Wang, Ding, Z. Li and Ma (2021) by applying it to the point cloud hierarchically, in a multiscale approach called *PCGCv2*. To reduce memory and time requirements, sparse convolutions on voxels were used. Both *PCGCv1* and *PCGCv2* use a similar compression rate control mechanism as *GeoCNNv1* (Wu, Hsu, Hung et al., 2022). The authors recently improved this approach with SparsePCGC, utilizing sparse tensor processing to reduce the redundancies that come with voxelized representation of 3D space (Jianqiang Wang, Ding, Z. Li, Feng et al., 2021).

# Chapter 4

# Tools

## 4.1 Velodyne VLP-16

The specific sensor used in this thesis is the Velodyne VLP-16, recently renamed to Velodyne Puck (*Puck Lidar Sensor, High-Value Surround Lidar* 2020). The VLP-16 sensor uses an array of 16 infra-red laser emitters paired with detectors to measure distances to objects. Figure 4.1 shows the sensor, with the lasers hidden inside. The laser beams are produced by solid-state

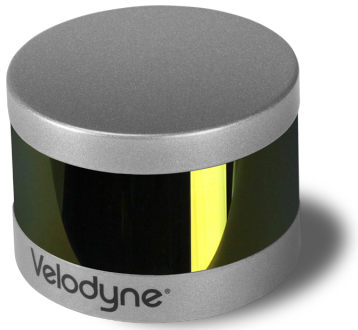



Figure 4.1: The outer shell of the Velodyne VLP-16 (Velodyne Puck). The sensor can be attached to a tripod or a mount using a screw.

semiconductor laser diodes. The array sits on a rotating head spinning around the vertical axis, capable of spin rates from 300 RPM to 1200 RPM. The laser channels are vertically spaced out over a 30° interval (15° up and down from the normal vector of the rotation axis) in 2° increments, as can be seen in Figure 4.2. This setup produces a 360° horizontal and 30° vertical field-of-view (*Velodyne LiDAR: VLP-16 User Manual* 2019). The lasers do not fire in a linear sequence, but rather alternate or "hop around" to avoid interference with each other. The mapping between the laser channel ID and the vertical angle is provided in Table 4.2.

The manufacturer determines the usable measurement range at about

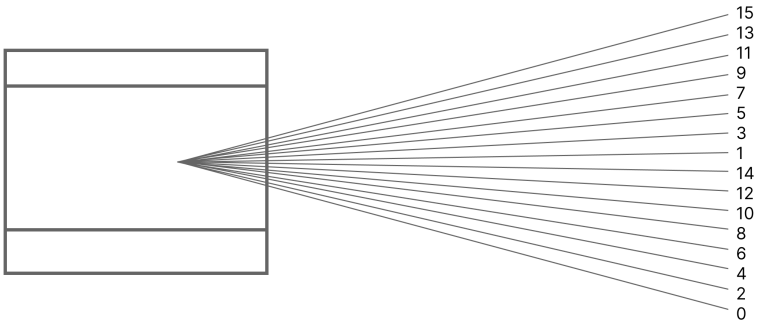



Figure 4.2: Velodyne VLP-16 laser channels. Each channel has a unique ID. The IDs are mapped to the lasers as their vertical angle increases in an alternating sequence, starting from 0 at the lowest laser channel. The specific mapping is shown in Table 4.2.

100 meters with beam divergence of about 3.0 mrad (0.18°) in the horizontal axis and 1.5 mrad (0.09°) in the vertical axis. The actual laser spot itself is rather a small rectangular area comprised of three smaller bands of light. The horizontal resolution is determined by the spin rate and ranges from 0.1° at 300 RPM to 0.5° at 1200 RPM. Table 4.1 lists some of the properties of the VLP-16 relevant for scanning and simulation purposes.

Table 4.1: VLP-16 LiDAR sensor specifications that are relevant for the simulation purposes (*Velodyne LiDAR: VLP-16 User Manual* 2019).

| **VLP-16 Specifications** | |
|---|---|
| Wavelength | 903 nm |
| Pulse length | 6 ns |
| Field of view (vertical) | 30°(+15°to -15°) |
| Field of view (horizontal) | 360° |
| Beam divergence (horizontal) | 3.0 mrad (0.18°) |
| Beam divergence (vertical) | 1.5 mrad (0.09°) |
| Range | 1 to 100 m |
| Typical accuracy | 3 cm |

Because the individual laser channels are stacked on top of each other, Velodyne provides a list of vertical corrections that can be applied to the scanned data in order to improve accuracy. These corrections bring the center of origin to the same value for all the laser channels. The vertical correction can be applied to $z$-coordinate of the returned sample (*Velodyne LiDAR: VLP-16 User Manual* 2019). The vertical correction values are also provided in Table 4.2.

## 4.2 Point Cloud Library

The Point Cloud Library (PCL) is a free library for processing n-D point clouds and 3D geometry with a BSD license. It is a collection

Table 4.2: Mapping between the individual lasers in the VLP-16 and the vertical angles they are pointing at in the laser array, along with the vertical correction (*Velodyne LiDAR: VLP-16 User Manual* 2019).

| Laser ID | Vertical angle | Vertical correction (mm) |
|---|---|---|
| 0 | -15° | 11.2 |
| 1 | 1° | -0.7 |
| 2 | -13° | 9.7 |
| 3 | 3° | -2.2 |
| 4 | -11° | 8.1 |
| 5 | 5° | -3.7 |
| 6 | -9° | 6.6 |
| 7 | 7° | -5.1 |
| 8 | -7° | 5.1 |
| 9 | 9° | -6.6 |
| 10 | -5° | 3.7 |
| 11 | 11° | -8.1 |
| 12 | -3° | 2.2 |
| 14 | -1° | 0.7 |
| 15 | 15° | -11.2 |

of algorithms that operate on point cloud data, including filtering, feature estimation, registration, surface reconstruction, model fitting, segmentation and others. PCL is a modern templated C++ library, built with performance and efficiency in mind. Under the hood, it uses several popular open source libraries, such as Eigen (Guennebaud, Jacob et al., 2010) for mathematical operations, FLANN (Muja and Lowe, 2009) for fast k-nearest neighbour search and Boost (Schäling, 2014) for efficient passing of data. PCL is built with robotics applications in mind and easily integrates with ROS (Robot Operating System) and comes with it's own PCD point cloud file format, now widely used. PCL also offers extensive visualization capabilities using the Visualization Toolkit (VTK) library (Rusu and Cousins, 2011). In this thesis, PCL was used in several of the scripts for processing point clouds, fitting planes, clustering points and other use-cases.

## 4.3 CloudCompare

CloudCompare is a point cloud processing software with a graphical user interface that makes it easy to experiment with various operations on point clouds. Apart from the usual collection of algorithms, it has been originally designed to perform comparison between two dense 3D points clouds, such as the ones acquired with a laser scanner (*CloudCompare* 2021). It is built on the Qt framework and OpenGL and offers fast manual exploration of point cloud data using the user interface. This was used throughout the entirety of the work on this thesis, along with its ability to be programmatically controlled from the command line.

## 4.4 Open3D

Open3D is an MIT-licensed open-source library for development of 3D data software, exposing both data structures and algorithms for easier processing of this type of data. It can be used from C++ or Python and is available for Linux, MacOS and Windows (Q.-Y. Zhou, Park and Koltun, 2018). Along with CloudCompare, this library was used in the various scripts and exploratory analysis of point clouds during writing of this thesis.

## 4.5 Indoor reconstruction++

In their master thesis, Sillerud and Johanssen (2019) created an acquisition system to collect high density point clouds in indoor environments. The system had three components, the VLP-16 rotating LiDAR, an inertial measurement unit (the Tinkerforge IMU Brick 2.0) and a simple camera tripod. The idea was to perform a vertical sweep using the tripod controls in order to acquire more data from the floor and ceiling area. They captured the data packets coming from the sensor using custom C++ code with the use of the VelodyneCapture library (Sugiura, 2022), and combined them using the rotation data from the IMU.

In order to create higher density point clouds, the scans had to be performed at multiple positions within an indoor environment and the individual scans had to be aligned and registered into a single point cloud. To estimate the translation between different positions, an approach called odometry was used. This involves continually scanning the environment as the sensor is moving between two positions, and then performing pairwise registrations between the acquired point clouds. Because the point clouds are always quite close together, the registration is fairly precise. Once each pair of point clouds are registered, the translation matrices between them are known, and can be composed into the final estimated translation matrix. This estimation is then optimized using a pose-graph and finally, used as an initial starting point for the mutual registration of the point clouds acquired from all the positions (Sillerud and Johanssen, 2019).

This was was then modified and extended by Hansen (2020) in *Indoor Reconstruction++*. Hansen extended the work of Sillerud and Johanssen (2019) in several important regards:

1. Translated the post-processing code from Python and Open3D to C++ and PCL, unifying the acquisition and post-processing code

2. Optimized the acquisition procedure

3. Moved away from using pose-graphs in lieu of incremental pairwise registration using more complex Iterative Closest Point (ICP) algorithms

Both works observed the sensor noise to be of roughly Gaussian distribution. Sillerud and Johanssen (2019) specified the measured error as two times the standard deviation of the normally distributed noise and Hansen found that using his acquisition system, the error is about 0.0313 m for the floor scans and 0.0172 m for the wall scans. Hansen's work is the starting point for the initial real-world noise exploration in this thesis.

## 4.6 BlenSor

BlenSor is a software package for emulating range scanning devices, integrated into the popular open source 3D content creation program Blender. It was created at the University of Salzburg and is targeted at the areas of obstacle detection, tracking, surface reconstruction and object segmentation. Its goal is to provide a standardized and unified simulation environment which replaces home-brewed solutions. This lets researchers focus on the algorithms rather than the surrounding framework. The package allows simulation of several types of range sensors with a strong focus on modelling their physical characteristics. This includes rotating LiDAR sensors, line LiDAR sensors and time-of-flight cameras. Its focus is mainly *offline* data creation rather than *realtime* processing, which means that BlenSor can simulate scenarios with a higher degree of detail than what was possible with existing robot simulators at the time of its release. However, it's fully tied into Blender's physics engine, making it easy to simulate scenes with motion (Gschwandtner et al., 2011).

BlenSor can be downloaded as a binary available for the Ubuntu Linux distribution and third party releases are also available for Windows and MacOS. Alternatively, for modifying and extending the package, the source code can be downloaded from GitHub.

One of the major benefits of BlenSor is its ease of use. The scanning controls are a part of the regular Blender user interface and can be controlled directly from the program. Figures 4.3 and 4.4 show the BlenSor interface in Blender's 3D viewport and user interface controls, respectively. The virtual sensor behaves like a camera object in Blender and its parameters can be changed in the same way. The resulting scans can be interacted with inside the scene or saved into files in standard point cloud formats, like `.pcd`. The entire simulation setup is packaged as a standard Blender `.blend` file. This ensures reproducibility and also scalability, as files can be distributed to multiple hosts or threads, which simulate sub-intervals in the scanning time interval. Unlike most simulators based on game engines, BlenSor does not utilize the graphics card.

The rich scripting capabilities of Blender using the Python programming language extend into BlenSor, where parts of the scanning logic can be easily modified. Prebuilt binaries of the package can be downloaded from BlenSor's website and the full source code is available on GitHub. Building the package directly from the source files allows for modifying the core parts of scanning code that interacts with Blender, written in C/C++. Just like Blender, BlenSor is distributed under the GPLv3 license. The

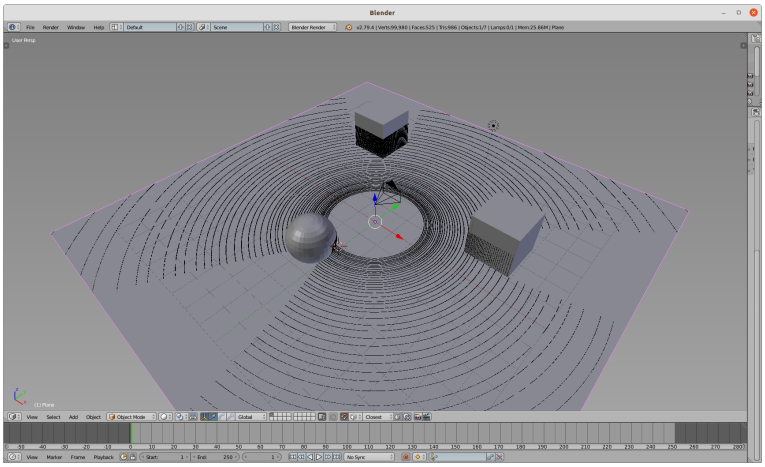


Figure 4.3: Scanned point cloud rendered within Blender's viewport. As shown, the point samples obtained from the scan can be optionally rendered inside the viewport.

ease of programmatic control and modification was heavily utilized in this thesis.

Similarly to other range scanning simulation packages, BlenSor utilizes ray-tracing to model the real sensor data acquisition process. It patches the Blender codebase to allow for casting several rays simultaneously for improved performance. Blender constructs a raytree and returns the distance for each hit along with an identifier for the object with which the ray collided. BlenSor then approximates light attenuation based on the distance and adds two error terms. One is the distance bias, which is constant per laser unit, and a per-ray noise term. Both of these are sampled from the Gaussian distribution. The error model is described in detail in Section 5.2. Parameters of these terms can be easily tuned. BlenSor comes with a fully implemented Velodyne HDL-64E S2 rotating LiDAR sensor. The properties of this model are used as a starting point for implementing a VLP-16 sensor model.

BlenSor is well suited for obtaining ground truth data as it returns a rich collection of data for each sampled point. The returns consist of 12 data points:

- timestamp of the measurement $t$
- yaw and pitch angles $\alpha$ and $\omega$
- measured distance $d$
- distance with added noise $d_{\text{noisy}}$
- clean $x$, $y$ and $z$ coordinates
- $x_{\text{noisy}}$, $y_{\text{noisy}}$ and $z_{\text{noisy}}$ coordinates with added noise
- *objectID* of the object that was hit

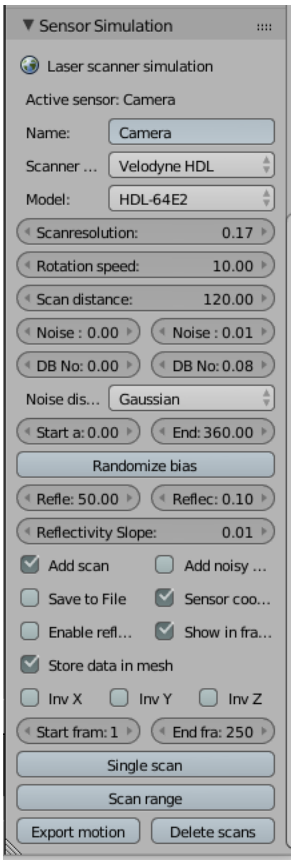



Figure 4.4: User interface controls for the scanning configuration. Users can select the scanner type, rotational speeds, as well as the noise properties and perform a single scan or a range of scans over a certain duration of frames. Equivalently, BlenSor can be controlled programmatically from Python.

## 4.7 PCC Arena

PCC Arena is a recently released open source benchmark platform for evaluating point cloud compression algorithms. The framework is built in Python 3 and takes individual point clouds from specified datasets and runs them through a set of compression algorithms. After optional pre-processing steps (which are dependent on the dataset and the algorithm tested), the point cloud is fed into an *encoder* which wraps the underlying compression algorithm. The encoder then saves the compressed file in a binary representation. Subsequently, a *decoder* performs the inverse operation, possibly with some post-processing steps. The processing steps might include sampling meshes, normalizing each point cloud to its maximal dimension along the three axes or computing the normals for a point cloud that does not have them.

Each tested compression algorithm has a set of *coding parameters* which are used to control the trade-off between bitrate and quality. The input and output point clouds and the compressed representation are then fed

into the *performance evaluator* to compute the various quantitative metrics. The platform is modular and easily extensible with new datasets, new compression algorithms (along with their coding parameters) or new performance metrics. *PCC Arena* was first published by Wu, Hsu, Kuo et al. (2020) and recently expanded in (Wu, Hsu, Hung et al., 2022).

Several metrics are used to evaluate the algorithms, and they can be split into 3 categories:

- **non-visual metrics**

  First, metrics such as *bitrate* (defined as the binary size over the number of points in the reference point cloud) and the *encoding* and *decoding* time are measured

- **3D coordinate-only visual metrics** These include the metrics defined in Section 2.4, like Asymmetric Chamfer Distance (Equation (2.3)), Chamfer Distance (Equation (2.4)), Hausdorff Distance (Equation (2.5)) and the Chamfer Distance Peak Signal-to-Noise Ratio, evaluated in both the point-to-point and point-to-plane variants. The latter are the reason for computing the normal information for each point cloud, as described in Section 2.4.4.

- **3D colored visual metrics**

  Two metrics focus on colors. *Luminance Color Peak Signal-to-Noise Ratio* measures the mean square error of luminance values across all pairs of nearest points, and *Viola et al.'s QoE* metric combines the Chamfer Distance with the L2 distance of histograms of luminance values between the two point clouds.

- **2D visual metrics**

  The point clouds are rendered into 2D images from several different views and image quality metrics such as *Peak Signal-to-Noise ratio* and *Structural Similarity* are computed.

To put the framework to use, the authors used PCC Arena to evaluate a number of current compression algorithms on several datasets, both mesh-based and point cloud-based. However, they only focused on object and avatar datasets, not scenes. These include mesh-based model datasets like *CAnonically Posed 3D Objects Dataset (CAPOD)*, *ModelNet40*, *ShapeNetCore* and *ShapeNetCoreColor* as well as two raw point cloud sequence datasets, *8i* and it's colored version *8iC*. The model-based datasets are converted into point clouds by means of a random sampling on the mesh, collecting the color and normal information with each point.

Apart from objective metrics, the authors conducted a user study on the rendered 2D images for perceived image quality. The compression algorithms considered were divided into two categories:

- **Signal processing–based**, specifically Draco, G-PCC and V-PCC.

