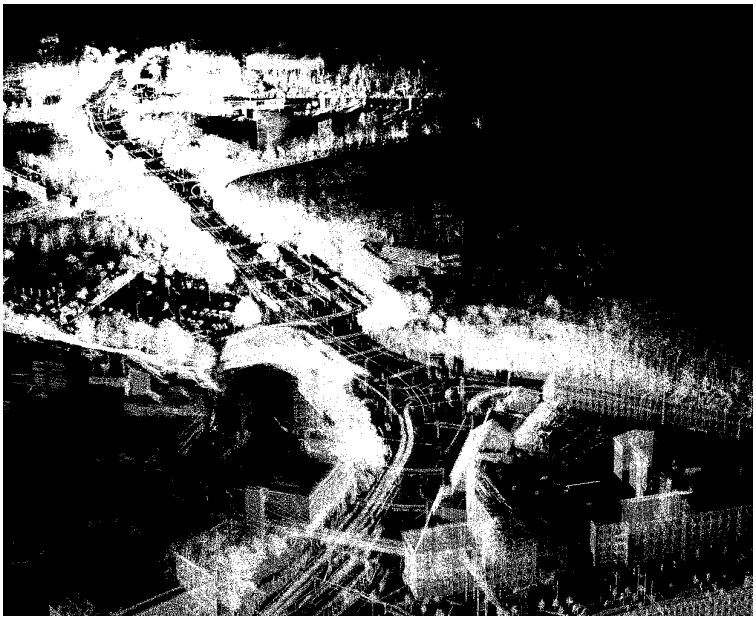


LiDAR Point Cloud Compression for Visualization

Utilizing compression techniques to enable the visualization of a LiDAR point cloud dataset

Master's thesis in Computer Science and Engineering

Sofia Alowersson, Noa Bengtsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

LiDAR Point Cloud Compression for Visualization

Utilizing compression techniques to enable the visualization of a LiDAR point cloud dataset

Sofia Alowersson, Noa Bengtsson

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

LiDAR Point Cloud Compression for Visualization
Utilizing compression techniques to enable the visualization of a LiDAR point cloud dataset
Sofia Alowersson, Noa Bengtsson

Supervisor: Ulf Assarsson, Department of Computer Science and Engineering
Advisor: Frej Karlsson & Fredrik Wennermark, Zenseact
Examiner: Ahmed Ali-Eldin Hassan, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A rendered image from the renderer developed for this project. It is a sequence of LiDAR images merged and downsampled from drive 000010 from the *Zenseact Open Dataset* [1].

Typeset in LaTeX
Gothenburg, Sweden 2026

LiDAR Point Cloud Compression for Visualization
Utilizing compression techniques to enable the visualization of a LiDAR point cloud dataset
Sofia Alowersson, Noa Bengtsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Methods to downsample and compress a large LiDAR point cloud dataset were developed for the purpose of enabling the visualization of a multi-minute sequence in a 3-dimensional `OpenGL` environment. The reduction in data is the result of segmenting dynamic objects and downsampling the LiDAR point clouds either by sampling a number of points in each cell in a 3-dimensional grid, or by computing an average point for each cell in a finer grid or octree structure. The ground plane can also be downsampled separately, with a different resolution, to give visual distinction between ground and environment. Together with a point cloud compression algorithm the final file size of a point cloud that is usable for visualization purposes can be reduced to 0.06%-3.6% of the original file size depending on chosen resolution.

Keywords: LiDAR, Point Cloud, Automotive, Compression, Visualization, 3D Rendering,

Acknowledgements

We want to thank our supervisors at Zenseact, Frej Karlsson and Fredrik Wennermark for valuable insight and help with the project and the thesis writing. Martin Lanzén from Zenseact for comments on the thesis writing. We also want to thank our academic supervisor Ulf Assarsson for help with thesis writing and our examiner Ahmed Ali-Eldin Hassan for ensuring the administrative side of the thesis work went smoothly. Finally, we want to give our thanks to the Vizards team at Zenseact for making us feel welcome at Zenseact during our thesis project.

Sofia Alowersson, Gothenburg, Noa Bengtsson, Gothenburg, 2026-06-15

Contents

List of Figures

List of Tables

1

Introduction

Within the automotive industry, a *Light Detection And Ranging* (LiDAR) camera can be used to capture the 3-dimensional geometry of the world surrounding a vehicle. LiDAR cameras, contrary to typical RGB cameras, can measure the precise location of obstacles in space. The ability to capture 3-dimensional spatial information makes LiDAR a useful tool for automotive self-driving and safety technologies.

A LiDAR camera functions by sending out light rays, then measuring the time it takes for reflections to return to calculate the distance to the ray's reflection point. The resulting data from one reflection