- **Neural network–based**, specifically GeoCNNv1, GeoCNNv2, PCGCv1 and PCGCv2

For a detailed description of these algorithms, see Section 2.3 and Section 3.4.

Several findings are presented in Wu, Hsu, Hung et al. (2022). First, SP–based algorithms exhibited stable behaviour across different usage scenarios, and their differences mostly vanish at higher bitrates, when bandwidth is abundant. Second, NN–based algorithms have the potential to use lower bitrates with similar results to their SP-based counterparts. These can save up to 50% bandwidth, but their behaviour was very variable and at least 10 times slower. Third, NN–based algorithms sometimes produce artifacts and exhibit catastrophic quality drop. As part of the study, correlations between the objective and subjective metrics were also measured, but results proved inconclusive. The authors conclude that finding better Quality-of-Experience (QoE) metrics is important in future research. They also suggest further exploration of NN-based algorithms.

# Chapter 5

# Simulating VLP-16

The goal in this section is to create a system for simulating virtual room–scale LiDAR scans with a realistic error model. This necessitates studying the error characteristics of a real VLP-16 sensor. With these observations in mind, a virtual scanner with a realistic error model using the BlenSor tool is created.

BlenSor was chosen for this implementation because of its ease of use and extensibility. At the same time, its default error model is, while simplistic, a good first order approximation.

## 5.1 Collecting error measurements from the real VLP-16

As mentioned in Section 3.1, existing efforts in the LiDAR simulation area often work without any artificial noise at all. When added, it is most often simply sampled from a Gaussian distribution. However, it has been observed that certain factors can influence this distribution (Kidd, 2017; Glennie, Kusari and Facchin, 2016). With these considerations in mind, this section will work with three hypotheses:

1. the aggregated noise distribution is approximately Gaussian
2. the incidence angle[1] on the surface affects the noise
3. different laser channels have different noise characteristics

In order to effectively model the error distribution in synthetic scans, a scanning procedure was followed with the real VLP-16 sensor in an indoor environment. The goal of this procedure was to collect a sample of the real error distribution in a situation similar to the majority of indoor scanning applications—regular rectangular rooms with a ceiling, floor and walls, with relatively short scanning distances, up to 5 meters.

[1]The incidence angle is the angle between a ray hitting a particular surface and the normal vector of that surface. Its value is in the interval $(0, 90]°$.

### 5.1.1 Implementation

In order to collect information about the performance of the individual laser channels in the stack, the following modifications were done to the code for the scanning procedure taken from the work of Hansen (2020):

1. the core point cloud class from the PCL library, `pcl::PointXYZ`, was switched to `pcl::PointXYZL` so that a label value can be stored along each point's coordinates

2. A vertical correction, as stated in the VLP-16 manual, was added to the $z$-coordinate of each return

3. During the scanning procedure, each return sets the label value to the id of the laser channel it originated from

Having the label data, each point can be mapped to the specific laser channel, which can be used to test hypothesis 3. For the purposes of visualization, individual lasers channel identifiers were translated into values on the RGB color spectrum as can be seen on Figures 5.2 and 5.3. The recorded point clouds were saved as `pcd` files.

### 5.1.2 Procedure

The sensor was placed at distances from 1 to 5 meters away from a vertical wall (with a 1 meter interval) and a densified scan was performed using the sweep acquisition technique developed by Sillerud and Johanssen and improved by Hansen. Such a sweep is depicted in Figure 5.1. The scan

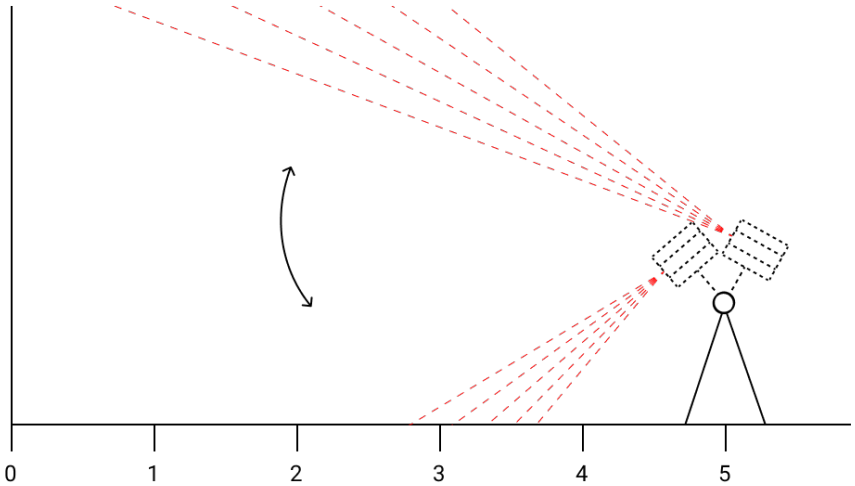



Figure 5.1: Diagram of a single sweep. The sweep was repeated at 1 to 5 meters distance from the wall, in 1 meter increments. For illustration purposes, only 5 rays (out of 16 in total) are drawn.

collected a 180°-wide area in front of the sensor. Figures 5.2 and 5.3 show examples of the full extent of the captured point clouds. Details of the data acquisition system are described in Section 4.5 and in the respective master theses, see Hansen (2020) and Sillerud and Johanssen (2019).

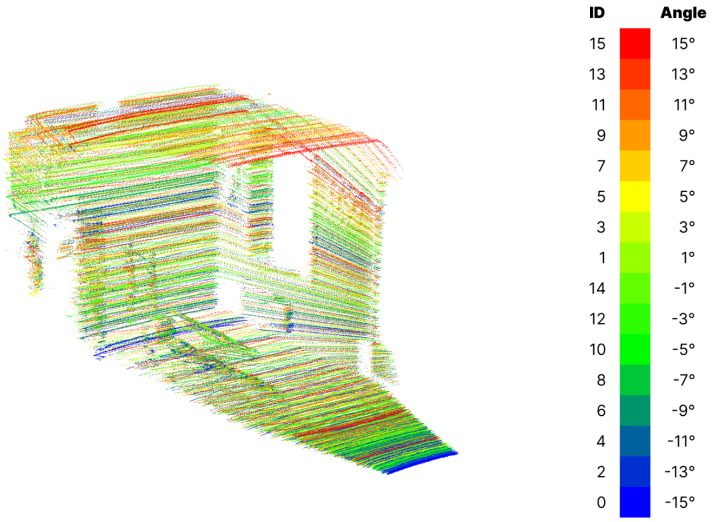



Figure 5.2: Full scan of the environment at 5 meter distance from the wall. The areas in the ceiling and side walls are subsequently removed. Each point is colored based on the laser channel that was used to obtain it.

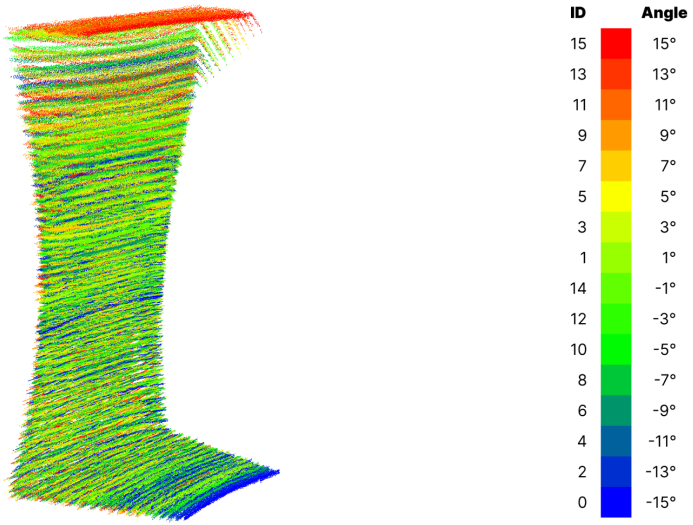



Figure 5.3: Full scan of the environment at 1 meter distance from the wall. The areas in the ceiling are subsequently removed. Each point is colored based on the laser channel that was used to obtain it.

The densified scans were manually cleaned up and cropped so that they consist only of points lying on the vertical wall or the horizontal floor, per each distance measurement. This process was done using the

`get_plane_distances` helper program, written in C++ using the `Open3D` library. 3 scans were performed at each distance, giving altogether 15 scans of the floor and 15 scans of the wall. The reason for this split is to test the differences in performance with high incidence angles, which are present more often in samples returned from the floor, as opposed to the lower incidence angles, present in the samples returned from vertical walls facing the sensor.

The noise distribution was then collected from the cropped scans by first fitting a plane to the point cloud using the RANSAC algorithm and then calculating the signed distance from each sample to the plane. This signed distance can be called the *plane residual*. Each value has the corresponding laser channel id associated with it. The procedure is summarized in Procedure 1.

```
for distance d in 1...5 meters do
    for run r in 1...3 at distance d do
        perform sweep scan;
        crop to remove everything but the wall;
        crop to remove everything but the floor;
        for each kind in [wall, floor] do
            fit a plane using the RANSAC algorithm;
            for each point in point cloud do
                get the signed distance between the point and the
                  plane;
                save the distance along with the laser id to a file;
            end
        end
    end
end
```

**Procedure 1:** Collecting error data

### 5.1.3 Results and discussion

#### Visual observations

Figure 5.4 shows the point cloud for the wall at five meters' distance. We can discern that the wall scan seems to have both uniform density and a uniform distribution of samples and also of the laser channel identifiers within the points. On the other hand, the floor scan on Figure 5.5 shows a decreasing density along the horizontal plane away from the sensor location.

This likely stems from the fact that the further the horizontal plane extends, the more acute the incidence angle becomes and the likelihood of a successful return diminishes. At the same time, with roughly linear acceleration of the vertical sweep, the same number of returns has to be spread over a larger area the farther the return is.

The laser identifier distribution is also skewed with a bias towards the

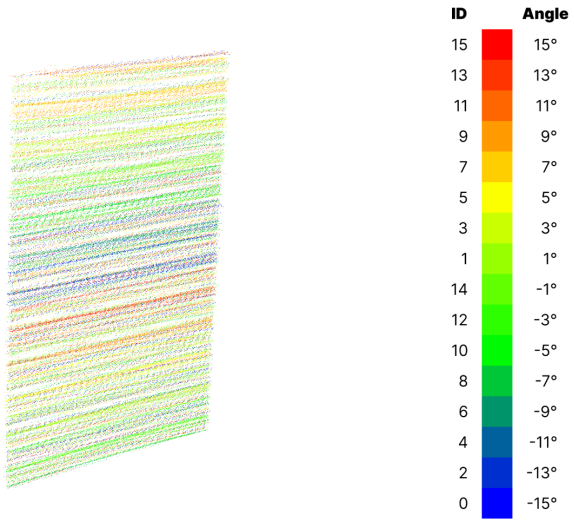



Figure 5.4: Cropped scan of the environment at 5 meter distance from the wall. Each point is colored based on the laser channel that was used to obtain it.

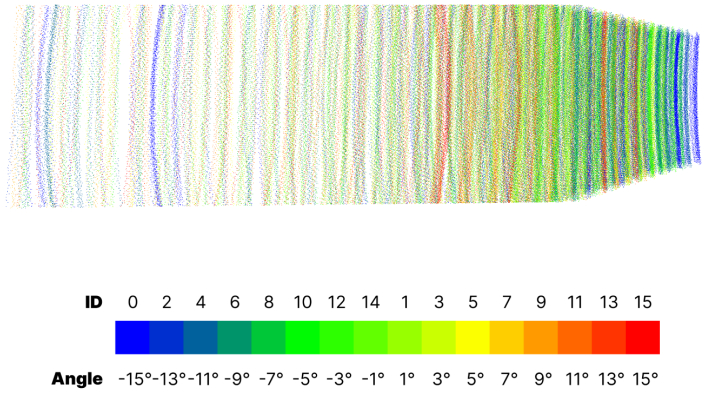



Figure 5.5: Top-down view of a cropped scan of the environment at 5 meter distance from the floor. The sensor was placed on the right, facing left. Each point is colored based on the laser channel that was used to obtain it.

lower units in the laser stack as we go near the sensor (bluer tones). This stems from the fact that in the lower extreme of the sweep, only the lasers channels at the bottom part of the sensor reach the closest parts of the floor.

Another interesting feature is shown when looking at the scanned plane from the side. Figure 5.6 shows that the points from each laser are distributed in clusters along the horizontal axis. This can be seen better when isolating only one of the lasers, seen on Figure 5.7.

This is also apparent in the floor scan (Figures 5.8 and 5.9), which

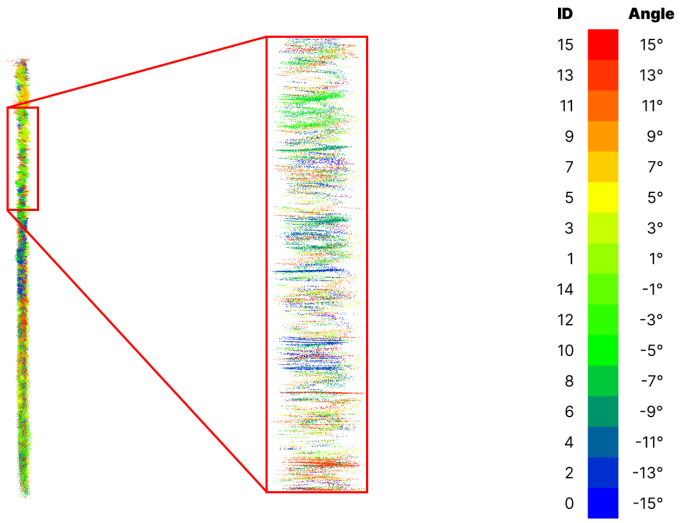



Figure 5.6: Side view of the wall scan at 5 meters distance. Each point is colored based on the laser channel that was used to obtain it.

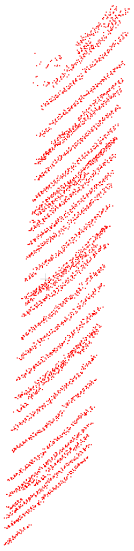


Figure 5.7: Three-quarter view of the isolated samples obtained by one of the 16 laser channels when performing a scan of the wall at 5 meters distance.

especially highlight the fact that the clusters are spread along the direction of the ray at a particular point in time of the sweep.

### Noise distribution

The plane residuals data was computed and aggregated to obtain the noise distribution statistics, as explained in Section 5.1.2. Figure 5.10 shows the

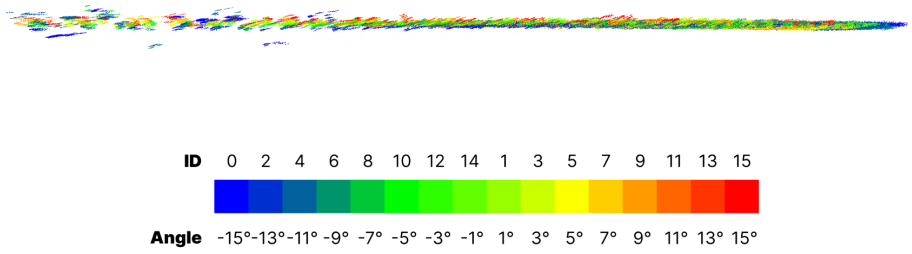



Figure 5.8: Side view of the floor scan. The sensor was placed on the right, facing left.

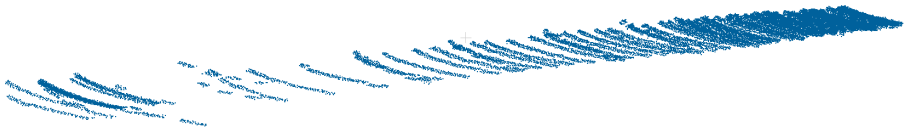


Figure 5.9: Three-quarter view of the isolated samples obtained by one of the 16 laser channels when performing a scan of the floor at 5 meters distance.

violin plots of the noise distribution of all measurements across all sixteen lasers and at all the distances, data taken for the wall scans and the floor scans. It is apparent that the error is close to a Gaussian distribution. The

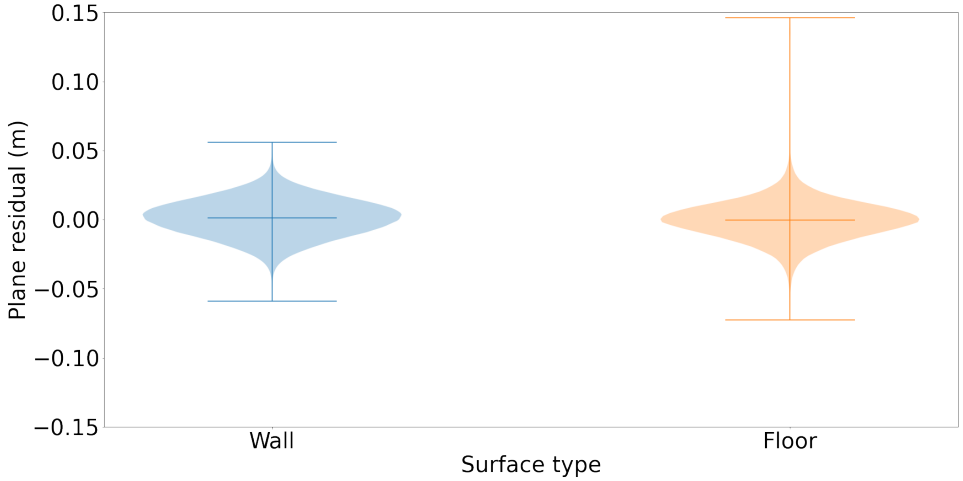



Figure 5.10: Noise distribution comparison between all samples obtained from the wall scans and the floor scans.

recorded mean was equal to 0.00 m with the standard deviation of 0.014 m

(rounding to three decimal places) for both surface types, even though the floor scans have higher extremes.

In the work of Sillerud and Johanssen (2019), the definition of error was specified to be two times the standard deviation. Compared to the work of Hansen, 2020, described in more detail in Section 4.5, the numbers suggest a higher measured error for the wall scans ($2 * 0.014 = 0.028$ m vs. 0.0172 m) but roughly similar error for the floor scans ($2 * 0.014 = 0.028$ m vs. 0.0313 m).

### Noise at different distances

In Figures 5.11 and 5.12 we can see the noise distributions at different distances. While the wall scans distributions are mostly unchanged, it

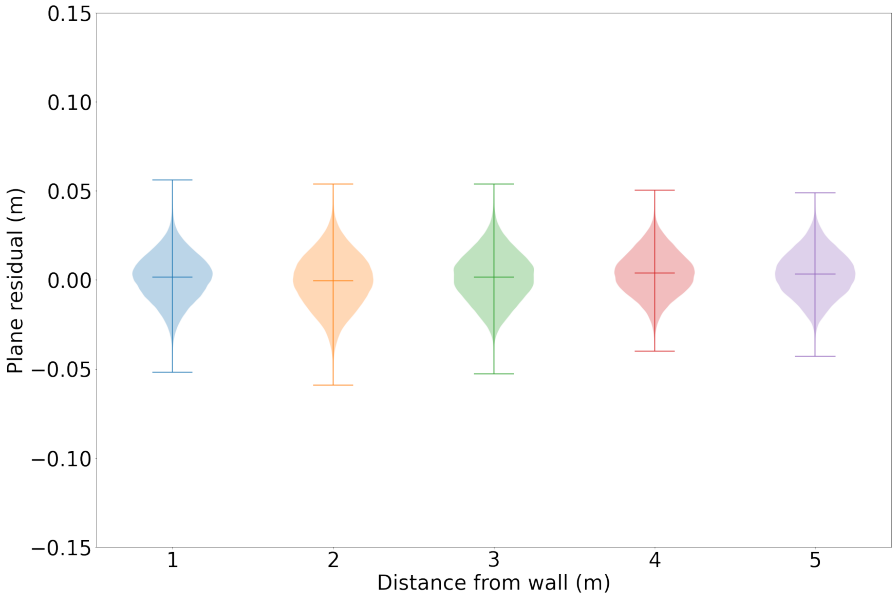



Figure 5.11: Noise distribution per sensor distance from the wall. Showing data obtained from the wall scans.

is apparent that the floor scans have a higher variability in how skewed the distribution is, and also what the extremes are. These two facts suggest that, at least between the distances of 1 to 5 meters, distance alone does not affect the noise distribution. It is likely that the higher incidence angle in the floor scans creates more variability.

### Noise of different lasers channels

Noticeably different performance of some of the lasers was not observed when collecting data. Figures 5.13 and 5.14 show that the performance varies only slightly. This was surprising as the work of Glennie, Kusari and Facchin (2016) suggests that laser channels with ID 1, 3 and 5 consistently perform slightly worse than the other laser channels.

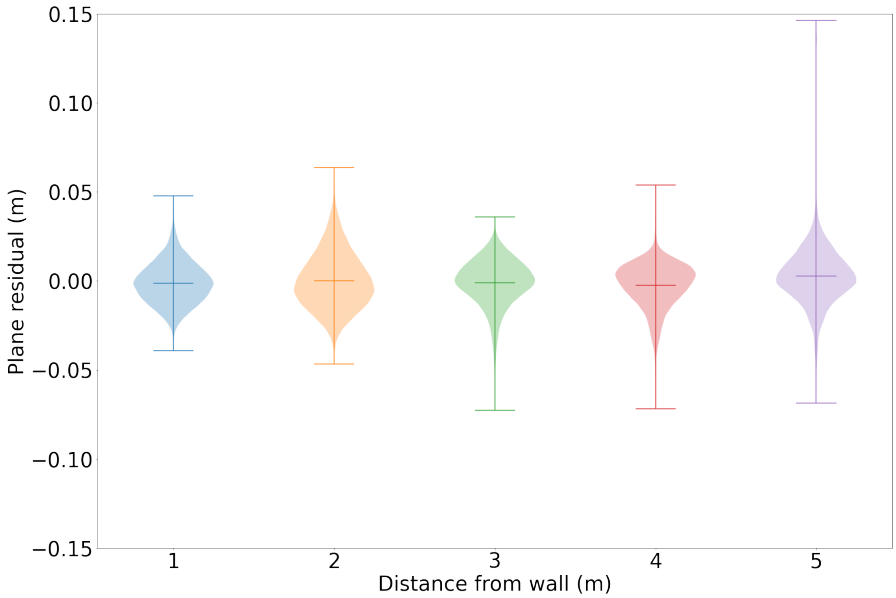



Figure 5.12: Noise distribution per sensor distance from the wall. Showing data obtained from the floor scans.

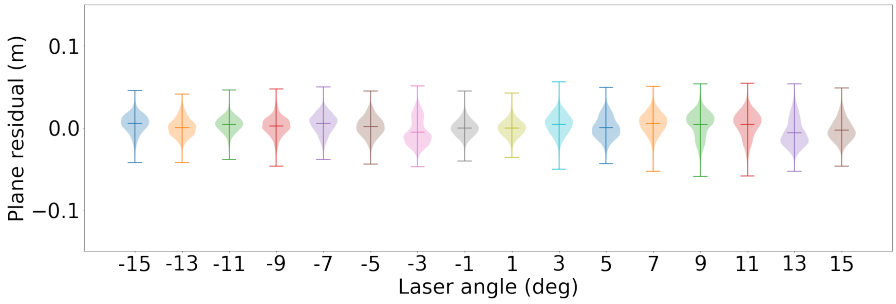



Figure 5.13: Noise distribution per angle of a laser channel. Showing data obtained from the floor scans.

### Noise and incidence angle

In order to observe whether the incidence angle affects the noise distribution, the angle had to be computed for each sample. This is possible because the position of the sensor in 3D space is known, as well as the $x$, $y$ and $z$ coordinates of each point. Finally, in the case of our error measurements, the important piece of information that is available to us is whether the point is lying on a vertical wall or horizontal floor. This means that the surface normal vectors of the returns can be used.

Based on this data, we can make a simple calculation to get the incident angle:

$$\theta = \arccos\left(\mathbf{n}_{\text{surface}} \cdot \mathbf{v}_{\text{ray}}\right), \tag{5.1}$$

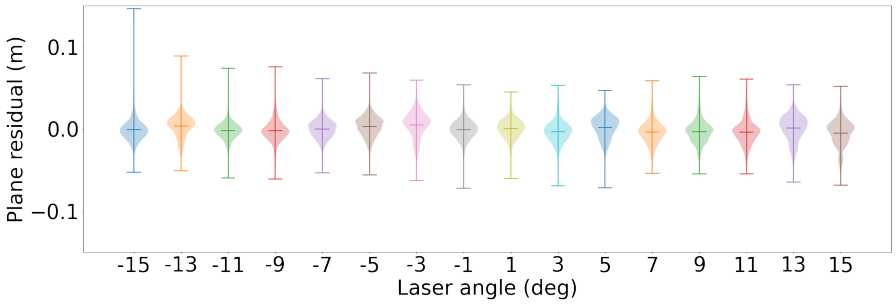



Figure 5.14: Noise distribution per angle of a laser channel. Showing data obtained from the floor scans.

where $\mathbf{n}_{\text{surface}} = [0, 1, 0]$ when it's on the wall and $\mathbf{n}_{\text{surface}} = [0, 0, 1]$ when on the floor, respectively. The vector of the laser ray emitted from the sensor, $\mathbf{v}_{\text{ray}}$, is given by

$$\mathbf{v}_{\text{ray}} = \mathbf{p}_{\text{sample}} - \mathbf{p}_{\text{sensor}}, \tag{5.2}$$

where $\mathbf{p}_{\text{sample}}$ is the position of the sample and $\mathbf{p}_{\text{sensor}}$ is the position of the scanner in world coordinates.

The incidence angle was computed for a sample of the points and plotted as seen on Figure 5.15. It is apparent that the spread of values

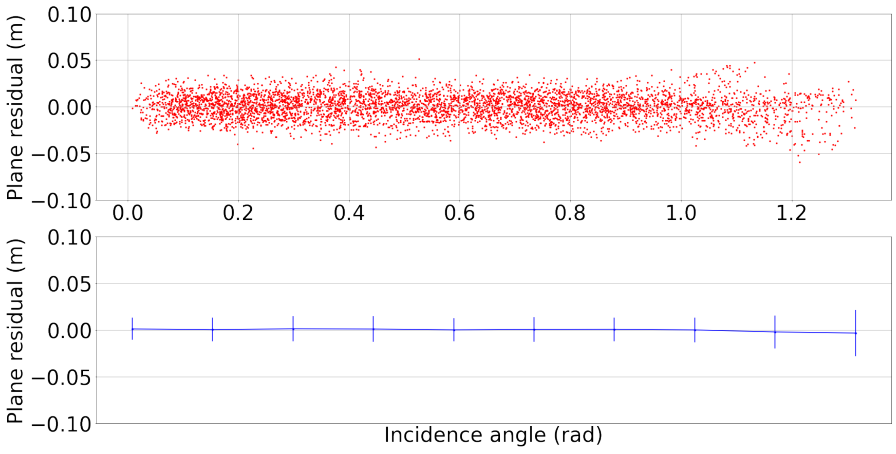



Figure 5.15: The spread of plane residuals increases with incidence angle (sample size $N = 30000$). The second plot shows the means and standard deviations calculated from the data split into 10 bins.

increases with increasing angle of incidence. At the same time, the number of points decreases as well. This suggests that a high incidence angle both diminishes the probability of a successful return and that the noise increases.

## 5.2 Base error model in BlenSor

The latest version of BlenSor includes simulation of the Velodyne HDL-32E and HDL-64E sensors, which share the same error model. The core ray tracing approach used in all the sensor simulations implemented in BlenSor (rotating LiDAR, line LiDAR and ToF cameras) is the same and it is explained below. However, the way the rays are cast differs between the sensor types, as well as their error model. In this section, we focus on the specifics of the sensor model for rotating LiDAR. The model consists of four parts, which will be introduced in the following section.

#### Ray casting approach

BlenSor uses a relatively simple approach to model a rotating LiDAR casting rays. Each individual scan has a defined `start_angle`, `end_angle` and `angle_resolution`. These values are used to calculate the number of vertical lines that the laser will emit rays at. Afterwards, for each line and each laser, a ray is created with the correct pitch and yaw angle, along with the timestamp. These rays are then sent to the scanning part of the code.

#### Modelling light attenuation

As light travels through a medium, its intensity gets reduced. This reduction is proportional to both the distance travelled and the properties of the medium. As we only consider applications where the scanner is used in air, we disregard these parameters, though different mediums could be modelled. A reduction in light intensity means that the sensor has less probability that a given return is recorded. As an example, the Velodyne HDL-64E S2 sensor can detect objects with minimal diffuse reflectivity of 10% at 50 meters and 80% at 120 meters (Gschwandtner et al., 2011). The decrease in reflected light is compensated in the real sensor by lowering its threshold, however the relationship is not specified. BlenSor chooses to implement a linear interpolation based on the given values for each ray:

- objects closer than 50 meters are detected if their reflectivity is above 0% (this is the reflectivity distance parameter seen in Table 5.1)
- objects between 50 and 120 meters are detected if their reflectivity $r$ is above $r_{\text{min}}$, where

$$r_{\text{min}}(dist) = w_{\text{lower}} + \frac{(r_{\text{upper}} - r_{\text{lower}}) \times dist}{d_{\text{upper}} - d_{\text{lower}}}, \tag{5.3}$$

where $dist$ is the distance between the sensor and the potentially returned sample, $r_{\text{upper}}$ and $r_{\text{lower}}$ are the maximum and minimum reflectivity values. $d_{\text{upper}}$ and $d_{\text{lower}}$ denote the maximum and minimum distance for which this particular calculation is used.

#### Distance bias

As stated by the manufacturer and observed by the authors of BlenSor, the $z$-distance of a plane normal to the laser's $z$-axis may differ up to 12 cm for any two lasers (Gschwandtner et al., 2011, p. 6). In other words, the distance between the laser emitter and the same recorded surface for two different lasers can vary. Therefore a per-laser distance bias term is added which remains the same for each rotation, sampled from the Gaussian distribution. This distance bias somewhat models the clusters of samples coming from individual laser channels that is visible in the real data in Figures 5.7 to 5.9.

#### Per-ray noise

On top of the distance bias, each ray has an added error term that represents the light beam divergence. This one is also sampled from the Gaussian distribution. The resulting distance noise from the two terms is stated as

$$dist_{\text{noisy}}(\text{yaw}, \text{pitch}) = dist_{\text{real}}(\text{yaw}, \text{pitch}) + \epsilon_{\text{bias},i} + \epsilon_{\text{ray}}, \tag{5.4}$$

where

$$\epsilon_{\text{bias},i} \sim \mathcal{N}(0, \omega_{\text{bias}}), \tag{5.5}$$
$$\epsilon_{\text{ray}} \sim \mathcal{N}(0, \omega_{\text{ray}}), \tag{5.6}$$

and $dist_{\text{real}}(\text{yaw}, \text{pitch})$ is the precise distance between the sensor and the sample. This value is computed from the yaw and pitch values of the sensor. $\omega_{bias}$ is the customizable parameter that specifies the standard deviation of the Gaussian noise distribution for the distance bias and $\omega_{\text{ray}}$ is similarly a customizable parameter that specifies the standard deviation of the Gaussian distribution for the per–ray noise.

For example, in case of the most developed virtual LiDAR in BlenSor, the Velodyne HDL-64E, the value of $\omega_{bias}$ is set to 0.078 m and the value of $\omega_{\text{ray}}$ is set to 0.01 m. The distance bias for each laser channel can be manually randomized and saved for reproducibility, or resampled between every scan.

The values for the noise Gaussian distribution for the HDL-64E sensor seem to be too high, which is also apparent in Figure 5.16b when compared to the clouds from the real sensor (Figure 5.2).

## 5.3 Implementing the virtual VLP-16 sensor

The already–implemented virtual Velodyne HDL-32E and 64 are bigger and more complex sensors with 32 and 64 lasers respectively. The first step was therefore to implement a simulation of our VLP-16 sensor. The implementation of these virtual sensors is situated in the `blendodyne.py` source file. The specifics of various types of sensors are handled in respective Python files, whereas the core of the raycasting functionality

is implemented in the underlying C++ code and is common between the virtual rotating LiDARs, line LiDARs and Time-of-flight cameras.

Parameters for the VLP-16 sensor were added according to the manual from the manufacturer and in line with the existing sensor types. This included a list of the vertical angles corresponding to the 16 lasers as seen in Table 4.2 and setting the angle resolution and rotation speed. After that, it was necessary to add the VLP-16 model to an enumeration to ensure it's possible to select it in Blender's user interface. The initial values for the distance bias and per-ray noise were taken from the existing virtual HDL64E sensor parameters. The parameters are explained in Section 5.2.

Table 5.1: Parameters used for VLP-16 in BlenSor.

| **VLP-16 Parameters in BlenSor** | |
|---|---|
| Rotation speed | 5 Hz |
| Max distance | 100 m |
| Distance bias mean | 0.0 m |
| Distance bias sigma | 0.078 m |
| Per-ray noise mean | 0.0 m |
| Per-ray noise sigma | 0.014 m |
| Reflectivity distance | 50 m |

The scanning procedure itself was setup using Blender's Python API. First, a mesh for the floor or wall is placed into the scene. Then the sensor itself is created as an object of type Camera, and placed at the correct distance. The sweep was described using Blender's animation system, where at certain keyframes, the rotation angle of the sensor was set to specific increments. Blender then interpolates a smooth bezier curve transition between the keyframes. The virtual scanning sweep was set to start at $-60°$ below the angle when looking straight ahead, and the end was set to $60°$ from this angle. The sweep was set to take 4 seconds at 120 frames per second. The procedure can be found in the `scan_for_error.py` script.

## 5.4 Evaluation setup

At the start of each scan, a collection of ray objects is prepared. Their vertical and horizontal angles are then determined based on the laser angles and the timestamps. Each return contains the $x$, $y$ and $z$ coordinate as well as the distance between the point and the sensor. The coordinates are in the reference frame of the sensor.

This means that there are two points in the process where noise can be added. First, in the preparation phase where the angles of the rays are computed, and second, in the collection phase, where the final $x$, $y$ and $z$ coordinates are computed. Because the virtual scanner model sets all the ray origins at 0, the vertical correction applied when taking real scans is not necessary.

Five different error models were tested:

1. No noise
2. Default BlenSor noise from HDL-64E
3. Default BlenSor noise with VLP-16 values
4. Modelling beam divergence
5. Incidence angle effect

#### No noise

First, virtual scans without any noise added were performed. As the default BlenSor model is our starting point, this meant setting the distance bias mean and standard deviation to 0.0 m, as well as the per-ray noise mean and standard deviation to 0.0 m. An example of a wall scan is shown on Figure 5.16a and an example of a floor scan is shown on Figure 5.17a.

#### Base BlenSor error model with default HDL-64E parameters

As a base noise level for comparison, the noise parameters from the existing Velodyne HDL-64E were used. These include setting $\omega_{\text{bias}}$ to 0.078 m and $\omega_{ray}$ to 0.01 m. Both mean values are set to 0.0 m. An example of a wall scan is shown on Figure 5.16b and an example of a floor scan is shown on Figure 5.17b.

#### Base BlenSor error model with VLP-16 parameters

Results from the previous error model and the data collected in Section 5.1.3 then informed the values set to more closely match the VLP-16 performance. The $\omega_{\text{bias}}$ was set to 0.014 m and $\omega_{\text{ray}}$ kept at 0.01 m. Both mean values were left unchanged at 0.0 m. An example of a wall scan is shown on Figure 5.16c and an example of a floor scan is shown on Figure 5.17c.

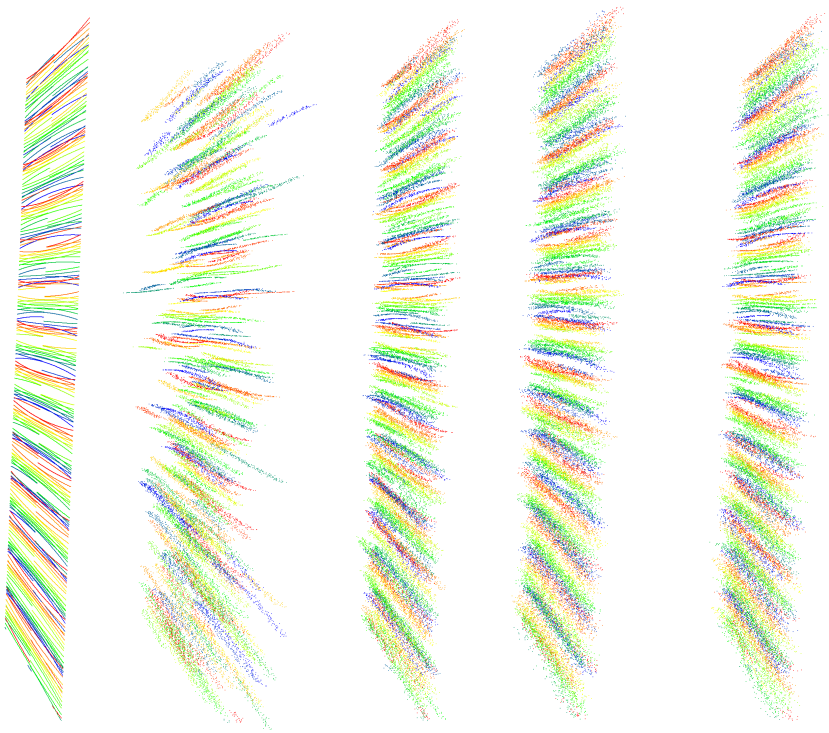
#### Adding beam divergence

Velodyne VLP-16 has a stated horizontal beam divergence of 3.0 mrad (0.18°) and vertical beam divergence of 1.5 mrad (0.08°). This was modeled by adding a small amount of noise to the rotation angles of the cast rays in the preparation step. The noise was uniformly sampled using these divergence values. Because the divergence noise has to be added in the ray preparation phase, there is no way to retrieve non-noisy coordinates from the sensor returns. An example of a wall scan is shown on Figure 5.16d and an example of a floor scan is shown on Figure 5.17d.

#### Incidence angle

Finally, the incidence angle information was used to affect the noise distribution. The principle used was to increase the distribution variance

as the incidence angle of a particular sample increases. Tuning this relationship is described in the next section. An example of a wall scan is shown on Figure 5.16e and an example of a floor scan is shown on Figure 5.17e.



(a) None (b) Default (c) VLP-16 (d) Divergence (e) Incidence

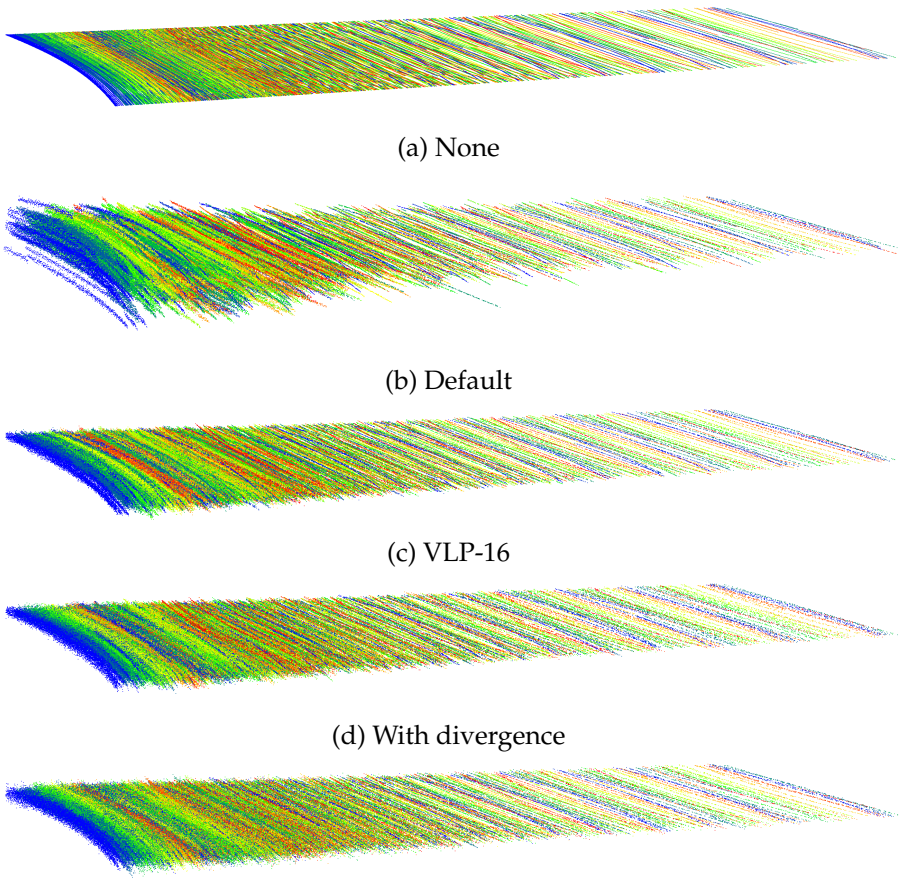
Figure 5.16: Comparison of the different error models at 5 meter distance. Scans of the wall shown.

## 5.5 Results and discussion

Looking at the comparison in Figure 5.18, it as apparent that the default values for the HDL-64E sensor are too high to effectively model the distribution for the VLP-16. Decreasing them for the VLP-16 error model brings the distribution much closer to the real one. The HDL-64E values can be discounted for our error model.

It is also apparent that modelling the beam divergence provides almost no discernible difference. This could stem from the fact that the effect will not be pronounced enough in distances lower or equal to 5 meters. To test this hypothesis, a virtual scan with a higher distance was performed. The results in Figure 5.19, where the virtual sensor was place 30 meters away from the wall, however suggest that this effect is not noticeable even at these distances.

In order to test whether taking the incidence angle into account is necessary, in Figure 5.20, a sample of the points was plotted for all the



(a) None

(b) Default

(c) VLP-16

(d) With divergence

(e) Incidence angle

Figure 5.17: Comparison of the different models at 5 meter distance. Scans of the floor shown with the sensor pointing left to right.

relevant error models and compared to the real data.

With regards to incidence angle affecting the noise, it is important to remark that our error metric—the plane residuals—does not take into account the incidence angle, only the distance between the ideal plane and the position of the particular sample. This creates a relationship where a point that was produced by a ray with high incidence with a high distance error actually has a lower planar residual than a point with low incidence and a low distance error. The relationship is visualized in Figure 5.21.

This suggests that it is not possible to model this behaviour only by increasing the variance of the noise as the incidence angle increases. An interesting approach would be to try and add noise to the resulting $x$, $y$ and $z$ coordinates of the returned sample instead, or even add noise specifically in the direction of the normal of the plane.

For the purposes of our simulation, the VLP-16 with the added effect of beam divergence was chosen as an appropriate error model. For more detailed analysis and calibration of the VLP-16, see the work of Kidd (2017) and Glennie, Kusari and Facchin (2016).

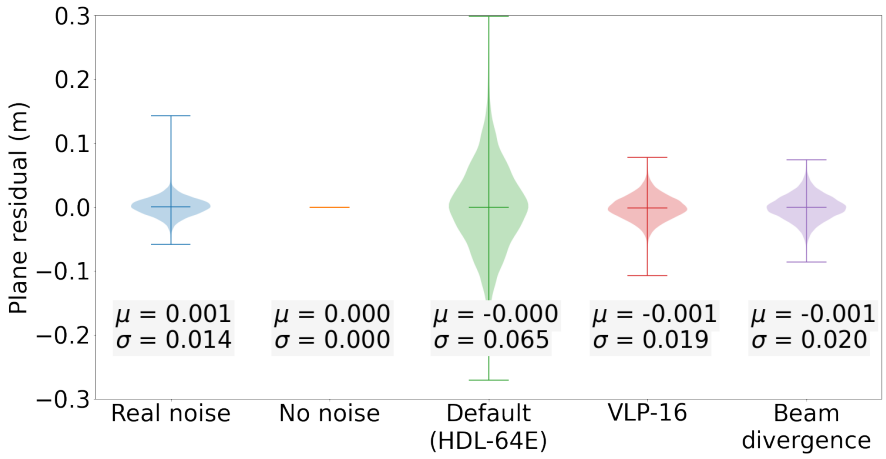



Figure 5.18: Different noise distributions for the error models (sample size $N = 100000$)

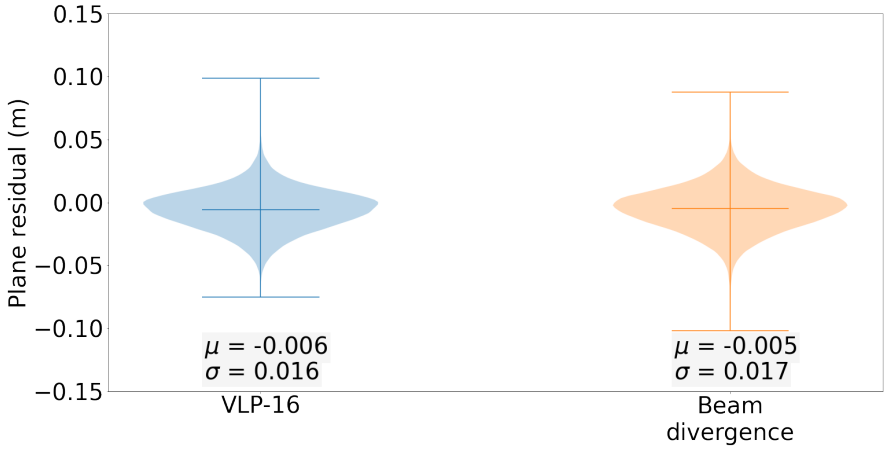



Figure 5.19: Comparison of the VLP-16 noise parameters without and with the beam divergence noise added at the distance of 30 meters. The effect is negligible.

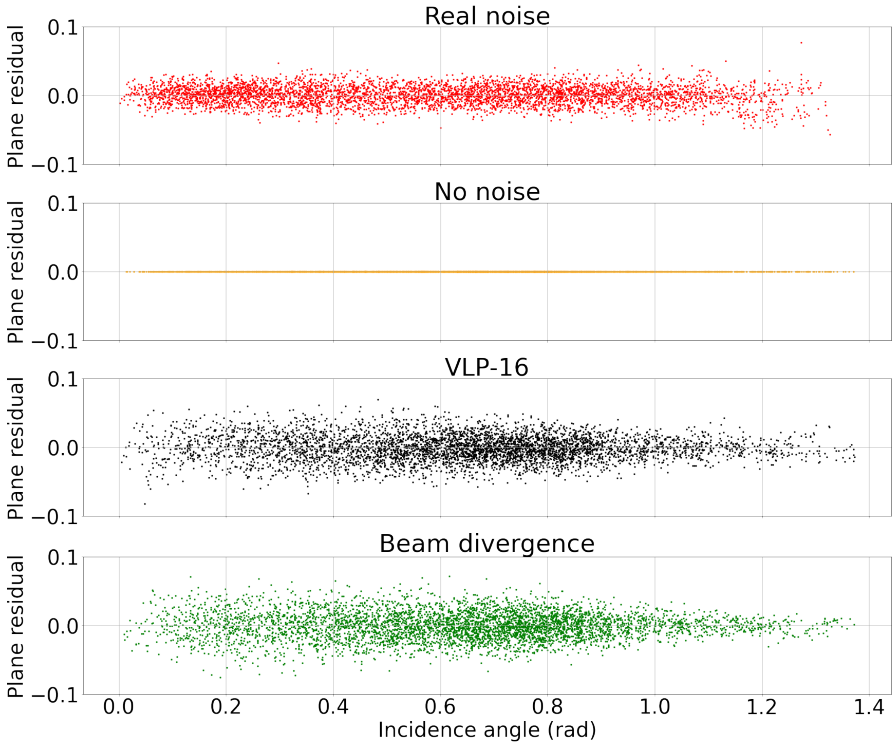



Figure 5.20: The relationship between the incidence angle and the planar residuals for the various error models. Only the real data show a higher variance as the incidence angle increases.

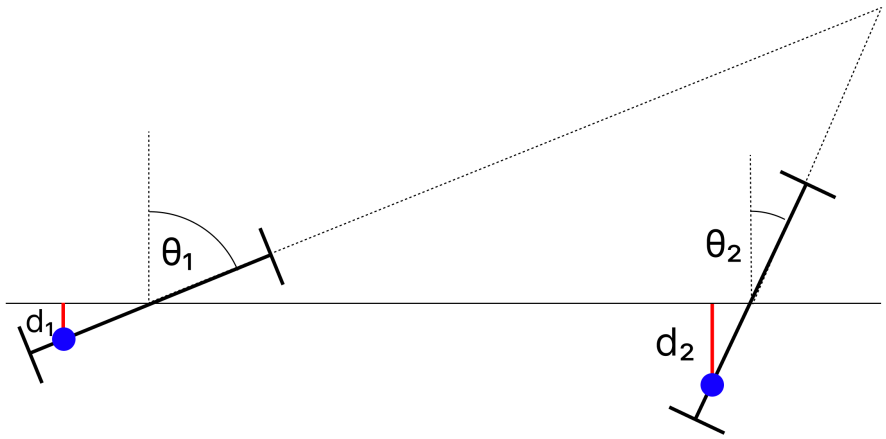



Figure 5.21: Relationship between incidence angle and noise for two samples from the floor. With an identical distance bias, the planar residual (shown in red) is lower for the noisy sample (in blue) with the higher incidence angle, even though it is farther from the true sample.

# Chapter 6

# Compressing point clouds

An important step in creating a point cloud compression (PCC) algorithm based on geometry fitting is to evaluate the performance of existing compression approaches. Point cloud compression was described in more detail in Section 2.3 and some of the existing algorithms explored in Section 3.4. Section 4.7 described *PCC Arena*, a framework for comparing the performance of point cloud compression algorithms, whether they are classical (signal processing–based) or neural network–based. However, the project has focused on datasets of objects or avatars, with highly dense and noise-free point clouds. In this part of the thesis, PCC Arena is extended with the focus on room-scale, outward looking scans with noise and imperfections. As such, performance of existing compression algorithms on noisy indoor scan data is evaluated. The authors of PCC Arena observe that the lower quality of the non-avatar objects in their results shows that the current PCC algorithms may not be general enough for arbitrary object classes (Wu, Hsu, Hung et al., 2022).

It is therefore useful to test the existing point cloud compression algorithms on different types of scenes, in this case specifically indoor environments. Room-scale, outward-looking scans present a very different challenge to the compression approaches as they are often larger, more variable in dimensions, noisier, and may include extreme outliers.

In order to test the existing compression algorithms on indoor scene point clouds, the *Stanford Large-Scale 3D Indoor Spaces Dataset (S3DIS)* (Armeni, Sener et al., 2016; Armeni, Sax et al., 2017) was used as baseline for ground truth. Described in Section 3.3, the dataset presents a good benchmark for indoor environments as it's created from real-world scenes filled with objects and clutter. Because the dataset also has a mesh version of all the scenes, it was possible to use BlenSor with the VLP-16 error model created in Chapter 5 as a virtual scanner in these environments, in order to test how occlusions and a realistic LiDAR error model affects the compression performance. Scans were performed both with the added noise and without any noise. The reason for this was to compare both how the noise itself might influence the results, as well as how effects like occlusion, non-uniform point distribution and the presence of outliers might influence the results, even without noise present. For further

comparison, the *CAPOD* dataset was also used in this part of the project. *CAPOD* is an object–based mesh dataset, also used by Wu, Hsu, Hung et al. (2022), containing 180 meshes of generic objects uniformly distributed into 15 classes.

The hypothesis was that the compression algorithms will perform worse in the case of noisy indoor environments, compared to the inward–looking object datasets. Furthermore, the neural network–based algorithms that have been trained on object datasets, are expected to perform comparatively worse than their traditional counterparts. Much like what is seen in the object datasets, a higher variance in their performance is also expected.

## 6.1 Implementation

To compare the existing compression algorithms on indoor scene datasets and especially their performance on noisy data, the following procedure was performed:

1. perform the PCC Arena tests on the CAPOD object dataset with a subset of the existing point cloud compression algorithms
2. perform the PCC Arena tests on the raw point clouds from the Stanford Large-Scale 3D Indoor Spaces Dataset
3. create virtual scans of the room meshes from the Stanford 2D-3D-Semantics Dataset using the VLP-16 scanner in BlenSor, once without noise and once with the VLP-16 error model noise
4. perform the PCC Arena tests on the point clouds generated from the previous step
5. evaluate and compare the results

### 6.1.1 Preparing the *CAPOD* dataset

The *CAPOD* dataset introduced by Papadakis (2014) was chosen as a benchmark because it is a relatively general purpose object dataset. It is also often used in the *PCC Arena* paper as a sample when showing the various performance metrics of the PCC algorithms. Finally, neither of the NN–based algorithms have been trained on it.

CAPOD is a mesh-based dataset with meshes stored in `.obj` files. Before using it in PCC Arena, the meshes had to be converted into point clouds by the means of random sampling. This was achieved with one of the utility scripts that come with PCC Arena. The `ds_mesh2pc.py` script uses CloudCompare under the hood (described in Section 4.3) to perform a random sampling of the mesh into 500 000 points. As mentioned in Section 4.7, normal information is necessary to compute the point-to-plane metrics. These are generated as part of the same script, also using CloudCompare under the hood. Finally, the point clouds are

proportionately scaled to the interval $[0, 1024]$ using their longest axis. This is done in accordance to the process in the original paper, in order to be able to fairly compare results across the different datasets. An example of a point cloud extracted from the CAPOD dataset is shown on Figure 6.1.

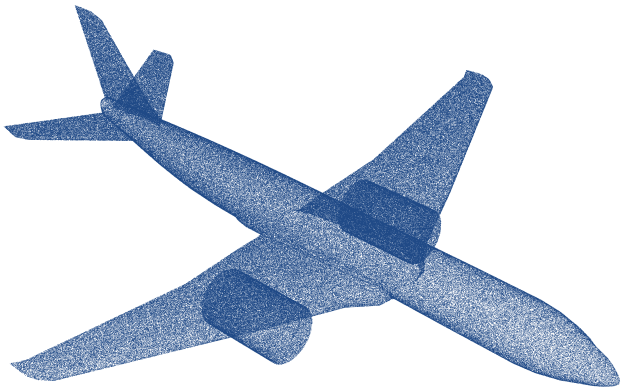


Figure 6.1: An example of a point cloud from the CAPOD dataset. The point cloud was created by sampling a simple mesh model of a plane.

### 6.1.2 Preparing the raw Stanford Large–Scale 3D Indoor Spaces Dataset (S3DIS)

The S3DIS dataset already has each room stored as it's own separate point cloud that can be used in PCC Arena. Like the previous dataset, the normal information was computed for S3DIS point clouds using the scripts available in the *PCC Arena* repository. The point clouds were subsequently scaled to the same $[0, 1024]$ interval. An example of a point cloud from this dataset is shown on Figure 6.2.

### 6.1.3 Creating the scanned datasets

To create the actual scanned datasets using BlenSor, the *Stanford 2D-3D-Semantics Dataset* was used. Unlike its predecessor used in the previous section, which only consists of point clouds of the individual rooms, this expanded dataset includes meshes of the entire areas at once. These were used as virtual indoor environments to be scanned using the VLP-16 implementation in *BlenSor* created in Chapter 5. In order to scan the area meshes, suitable positions for the scanner had to be picked. These were sampled from the collection of positions that the dataset itself was created from (i.e. the positions that the Matterport Camera was located at when scanning happened).

Because in a virtual scan there is no need to worry about the human standing behind the sensor, the scans were performed in the full

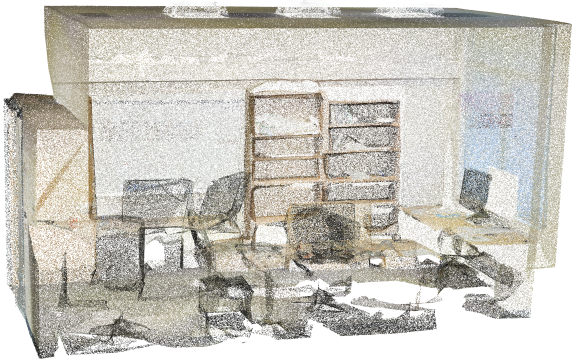
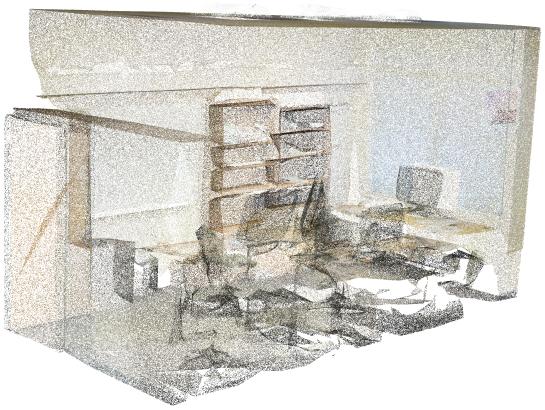


Figure 6.2: An example of a point cloud from the S3DIS dataset. The point cloud was created from an indoor scan of an office environment. Bookshelves, chairs and desks can be discerned.

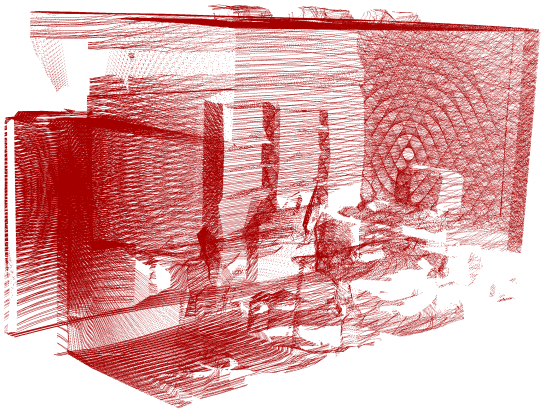
360° horizontal field-of-view. An important consideration with outward-looking scans is the high likelihood of the presence of outliers. Open doors and windows in scanned rooms can lead to the scanner picking up samples from outside of the current room. The scans were performed in each room inside the 6 large scale area meshes, which means that the laser would likely scan surfaces outside the room it was in (through open doors). To create more compact clouds, the points collected from these surfaces were cropped away. This was done using the raw point cloud data segmented into individual rooms in Armeni, Sener et al., 2016. By obtaining the bounding box using this data, the scanned point clouds were cropped. This is possible because of the shared coordinate system. The procedure is situated in the `s3dis_pipeline.py` script. An example of the same room in its raw and scanned forms is shown in Figure 6.3.

### 6.1.4 Algorithm selection

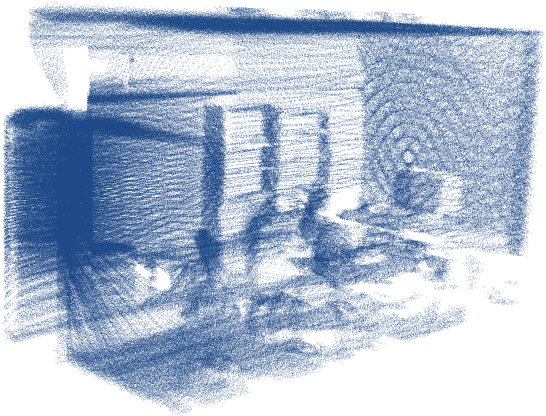
A subset of the algorithms used in the PCC Arena paper was used for the evaluation in this thesis. Namely the Draco and GPCC algorithms, based on traditional signal processing, and the PCGCv1 and PCGCv2 algorithms, based on neural networks. The VPCC algorithm is used for point cloud videos, and was thus not considered. The remaining two algorithms used by the paper, GeoCNNv1 and GeoCNNv2, were too computationally expensive to run on the machines available during writing of this thesis and were therefore also not used. Details of the used algorithms are presented in Section 3.4.



(a) Raw



(b) Scan without noise



(c) Scan with noise

Figure 6.3: Example of a room point cloud from the S3DIS dataset, along with its scanned versions, one without noise and one with noise added.

### 6.1.5 Metric selection

To measure the performance of the compression algorithms, a subset of the metrics presented in the PCC Arena paper were chosen. These include the 3D coordinate–only metrics, such as Chamfer Distance (defined in Equation (2.4)) and Hausdorff distance (defined in Equation (2.5)), as well as the encoding and decoding times. No color information is picked up by the VLP-16 sensor, or its virtual equivalent, so metrics which incorporate this information were not used. No subjective metrics were used either, though it is an interesting area to be further studied. As observed by Wu, Hsu, Hung et al. (2022), the correlation between the currently used objective metrics and subjective metrics is not significant, especially for object datasets. The point-to-plane versions of the metrics are used as they are a better measure of visual quality (Tian et al., 2017).

*PCC Arena* works with the notion of *compression rates* which describe a set of parameters for a particular algorithm that determine how much data is compressed. These format of these paremeters differs for each algorithm. In signal processing–based ones they often control the quantization granularity, whereas in the neural network-based ones they specify the pretrained models to use. The four algorithms were applied to the four datasets at rates ranging from $r1$ to $r8$, where a higher number means more compression, trading size for quality. The rates for *PCGCv1* and *PCGCv2* specifically ranged from $r1$ to $r6$ and $r1$ to $r7$ respectively. The parameter values were the default values available in PCC Arena. To add the aforementioned datasets into the framework, changes had to be made to the `cfgs/datasets.yml` configuration file, which was updated with the paths to the datasets and their versions with normals.

### 6.1.6 Scaling and compactness

The rooms in the S3DIS dataset have a high variability of shape. While the majority are offices close to a standard rectangle, there are several long and narrow hallways, or sprawling auditoriums. An assumption was therefore initially made that these rooms should be removed from the dataset as they will have a skewed scale compared to the other rooms when scaled to the same $[0, 1024]$ interval. Even so, the scaled rooms were scanned and used in the algorithm evaluation with the possibility of removing them later.

To see if the shape of the room has any bearing on the performance, the *elongation* metric of each room was computed from the raw point clouds in the S3DIS dataset. This metric is borrowed from 2D image processing where it's used to determine how compact a convex shape is, by dividing the width of the shape's bounding box with it's height. The closer the resulting value is to 1.0, the more square it is. The elongation metric for S3DIS rooms was defined as the length of the shortest axis divided by the length of the longest axis in the room's bounding box. All the rooms can also be thought of as convex shapes. The code for extracting this information from all the rooms can be found in the *get_elongation.ipynb* Jupyter notebook.

However, no discernible effect of the elongation on algorithm performance in the chosen metrics was observed. This could be explained by the fact that the Chamfer and Hausdorff Distance metrics simply aren't able to show the effect. The highly elongated rooms are often of two types, either long hallways with little clutter in them, or big auditoriums, with relatively uniform clutter made up of chairs or stairs. This means that the structural elements of the rooms—the walls, floor and ceiling—dominate the metrics, and the potential error created by the different scales gets lost as noise. As such, the rooms with the higher elongations were kept in the datasets.

## 6.2 Results and discussion

Figure 6.4 shows the distributions of point counts in the four studied datasets. As mentioned in Section 6.1.1 and apparent from the figure, the CAPOD dataset is sampled at 500 000 points per cloud. On the other hand, the raw S3DIS dataset has a wide spread of point counts with the median at about 800 000. The high variability can be explained by the fact that the individual scenes can differ significantly in their real world scale — the dataset contains mostly small offices but also several big auditoriums.

The last two datasets, which are a result of the virtual scanning pipeline, show a smaller spread of counts, but with a higher median. This, on the other hand, is a consequence of the procedure—each room was scanned for the same time duration with the same sensor parameters. Even though the rooms can be different in scale, the number of returns is going to vary less, as all the rooms are closed off environments. At the same time, the rooms do not get big enough for the virtual sensor to start missing returns because of light attenuation as described in Section 5.2.

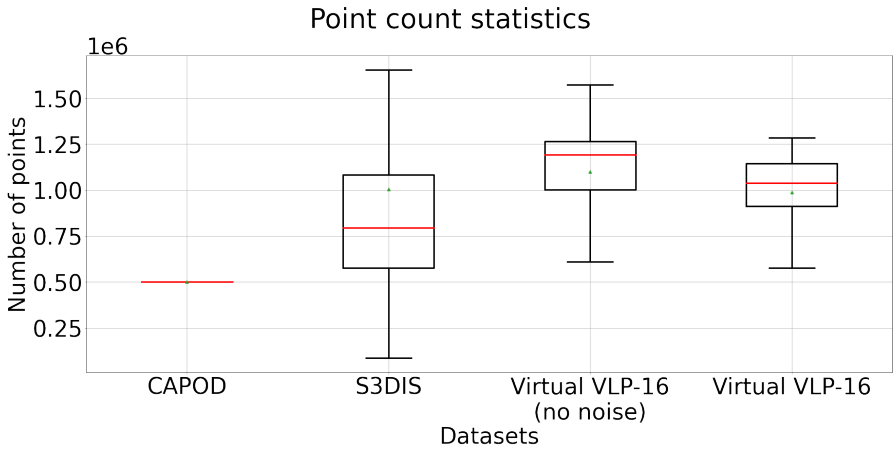



Figure 6.4: Distributions of the number of points present in the individual datasets. The CAPOD dataset point clouds are a result of randomly sampling on the original meshes with a set number of 500 000 points.

A coarse summary of the algorithm performance is shown in Figure 6.5.

The selected coordinate metrics of Chamfer Distance, Hausdorff Distance and the encoding and decoding times are shown on a log scale against the compression rates. At each rate, the median of the values is calculated and plotted. Several things can be observed from this plot:

1. For the Chamfer Distance metrics, all four algorithms improve their performance on all datasets when going up the compression rates, but with the exception of *Draco* this quickly levels off.

2. At low rates, the NN–based algorithms perform better or comparable to their SP–based counterparts.

3. For the Hausdorff distance metric, the NN–based algorithms lag behind, especially compared to GPCC. Worse performance at this metric, which highlights outliers, suggests that these algorithms are more likely to produce them.

4. As for decoding and encoding, the SP–based algorithms steadily increase the needed time as the compression rates go up, as well as PCGCv1. The times for PCGCv2 seem much less affected by the selected compression rate. A higher baseline for all the algorithms can be seen going from CAPOD to S3DIS and even more going to the scanned datasets. This can be explained by the higher median point counts in those datasets compared to CAPOD. GPCC's running time is the most affected by the selected compression rate.

5. The performance seems generally similar on the S3DIS and scanned datasets compared to CAPOD when looking at Chamfer Distance, with NN–based algorithms being slightly more affected in the Hausdorff Distance metric.

After looking at the coarse metrics, Figure 6.6 offers a more detailed look at the Chamfer Distance performance, this time on a log-log scale with the bits per point (bpp) value on the $x$-axis. bpp is calculated by dividing the size of the compressed file with the number of points in the original point cloud. This type of plot represents the *rate distortion curve*[1] of the different approaches. The bpp values were bucketed into 10 bins of equal number of samples and their median values were plotted in the figure.

Looking at the plots, we can discern that at lower bpp values, the NN–based algorithms and GPCC perform quite well, with Draco lagging behind. GPCC generally outperforms the NN–based algorithms once the bitrate increases. A significant horizontal offset of the Draco values for the scanned datasets can also be observed. This can be explained by the fact that *Draco* does not merge points occupying the same voxel in the quantization step (Wu, Hsu, Hung et al., 2022), which, coupled with the higher median point counts for the scanned datasets, inflates the bpp values. Apart from this, when comparing values for CAPOD versus the other datasets, no significant trends can be observed.

[1] A *rate distortion* curve describes the tradeoff between the quality and target compression rate of a compression system.

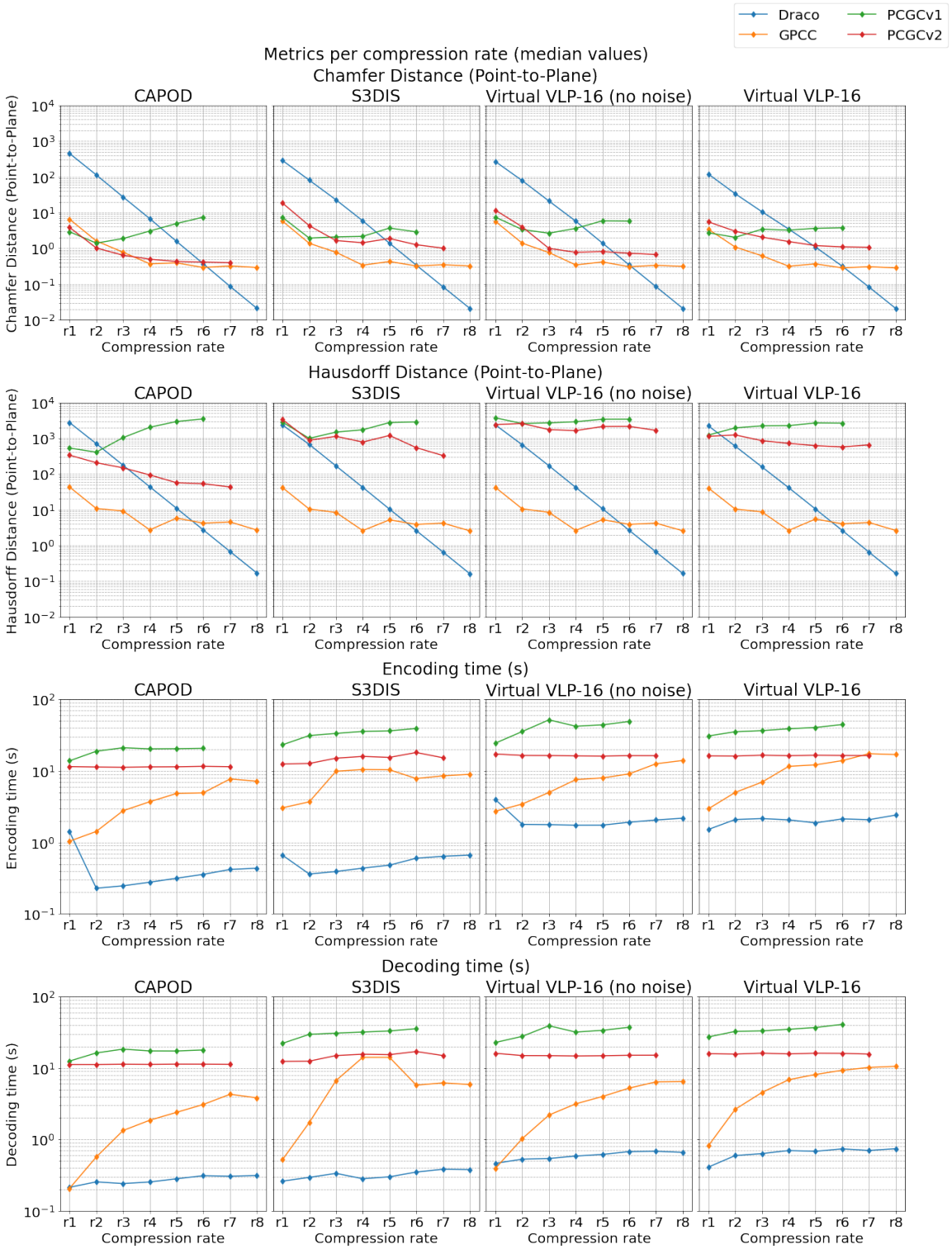



Figure 6.5: Coarse view of the metrics per compression rate. The values shown form the median value at each of the chosen compression rates.

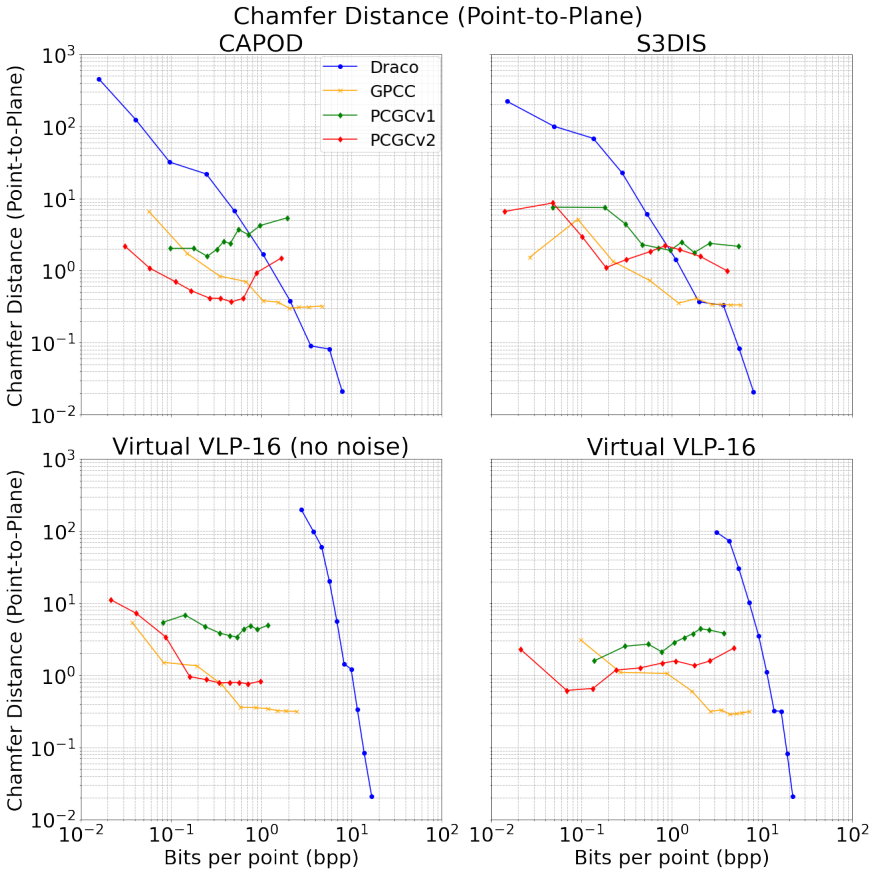



Figure 6.6: Comparison of the Chamfer distance metric in its point-to-plane version against the bits per point (bpp) value for the various algorithms on a log-log scale.

Somewhat more interesting trends can be observed in Figure 6.7, where the Hausdorff distance is plotted in more detail. These plots were created in a similar manner to the previous ones, showing a log-log scaled of the chosen metric against the bpp value, bucketed into 10 bins, with their medians plotted.

We can see a similar shift for the Draco values as in the previous plots. The NN–based algorithms perform somewhat worse at this metric in all datasets, compared to the SP–based ones, however the difference is especially big in the scanned datasets, implying a higher frequency of outliers.

Figures 6.8 and 6.9 show the encoding and decoding times plotted against the bpp metric. Similarly to the previous plots, they show a log-log scale of the chosen metric against the bpp value, bucketed into 10 bins, with their medians plotted. GPCC and to a smaller extent, PCGCv1 are the algorithms where the increasing bpp value means higher running time, whereas the others are relatively stable. Same shift as in the previous plots

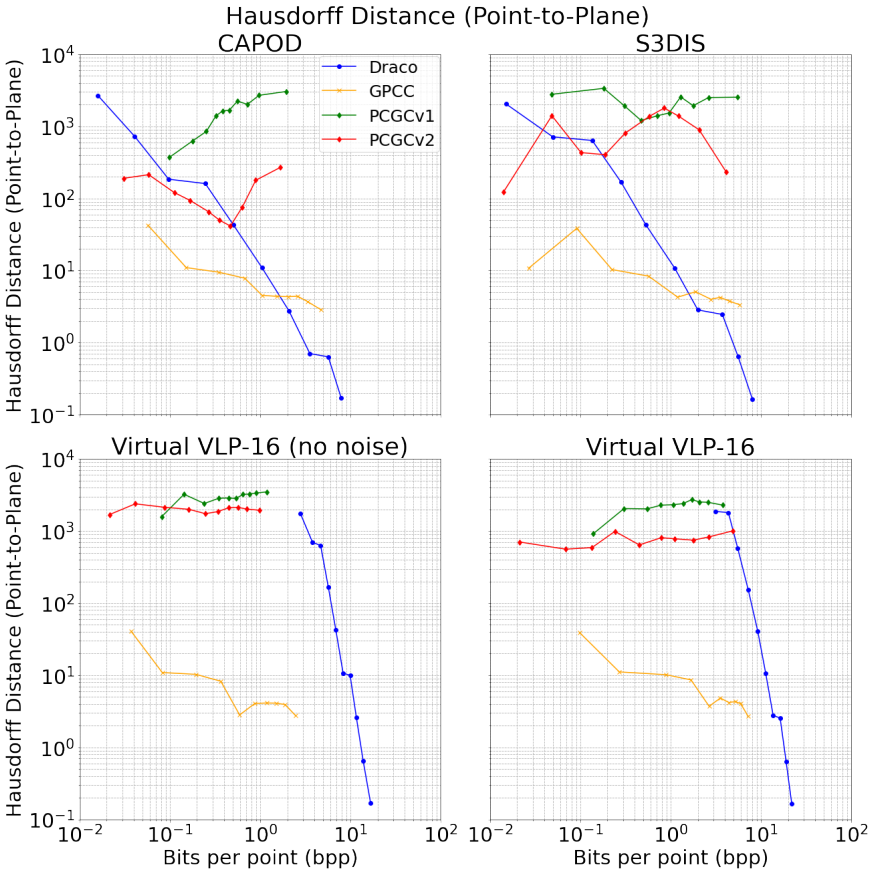



Figure 6.7: Comparison of the Hausdorff distance metric in its point-to-plane version against the bits per point (bpp) value for the various algorithms on a log-log scale.

can be observed for the Draco algorithm.

The results suggest that the performance of the evaluated algorithms is not significantly affected when presented with room-scale outward looking datasets. There is a small effect with the NN–based algorithms when looking at the Hausdorff distance metric, which highlights outliers. This is in line with the observations by Wu, Hsu, Hung et al. (2022), that the NN–based algorithms have a higher variance of performance and more potential for catastrophic failures. On the other hand, the results confirm the fact that these algorithms perform well at low bpp values. This is the case even when they have not been trained on room-scale outward looking datasets. A performance comparison after training them on this type of dataset would be a interesting direction to study further.

The fact that the SP–based algorithms are quite unaffected by the different type of scenes and presence of noise could be explained by the fact that these algorithms perform relatively simple operations on points themselves, without taking into account any actual geometry information.

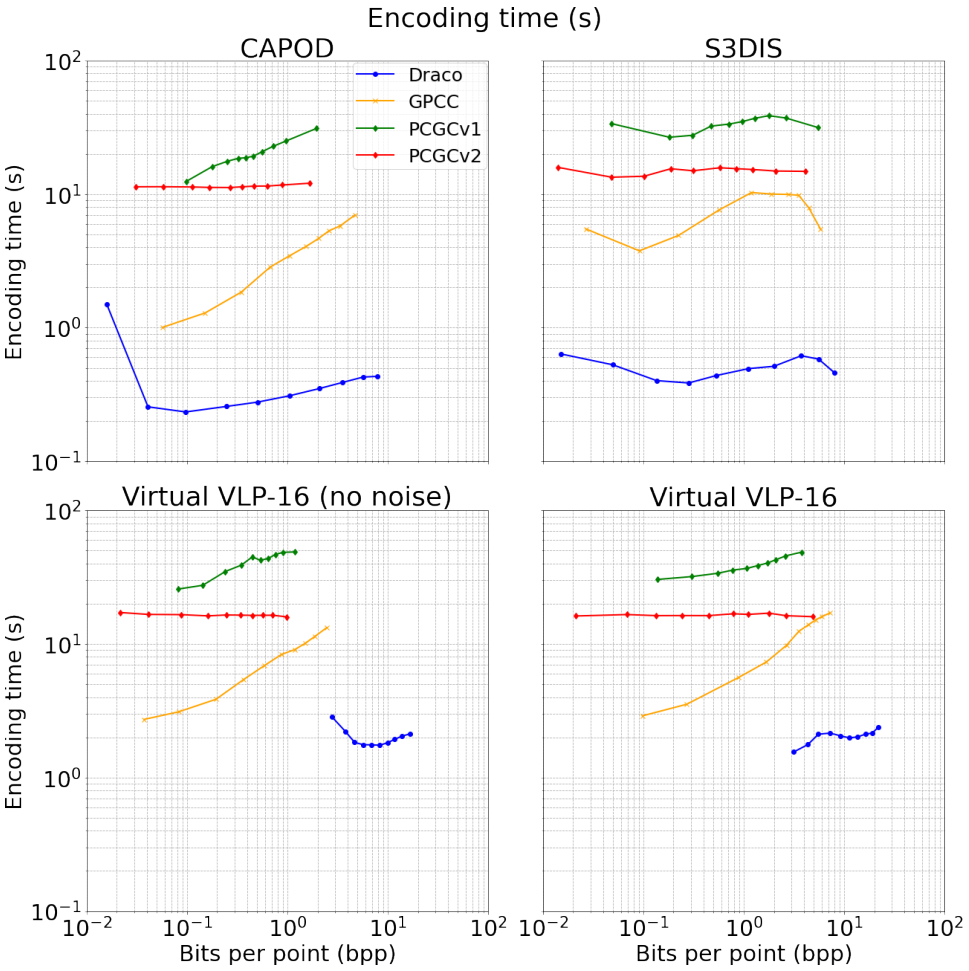



Figure 6.8: Comparison of the encoding time against the bits per point (bpp) value for the various algorithms on a log-log scale.

A point that is precise and one that is a part of noise is not going to affect how the algorithms treat that particular point.

Considering the goal of the effort that this thesis is a part of—to create a point cloud compression system based on geometric primitive fitting, such a system might be *more* susceptible to noise and type of scene than the SP–based algorithms. Presence of noise and occlusion is likely going to significantly affect the fitting of primitives and potentially lead to errors more easily. Whether the savings in data needed to be stored in such a system outweight this tendency remains to be seen. It is also important to note that using the S3DIS as a baseline for our scanning procedure means that the scanned datasets have compound error from both the original dataset acqusition (even though it acquired with the goal was to minimize error) and the BlenSor error model.

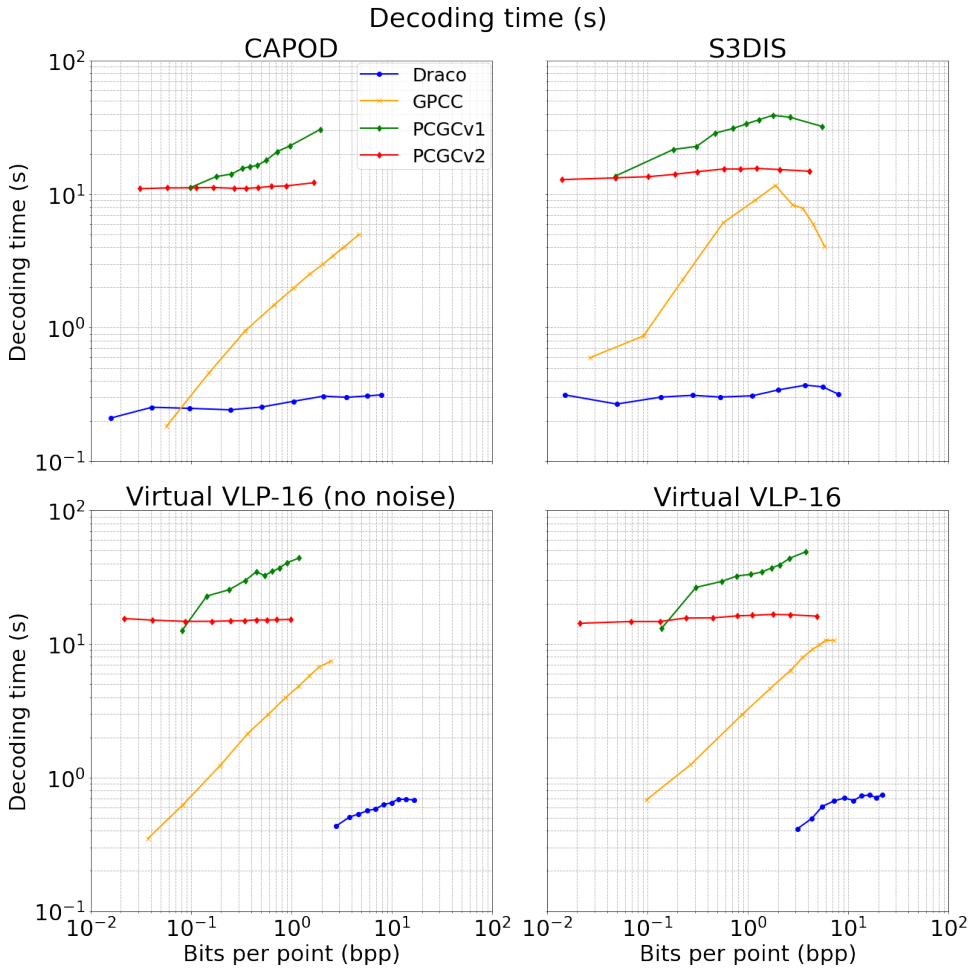



Figure 6.9: Comparison of the decoding time against the bits per point (bpp) value for the various algorithms on a log-log scale.

# Chapter 7

# Preparation for geometric primitive fitting

## 7.1 Background and related work

One of the most efficient ways to represent 3D scenes is to describe them using simple geometric primitives. This is also the basis of Constructive Solid Geometry (CSG), where complex objects are made up of Boolean operations (such as intersection, union, or difference) on simple geometric primitives. As presented by Kaiser, Ybanez Zepeda and Boubekeur (2019), simple 3D geometric primitives can be defined as convex symmetric geometric shapes with a limited set of parameters that completely describe the size, orientation and position of the shape. The authors categorize these shapes into 4 categories:

- **Planes**

  The most basic shape that can be defined by its normal vector $\mathbf{n}$ and distance $d$ to the origin of the reference frame.

- **Boxes and cuboids**

  A combination of orthogonal planes. These shapes are defined by their center position $\mathbf{x}_0$, orientation $\mathbf{n}$ and the lengths $a, b, c$ of their edges.

- **Spheres, cylinders and cones**

  These shapes can be defined as rotational sweeps of a specific 2D curve or line segments along an axis. A sphere can be defined by its center position $\mathbf{x}_0$ and radius $r$, while the cylinder needs an additional orientation vector $\mathbf{n}$. The cone adds the parameter $\alpha$ representing the angle between its axis and the surface.

- **Others shapes**

  Ellipsoids, tori and non-rectangular parallelepipeds and others are shapes of higher complexity that usually need several additional parameters on top of the center position $\mathbf{x}_0$ and orientation vector $\mathbf{n}$.

- **Superquadrics** Superquadrics are a powerful family of parametric shapes that extend the basic quadric surfaces and solids. They can describe several types of shapes, like cubes, cylinders, spheres and others using a relatively simple parametrization. They were first introduced by Barr (1981) and subsequently used in the context of describing more complex shapes by Pentland (1986). Superquadrics can be described implicitly as

$$|x|^r + |y|^s + |z|^t = 1 \tag{7.1}$$

  where the $r$, $s$ and $t$ parameters control the overall shape of the solid.

In order to efficiently process massive point clouds, it is important to try and reduce the size of the data and remove noise. The hypothetical approach to achieve this explored in this thesis is to find simple geometry that can be represented more efficiently and is useful for computer graphics applications (Deschaud and Goulette, 2010). It is natural to start with the simplest geometric primitives—planes. This is especially useful in indoor environments, which usually have many planar surfaces. Some of them, like the floor, walls and ceiling are even almost ubiquitous and generally follow the Manhattan world assumption. Other objects in the indoor environment can be approximated by other primitives, notably cylinders and cones for structural columns, table legs, cups, vases, curtain rails, segments of power cables or plumbing fixtures. As a step further, furniture can be approximated by a composition of many different primitives. Over the last decades, several different approaches to geometric primitive detection have emerged.

### 7.1.1 Hough transform

The Hough transform is a parameter space–based technique that uses an accumulative procedure where each sample of the input data votes for all the primitives of which they are inliers. The procedure then takes the maximum of the accumulated votes in the parameter space to determine a likely candidate for a primitive that best fits the data.

It is named after Paul Hough, who first used it to detect lines in bubble chamber images (Hough, 1959) and subsequently patented it. Researchers have since then expanded on it and generalized it to detect a multitude of primitives in both 2D and 3D data. It is robust to noise and missing data, which makes it very useful in robotics and other areas that frequently use low-cost sensors suffering from noise. However, the procedure is computationally expensive both in terms of processing time and memory requirements, especially as the number of parameters increases, which is the case when detecting more complex primitives. In the case of detecting planes in 3D space, the computation cost of the standard Hough transform is $\mathcal{O}(nN_\theta N_\phi)$, where $n$ is the number of points in the point cloud, and $N_\theta$ and $N_\phi$ are the number of bins used to discretize angles $\theta$ and $\phi$, which represent the orientation of the plane (Araújo and Oliveira, 2020).

A lot of work has therefore been put into improving the performance and reducing complexity of the Hough transform, notably the *Fast Hough transform* (H. Li, Lavin and Le Master, 1986) which hierarchically divides the Hough space into squares, and *Randomized Hough transform* (Kultanen, Xu and Oja, 1990), which was used to more efficiently detect curves in images. *Kernel-based Hough transform* uses an elliptical-Gaussian kernel, which significantly improves the speed of the voting scheme, making the technique work in real time (Fernandes and Oliveira, 2008).

In the area of 3D plane detectors, some kind of hierarchical voxelization of the point cloud is often used, such as in (Hulik et al., 2014), where the researchers also apply Gaussian smoothing and specific peak extraction in the parameter space. The *Kernel Hough transform* was also extended to 3D using octrees on point clouds along with principal component analysis (PCA) to achieve real-time performance, at the cost of higher sensitivity to outliers (Limberger and Oliveira, 2015). Rabbani and Heuvel (2005) optimized the Hough transform to detect cylinders in point clouds. As the standard approach would lead to a computationally expensive 5-dimensional Hough space, they decide to split the procedure into two sequential steps of lower dimensional Hough transforms. While the first step detects candidates for orientations of cylinders, the second step then estimates the position and radius.

### 7.1.2 Random sample consensus

Random sample consensus (RANSAC) is a stochastic method for estimating model parameters. Applications based on this method are usually iterative, non-deterministic and robust against outliers (Kaiser, Ybanez Zepeda and Boubekeur, 2019). The usual algorithm is a two-step procedure. In the first step, it draws a minimal random sample from the input data and estimates a model. In the second step, it evaluates all the input data on the estimated model using an inlier function or another similarity measure. The steps are repeated and the model with best estimated score is returned. The process can be repeated for finding multiple models. Various approaches that use this mechanism usually differ in how they evaluate the estimated models against the entire data set.

This approach does not work well on point clouds that are not uniformly distributed, a requirement often not met in real world scans. This leads to poor performance on samples farther away from the sensor, as they are underrepresented in the resulting point cloud. It is also highly sensitive to parameter tuning, namely the threshold used in the inlier function that is highly dependent on the noise level of the point cloud. The stochastic nature of these techniques makes it hard to produce repeatable results. The computational complexity of finding a single plane in 3D space is equal to $\mathcal{O}(I(E + nK))$ where $I$ is the number of iterations, $E$ is the cost to estimate a plane from three samples and $K$ is the cost of checking if a sample satisfies the inlier function on the estimated plane (Araújo and Oliveira, 2020).

Schnabel, Wahl and Klein (2007) use an optimized RANSAC approach

to detect different simple geometric primitives in point clouds, where only nearby samples are drawn and sample normals are used to improve the inlier function. Like the original formulation, this approach can suffer from detecting spurious planes in presence of noise. To improve the technique, L. Li et al. (2017) use Normal Distribution Transformation (NDT) cells as minimal samples in each iteration. NDT cells are another one of many approaches to subdivide the space occupied by the point cloud into a grid of cells. The subdivision can be regular, or use octress. A Probability Density Function (PDF) is computed for each cell, representing the point distribution. Only cells containing possible coplanar samples are then used in the RANSAC algorithm.

### 7.1.3 Region growing

Region-growing (also known as primitive-growing) approaches take a different direction. The techniques treat the point cloud as a graph, where samples are treated as the vertices and each sample has a neighbourhood that defines the samples that are connected to it in the graph. This can be problematic in many types of point clouds, where there is no a priori sample neighbourhood information. The algorithms starts by selecting either a regular or random seed from the sample set and starts a graph search based on some similarity function. This function can use different properties of the samples, such as color, normal orientation or distance. This function depends on the primitive to be detected. Finding good thresholds for this function is also highly dependent on the type of scene and the application, a property that makes this technique suffer from a lot of parameter tuning. Much like with previous approaches, some applications partition the space into voxel grids and perform the actual algorithm within the segmented cells.

To avoid scene dependency when choosing parameters for the similarity function, Maalek, Lichti and Ruwanpura (2018) used the Robust PCA method to the neighbourhood of each sample. This method is however computationally expensive. Pham et al. (2016) extend the graph-based approach by considering both global and local planar surfaces, using a global energy minimization algorithm, making it especially suitable for indoor environment. Combining region growing with the Hough transform, Leng, Xiao and Y. Wang (2016) map each sample's estimated normal to a unit sphere accumulator, cluster the predominant directions and perform region growing within these clusters.

### 7.1.4 Robust statistics

One of the most recent approaches in plane detection is the work done by Araújo and Oliveira (2020). Conceptually it's similar to region growing, however the authors propose an $\mathcal{O}(n\ log\, n)$ algorithm for detecting planes that is robust to noise, scalable and does not require parameter tuning. They achieve this with a planarity test based on robust statistics that easily

removes outliers. [1]

The algorithm consists of three phases:

1. **Split phase** – detects minimal planar regions in the point cloud and subdivides it into an octree, where the leaves have at maximum $\epsilon$ samples, experimentally determined to be 0.1% of all samples. On each of these leaves, they apply their robust planarity test. This test first estimates a plane (it is point and a normal vector) and then performs checks to see if the patch passes the test. After going through all the leaves, the accepted ones move into the growth phase. Each accepted patch has information about the estimated plane and the distributions of distances and normal deviations from it for each point in the patch.

2. **Growth phase** – a neighbourhood graph of the entire point cloud is created and, in a breadth-first search manner, neighbouring samples are tested for an *inlier condition* and added to the currently grown patch if they are not a part of any other patch. After this is done, some patches might belong to the same plane, and they should be merged.

3. **Merge phase** – Each pair of planar patches is evaluated and merged if they pass several merging conditions.

The grow-merge phase is done iteratively, because after merging, the plane of the merged patch might be re-estimated, along with the distributions of distances and standard deviations. This stops when all patches are stable.

As the authors point out, this is an entirely bottom-up solution, which means local planarity tests are performed, disregarding the global geometry. This may lead to detection of false planes along low curvature surfaces.

### 7.1.5 Local statistics

Some approaches look at the occupancy distributions in the point clouds to try and estimate primitives using Gaussian Mixture Models or Bayesian methods (such as in the work of Jenke et al. (2006) and Diebel, Thrun and Brünig (2006)). These are computationally expensive for larger point clouds.

### 7.1.6 Generative modeling

Generative modeling is a form of unsupervised machine learning, where the goal is to take training samples as input from some distribution and learn a model that represents that distribution. The end goal for this can be

[1] The term robust statistics is used to describe statistical approaches that are designed to be insensitive to outliers. (Huber, 2004, p. 2) An example of this is to work with the median value, rather than the mean.

e.g. density estimation, sample generation, debiasing or outlier detection. Generative models can be a promising approach to use for point cloud data, where big labelled datasets are not available.

Kingma and Welling (2019) summarize the tradeoffs with generative models as such:

> While generative models can learn efficiently from data, they also tend to make stronger assumptions on the data than their purely discriminative counterparts, often leading to higher asymptotic bias when the model is wrong.

They conclude that in regimes with sufficiently large amounts of data, discriminative approaches will perform better, but studying the underlying generative models can be helpful regardless. As an example, Bagautdinov, Fleuret and Fua (2015) use generative models to evaluate the probability of target objects being present in the scene in single depth maps.

### 7.1.7 Shape parsing and parsimonious representation

The general approach of finding geometric primitives in 3D data for a more compact representation is called *shape parsing*. An important idea in detecting shapes from point clouds is *parsimony*, i.e. trying to find the minimal number of primitives that best fit the scene at hand. A more parsimonious representation has the benefit of being a low dimensional description, and therefore easier for machines to parse. It is hypothesized that human understanding of complex scenes also relies on such parsimonious representations using simple shape primitives (Biederman, 1987). Until recently, the computational power needed to extract shapes from scenes in this way was prohibitive. Steady rise in computational resources, along with the new developments in deep learning, have brought this area back into focus. The idea of parsimonious representation is central to the approach of point cloud compression by geometric primitive fitting.

Tulsiani et al. (2017) explore the idea of fitting cuboids as the simplest possible primitives. To tackle the *object assembly* problem, i.e. determining what cuboids approximate a particular object, they use 3D convolutional neural networks on a volumetric representation of an object point cloud in an unsupervised manner. On the other hand, Zou et al. (2017) use a generative recurrent neural network to encode implicit shape representation. They highlight the lack of good training datasets in this area and propose a generative method for creating ground truth data. Paschalidou, Ulusoy and Geiger (2019) build on this research by switching from cuboids to *superquadrics*, a family of surfaces that can describe several types of shapes, like cubes, cylinders, spheres and others using a relatively simple parametrization. They argue that cuboids are not sufficiently expressive to model many natural and man-made shapes. This in turn requires a larger amount of primitives to be used. Most importantly, superquadrics have a *continuous* parametrization determining their shape smoothly, which is amenable to deep learning.

## 7.2 Approach and discussion

The approach of using geometric primitive fitting as a means of compression was in the work of Boulet-Gilly (2019) and Hansen (2020). Boulet-Gilly (2019) used a region-growing approach to extract planes corresponding to the floor, ceiling and walls in indoor scene point clouds. Subsequently, high-frequency detail pertaining to these planes was brought back into the representation using heightmaps. Hansen (2020) searched for a more general and robust way to extract the main planar features using neural networks. He also used the S3DIS dataset which has label information for 13 classes, like floor, window, column etc., and trained a deep neural network with spherical kernels, modified from the work of Lei, Akhtar and Mian (2019).

This network was used to segment the point clouds into four classes—floor, walls, ceiling and the rest (interior)—created by partially combining the original 13 classes from the S3DIS dataset. After inference, a sequential planar least squares algorithm iteratively fits a planar model to the data and removes it, to finally arrive at a segmented point cloud. In the end, height maps are used to retrieve the high frequency data. As an aside, Hansen (2020) explored using the height maps for point cloud registration purposes. He also pursued a different approach to compression, using an autoencoder network, a component used in the neural network–based algorithms described in Section 3.4.

In light of his work and the contribution of this thesis, the next step in achieving geometry fitting compression is to look at primitives other than planes. Cuboids, cylinders and spheres are the next most simple candidates. As described in Section 7.1, a plane is well defined by its normal vector and the distance to the reference frame origin. But unlike an infinite plane, cuboids and the other primitives must be described by their position, orientation, and additional parameters defining their extents. The question of composed primitives also arises. A box could be described with the aformentioned parameters, or with the parameters of its six constituent planes. The difference between these two representations is that, as a box, the relationship between the planes is encoded—they have to be at right angles to each other. On the other hand, six planes can represent other shapes than a box. As such, the walls, floor and ceiling of rectangular or near-rectangular rooms could be instead described as a cuboid instead of six planes.

The main reason for starting with planes for walls, floor and ceiling, apart from them being the simplest primitive, has been the fact that existing indoor scene datasets, like the S3DIS dataset, already include this ground truth. However, these labels are semantic in nature, not geometric. Nonetheless they have been useful as a starting point with the assumption that these surface are planar in nature.

Another dataset that was explored during the writing of this thesis is 3D-FRONT, a large-scale, mesh-based collection of synthetic indoor scenes (Fu, Cai et al., 2021). Unlike the S3DIS dataset, these scenes represent usual living quarters and combine the structural layout information with

furniture models from a complementary dataset, 3D-FUTURE (Fu, Jia et al., 2021). This dataset also obtains semantic ground truths on the structural elements that could be exploited in a plane extraction system.

Notably, a general geometric fitting approach must deal with two problems. First, it is the geometry primitive detection, and second, it is the segmentation of their individual instances. This is similar to the problems of object detection and object instance segmentation in the image processing domain. A system that has to deal with these two problems could work in an iterative fashion:

1. detect a primitive in the point cloud
2. perform fitting to obtain primitive parameters
3. remove the primitive from the point cloud, and repeat until no primitives are detected

After fitting the individual primitives, their parameters are known and heightmap extraction can be performed to retrieve high frequency detail. Realistic error modelling is going to be crucial in this phase. At the same time, this approach might have to be hierarchical or multi-scale, iteratively advancing from the largest features to the smallest. As an alternative to supervised methods, reinforcement learning methods could be employed as well. A learning agent might use a trial-and-error approach for picking primitives with a reward function that signifies coverage of the point cloud. A similar iterative procedure could then be used.

BlenSor could be used to synthetically generate a point cloud dataset with a realistic error model and geometric ground truths. Each point can be labeled by the primitive it belongs to. The use of data augmentation is also possible, by using existing datasets and adding procedurally generated assets, or combining multiple datasets, similarly to 3D-FRONT and 3D-FUTURE. The biggest issue in this area is how to avoid encoding biases in the synthetic data, something that other LiDAR simulation efforts also highlight (Anderson et al., 2019).

# Chapter 8

# Conclusion and future work

This thesis explored the steps necessary to create a point cloud compression system based on geometric primitive fitting for room-scale LiDAR scenes. It is a continuation of the work done by Sillerud and Johanssen (2019), Boulet-Gilly (2019) and Hansen (2020), who focused on creating a high density point cloud acquisition system using the Velodyne VLP-16 sensor and subsequently on approaches of plane extraction from these point clouds.

The premise was that current studies that apply machine learning to point clouds usually work with rather clean data. This means uniformly sampled point clouds without any noise. This informed the first part of the work, where the noise characteristics of the Velodyne VLP-16 as applied to room-scale environments were studied and a virtual version of the sensor with a more realistic error model was implemented. This work is important for eventually training a supervised system for point cloud compression using geometric primitive fitting.

The contribution of this thesis is three-fold:

1. A survey of the literature was conducted, focused on point cloud processing specifically in the areas of synthetic data generation, machine learning and point cloud compression

2. A usable virtual LiDAR system with the focus on realistic error modelling was created by extending the BlenSor plugin. The error model used by the system is based on a real Velodyne VLP-16 sensor.

3. Performance of existing point cloud compression approaches was evaluated on noisy room-scale indoor environment point clouds, extending the *PCC Arena* benchmarking framework.

The results suggest that a base Gaussian noise applied to the distance measurements is a good approximation of the noise characteristics of a rotating LiDAR sensor. The important aspect of modelling the noise is to choose the correct mean and standard deviation based on measurements in the real world. While previous work has noted variance in the performance of the different laser channels for the VLP-16 sensor, this was not observed. This could be explained by the fact that the studied point clouds are a

combination of many different point cloud fragments coming from the acquisition procedure. For detailed exploration of these characteristics, a precise calibration study of the sensor is necessary.

Modelling beam divergence has a limited effect in short distance regimes such as standard indoor scenes. The role of the incidence angle on the noise distribution should be further studied. Different metrics other than a plane residual could be used to more accurately discern the relationship. The specifics of modelling these factors will also depend on the system used for the virtual sensor. Further work on synthetic point cloud generation should put generally more focus on noisy and non-uniformly distributed datasets.

Among the existing point cloud compression approaches, the traditional signal processing–based ones do well when applied to noisy room-scale data, comparably to noisy object data. On the other hand, neural network–based algorithms display higher variance and lower results in the worst-case Hausdorff distance metric. A good point of comparison would be to train the neural network–based algorithms on indoor noisy datasets and evaluate them again.

Interestingly, the evaluated algorithms all rely on a quantization step, utilizing a 3D autoencoder or variational autoencoder, or performing 3D convolutions. Future work should focus on better exploiting the point sets themselves as exemplified by the newer approaches of working on point cloud data, such as *PointNet*, graph convolutions and the attention mechanism.

To achieve the goal of creating a usable system for detecting geometric primitives, more headway has to be done in this direction. The possible approaches and work done to further this goal were outlined in Chapter 7. Future work in this area should focus on creating datasets with geometric labels using the system explored in this thesis.

The repository with scripts and other programs that were created as part of this thesis is available at https://github.uio.no/branisj/thesis.

# Bibliography


Anderson, Reed et al. (2019). 'Deep Learning for Segmentation of Point Clouds with Synthetic Vegetation'. In: *Deep Learning in Photogrammetry, Remote Sensing and Geospatial Information Processing, 4th PhD Colloquium of the DGK Division on Geoinformatics and the DGPF Working Group on Geoinformatics*. Rostock, pp. 1–3.

Araújo, Abner M.C and Manuel M. Oliveira (Apr. 2020). 'A robust statistics approach for plane detection in unorganized point clouds'. en. In: *Pattern Recognition* 100, p. 107115. ISSN: 00313203. DOI: 10.1016/j.patcog.2019.107115.

Armeni, Iro, A. Sax et al. (Feb. 2017). 'Joint 2D-3D-Semantic Data for Indoor Scene Understanding'. In: *ArXiv e-prints*. arXiv: 1702.01105.

Armeni, Iro, Ozan Sener et al. (June 2016). '3D Semantic Parsing of Large-Scale Indoor Spaces'. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1534–1543. DOI: 10.1109/CVPR.2016.170.

Bagautdinov, T., F. Fleuret and P. Fua (2015). 'Probability occupancy maps for occluded depth images'. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2829–2837. DOI: 10.1109/CVPR.2015.7298900.

Barr (Jan. 1981). 'Superquadrics and Angle-Preserving Transformations'. In: *IEEE Computer Graphics and Applications* 1.1, pp. 11–23. ISSN: 1558-1756. DOI: 10.1109/MCG.1981.1673799.

Bi, Shusheng et al. (Jan. 2021). 'A Survey of Low-Cost 3D Laser Scanning Technology'. In: *Applied Sciences* 11.9, p. 3938. ISSN: 2076-3417. DOI: 10.3390/app11093938.

Biederman, Irving (1987). 'Recognition-by-Components: A Theory of Human Image Understanding'. In: *Psychological Review* 94.2, pp. 115–147. ISSN: 1939-1471(Electronic),0033-295X(Print). DOI: 10.1037/0033-295X.94.2.115.

Boulet-Gilly, Edmond (2019). 'Point Cloud Based Room Reconstruction'. eng. MA thesis. Toulouse Research Institute in Computer Science.

Caccia, Lucas et al. (2019). 'Deep Generative Modeling of LiDAR Data'. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5034–5040.

Chang, Angel X. et al. (2015). *ShapeNet: An Information-Rich 3D Model Repository*. Tech. rep. arXiv:1512.03012. Stanford University — Princeton University — Toyota Technological Institute at Chicago.

*CloudCompare* (2021). Version 2.11. URL: https://www.cloudcompare.org/.

Deschaud, Jean-Emmanuel and François Goulette (May 2010). 'A Fast and Accurate Plane Detection Algorithm for Large Noisy Point Clouds Using Filtered Normals and Voxel Growing'. In: *3DPVT*. Paris, France.

Diebel, James R., Sebastian Thrun and Michael Brünig (2006). 'A Bayesian method for probable surface reconstruction and decimation'. In: *ACM Transactions on Graphics*. ISSN: 07300301. DOI: 10.1097/00024382-200606001-00117.

Dosovitskiy, Alexey et al. (2017). 'CARLA: An Open Urban Driving Simulator'. In: *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16.

*Draco 3D Graphics Compression* (2022). https://google.github.io/draco/.

Fang, Jin et al. (Apr. 2020). 'Augmented LiDAR Simulator for Autonomous Driving'. en. In: *IEEE Robotics and Automation Letters* 5.2, pp. 1931–1938. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2020.2969927.

Fernandes, Leandro A.F and Manuel M. Oliveira (2008). 'Real-time line detection through an improved Hough transform voting scheme'. eng. In: *Pattern recognition* 41.1, pp. 299–314. ISSN: 0031-3203.

Fu, Huan, Bowen Cai et al. (2021). '3d-front: 3d furnished rooms with layouts and semantics'. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10933–10942.

Fu, Huan, Rongfei Jia et al. (2021). '3d-future: 3d furniture shape with texture'. In: *International Journal of Computer Vision*, pp. 1–25.

Fuchs, Fabian et al. (2020). 'SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks'. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 1970–1981.

Furukawa, Yasutaka et al. (Sept. 2009). 'Reconstructing Building Interiors from Images'. en. In: *2009 IEEE 12th International Conference on Computer Vision*. Kyoto: IEEE, pp. 80–87. ISBN: 978-1-4244-4420-5. DOI: 10.1109/ICCV.2009.5459145.

Geiger, Andreas et al. (2013). 'Vision meets Robotics: The KITTI Dataset'. In: *International Journal of Robotics Research (IJRR)*.

Glennie, Craig L., Arpan Kusari and Aldo Facchin (Mar. 2016). 'Calibration and Stability Analysis of the VLP-16 Laser Scanner'. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-3/W4, pp. 55–60. ISSN: 2194-9034. DOI: 10.5194/isprsarchives-XL-3-W4-55-2016.

*GNU General Public License* (29th June 2007). Version 3. Free Software Foundation. URL: http://www.gnu.org/licenses/gpl.html.

Goodfellow, Ian J. et al. (2014). 'Generative Adversarial Nets'. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, pp. 2672–2680.

Gschwandtner, Michael et al. (2011). 'BlenSor: Blender Sensor Simulation Toolbox'. In: *Advances in Visual Computing*. Ed. by George Bebis et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 199–208. ISBN: 978-3-642-24031-7.

Guennebaud, Gaël, Benoît Jacob et al. (2010). *Eigen v3*. URL: http://eigen.tuxfamily.org.

Hanke, Timo et al. (Oct. 2017). 'Generation and Validation of Virtual Point Cloud Data for Automated Driving Systems'. en. In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. Yokohama: IEEE, pp. 1–6. ISBN: 978-1-5386-1526-3. DOI: 10.1109/ITSC.2017.8317864.

Hansen, Henry Haugsten (2020). 'Indoor 3D reconstruction by Lidar and IMU and point cloud segmentation and compression with machine learning'. eng. MA thesis. University of Oslo.

Hansen, Henry Haugsten et al. (2020). 'Dense LIDAR Point Clouds from Room-Scale Scans'. In: *Proceedings of the 11th ACM Multimedia Systems Conference*. MMSys '20. Istanbul, Turkey: Association for Computing Machinery, pp. 88–98. ISBN: 9781450368452. DOI: 10.1145/3339825.3391862.

He, Kaiming et al. (2016). 'Deep Residual Learning for Image Recognition'. eng. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 770–778. ISBN: 9781467388511.

Hough, P.V.C. (1959). 'Machine Analysis of Bubble Chamber Pictures'. In: *Conf. Proc. C* 590914. Ed. by L. Kowarski, pp. 554–558.

Huber, P.J. (2004). *Robust Statistics*. Wiley Series in Probability and Statistics - Applied Probability and Statistics Section Series. Wiley. ISBN: 9780471650720.

Hulik, Rostislav et al. (2014). 'Continuous plane detection in point-cloud data based on 3D Hough Transform'. eng. In: *Journal of visual communication and image representation* 25.1, pp. 86–97. ISSN: 1047-3203.

Intel (2020). *Intel® RealSense™ Depth Camera D435i*. URL: https://www.intelrealsense.com/depth-camera-d435i/ (visited on 16/11/2020).

Jenke, P. et al. (2006). 'Bayesian point cloud reconstruction'. In: *Computer Graphics Forum*. ISSN: 14678659. DOI: 10.1111/j.1467-8659.2006.00957.x.

Kaiser, Adrien, Jose Alonso Ybanez Zepeda and Tamy Boubekeur (Feb. 2019). 'A Survey of Simple Geometric Primitives Detection Methods for Captured 3D Data'. en. In: *Computer Graphics Forum* 38.1, pp. 167–196. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/cgf.13451.

Kidd, John Ryan (2017). 'Performance Evaluation of the Velodyne VLP-16 System for Surface Feature Surveying'. In: p. 86.

Kingma, Diederik P. and Max Welling (2019). 'An Introduction to Variational Autoencoders'. In: *Foundations and Trends® in Machine Learning* 12.4, pp. 307–392. ISSN: 1935-8237. DOI: 10.1561/2200000056.

Krizhevsky, Alex, Ilya Sutskever and Geoffrey Hinton (May 2017). 'ImageNet Classification with Deep Convolutional Neural Networks'. en. In: *Communications of the ACM* 60.6, pp. 84–90. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3065386.

Kultanen, P., L. Xu and E. Oja (1990). 'Randomized Hough transform (RHT)'. eng. In: *[1990] Proceedings. 10th International Conference on Pattern Recognition*. Vol. i. IEEE Comput. Soc. Press, 631–635 vol.1. ISBN: 0818620625.

Landrieu, Loic and Martin Simonovsky (2018). 'Large-Scale Point Cloud Semantic Segmentation with Superpoint Graphs'. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4558–4567. DOI: 10.1109/CVPR.2018.00479.

Lei, Huan, Naveed Akhtar and Ajmal Mian (2019). 'Octree guided CNN with Spherical Kernels for 3D Point Clouds'. In: *IEEE Conference on Computer Vision and Pattern Recognition*.

Leng, Xiaoxu, Jun Xiao and Ying Wang (2016). 'A multi-scale plane-detection method based on the Hough transform and region growing'. eng. In: *Photogrammetric record* 31.154, pp. 166–192. ISSN: 0031-868X.

Li, Bo, Tianlei Zhang and Tian Xia (2016). 'Vehicle Detection from 3D Lidar Using Fully Convolutional Network'. In: *Robotics: Science and Systems XII, University of Michigan, Ann Arbor, Michigan, USA, June 18 - June 22, 2016*. Ed. by David Hsu et al. DOI: 10.15607/RSS.2016.XII.042.

Li, Hungwen, Mark A. Lavin and Ronald J. Le Master (1986). 'Fast Hough transform: A hierarchical approach'. eng. In: *Computer vision, graphics, and image processing* 36.2, pp. 139–161. ISSN: 0734-189X.

Li, Lin et al. (May 2017). 'An Improved RANSAC for 3D Point Cloud Plane Segmentation Based on Normal Distribution Transformation Cells'. en. In: *Remote Sensing* 9.5, p. 433. ISSN: 2072-4292. DOI: 10.3390/rs9050433.

Limberger, Frederico A. and Manuel M. Oliveira (June 2015). 'Real-Time Detection of Planar Regions in Unorganized Point Clouds'. en. In: *Pattern Recognition* 48.6, pp. 2043–2053. ISSN: 00313203. DOI: 10.1016/j.patcog.2014.12.020.

Liu, Yongcheng et al. (2019). 'Relation-shape convolutional neural network for point cloud analysis'. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. ISBN: 9781728132938. DOI: 10.1109/CVPR.2019.00910.

Maalek, Reza, Derek D. Lichti and Janaka Y. Ruwanpura (2018). 'Robust segmentation of planar and linear features of terrestrial laser scanner point clouds acquired from construction sites'. In: *Sensors (Switzerland)*. ISSN: 14248220. DOI: 10.3390/s18030819.

Manivasagam, Sivabalan et al. (2020). 'LiDARsim: Realistic LiDAR Simulation by Leveraging the Real World'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11167–11176.

*Matterport Pro2 3D Camera* (2022). https://matterport.com/cameras/pro2-3D-camera.

*MPEG Point Cloud Compression* (2022). *The Public Information on MPEG Point Cloud Compression*.

Muja, Marius and David G. Lowe (2009). 'Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration'. In: *VISAPP 2009 - Proceedings of the Fourth International Conference on Computer Vision Theory and Applications*, pp. 331–340.

Papadakis, P. (Apr. 2014). 'The Canonically Posed 3D Objects Dataset'. In: *Proceedings of the 7th Eurographics Workshop on 3D Object Retrieval*. 3DOR '14. Goslar, DEU: Eurographics Association, pp. 33–36. ISBN: 978-3-905674-58-3.

Paschalidou, Despoina, Ali Osman Ulusoy and Andreas Geiger (2019). 'Superquadrics Revisited: Learning 3D Shape Parsing beyond Cuboids'. In: *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Pentland, Alex P. (1986). 'Parts: Structured Descriptions of Shape'. In: *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*. AAAI'86. Philadelphia, Pennsylvania: AAAI Press, pp. 695–701.

Pham, T. T. et al. (2016). 'Geometrically consistent plane extraction for dense indoor 3D maps segmentation'. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4199–4204. DOI: 10.1109/IROS.2016.7759618.

*Puck Lidar Sensor, High-Value Surround Lidar* (Nov. 2020). URL: https://velodynelidar.com/products/puck.

Qi, Charles R., Hao Su, Kaichun Mo et al. (July 2017). 'PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation'. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, pp. 77–85. DOI: 10.1109/CVPR.2017.16.

Qi, Charles R., Hao Su, Matthias Niebner et al. (2016). 'Volumetric and Multi-view CNNs for Object Classification on 3D Data'. eng. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 5648–5656. ISBN: 9781467388511.

Qi, Charles R., Li Yi et al. (2017). 'PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space'. In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., pp. 5099–5108.

Quach, Maurice, Giuseppe Valenzise and Frédéric Dufaux (Sept. 2020). 'Improved Deep Point Cloud Geometry Compression'. In: *IEEE International Workshop on Multimedia Signal Processing (MMSP'2020)*. Tampere, Finland.

Quach, Maurice, Giuseppe Valenzise and Frederic Dufaux (Sept. 2019). 'Learning Convolutional Transforms for Lossy Point Cloud Geometry Compression'. In: *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 4320–4324. DOI: 10.1109/ICIP.2019.8803413.

Rabbani, Tahir and Frank Heuvel (Jan. 2005). 'Efficient Hough transform for automatic detection of cylinders in point clouds'. In: *Proc ISPRS Workshop Laser Scan 2005, ISPRS Arch* 36.

Richter, Stephan R. et al. (2016). 'Playing for Data: Ground Truth from Computer Games'. In: *European Conference on Computer Vision (ECCV)*. Ed. by Bastian Leibe et al. Vol. 9906. LNCS. Springer International Publishing, pp. 102–118.

Riegler, Gernot, Ali Osman Ulusoy and Andreas Geiger (2017). 'OctNet: Learning Deep 3D Representations at High Resolutions'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Ros, German et al. (June 2016). 'The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes'. en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Las Vegas, NV, USA: IEEE, pp. 3234–3243. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.352.

Rubner, Yossi, Carlo Tomasi and Leonidas Guibas (Feb. 1998). 'Metric for Distributions with Applications to Image Databases'. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 59–66. ISBN: 978-81-7319-221-0. DOI: 10.1109/ICCV.1998.710701.

Rumelhart, David E. and James L. McClelland (1987). 'Learning Internal Representations by Error Propagation'. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pp. 318–362.

Rusu, Radu Bogdan and Steve Cousins (May 2011). '3D is here: Point Cloud Library (PCL)'. In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China.

Schäling, Boris (2014). *The boost C++ libraries*. XML Press. ISBN: 9781937434366.

Schnabel, Ruwen, Roland Wahl and Reinhard Klein (2007). 'Efficient RANSAC for Point-Cloud Shape Detection'. eng. In: *Computer graphics forum* 26.2, pp. 214–226. ISSN: 1467-8659.

Shelhamer, Evan, Jonathan Long and Trevor Darrell (2017). 'Fully Convolutional Networks for Semantic Segmentation'. eng. In: *IEEE transactions on pattern analysis and machine intelligence* 39.4, pp. 640–651. ISSN: 0162-8828.

Sillerud, Vetle Smedbakken and Fredrik Kristoffer Johanssen (2019). 'Reconstruction of Indoor Environments Using LiDAR and IMU'. eng. MA thesis. University of Oslo.

Sobczak, Łukasz et al. (May 2021). 'LiDAR Point Cloud Generation for SLAM Algorithm Evaluation'. en. In: *Sensors* 21.10, p. 3313. ISSN: 1424-8220. DOI: 10.3390/s21103313.

Su, Hang et al. (2015). 'Multi-view Convolutional Neural Networks for 3D Shape Recognition'. eng. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, pp. 945–953. ISBN: 1467383910.

Sugiura, Tsukasa (Mar. 2022). *VelodyneCapture*.

*The PCD File Format* (2022). *Point Cloud Library 1.11.1 Documentation*. URL: https://pcl.readthedocs.io.

Tian, Dong et al. (Sept. 2017). 'Geometric Distortion Metrics for Point Cloud Compression'. In: *2017 IEEE International Conference on Image Processing (ICIP)*, pp. 3460–3464. DOI: 10.1109/ICIP.2017.8296925.

Tulsiani, Shubham et al. (2017). 'Learning Shape Abstractions by Assembling Volumetric Primitives'. In: *Computer Vision and Pattern Regognition (CVPR)*.

*Velodyne LiDAR: VLP-16 User Manual* (2019). URL: https://velodynelidar.com/wp-content/uploads/2019/12/63-9243-Rev-E-VLP-16-User-Manual.pdf.

Wang, Fei et al. (July 2019). 'Automatic Generation of Synthetic LiDAR Point Clouds for 3-D Data Analysis'. en. In: *IEEE Transactions on Instrumentation and Measurement* 68.7, pp. 2671–2673. ISSN: 0018-9456, 1557-9662. DOI: 10.1109/TIM.2019.2906416.

Wang, Jianqiang, Dandan Ding, Zhu Li, Xiaoxing Feng et al. (Nov. 2021). 'Sparse Tensor-based Multiscale Representation for Point Cloud

Geometry Compression'. In: *arXiv:2111.10633 [cs, eess]*. arXiv: 2111.10633 [cs, eess].

Wang, Jianqiang, Dandan Ding, Zhu Li and Zhan Ma (Mar. 2021). 'Multiscale Point Cloud Geometry Compression'. In: *2021 Data Compression Conference (DCC)*, pp. 73–82. DOI: 10.1109/DCC50243.2021.00015.

Wang, Jianqiang, Hao Zhu et al. (Dec. 2021). 'Lossy Point Cloud Geometry Compression via End-to-End Learning'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.12, pp. 4909–4923. ISSN: 1558-2205. DOI: 10.1109/TCSVT.2021.3051377.

Wu, Cheng-Hao, Chih-Fan Hsu, Tzu-Kuan Hung et al. (2022). 'Quantitative Comparison of Point Cloud Compression Algorithms with PCC Arena'. In: *IEEE Transactions on Multimedia*, pp. 1–1. ISSN: 1941-0077. DOI: 10.1109/TMM.2022.3154927.

Wu, Cheng-Hao, Chih-Fan Hsu, Ting-Chun Kuo et al. (June 2020). 'PCC Arena: A Benchmark Platform for Point Cloud Compression Algorithms'. In: *Proceedings of the 12th ACM International Workshop on Immersive Mixed and Virtual Environment Systems*. MMVE '20. New York, NY, USA: Association for Computing Machinery, pp. 1–6. ISBN: 978-1-4503-7947-2. DOI: 10.1145/3386293.3397112.

Yue, Xiangyu et al. (2018). 'A LiDAR Point Cloud Generator: From a Virtual World to Autonomous Driving'. In: *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*. ICMR '18. Yokohama, Japan: Association for Computing Machinery, pp. 458–464. ISBN: 9781450350464. DOI: 10.1145/3206025.3206080.

Zhirong Wu et al. (June 2015). '3D ShapeNets: A Deep Representation for Volumetric Shapes'. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, pp. 1912–1920. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298801.

Zhou, Qian-Yi, Jaesik Park and Vladlen Koltun (2018). 'Open3D: A Modern Library for 3D Data Processing'. In: *arXiv:1801.09847*.

Zhou, Yin and Oncel Tuzel (June 2018). 'VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection'. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, pp. 4490–4499. DOI: 10.1109/CVPR.2018.00472.

Zou, Chuhang et al. (2017). '3D-PRNN: Generating Shape Primitives with Recurrent Neural Networks'. In: *The IEEE International Conference on Computer Vision (ICCV)*.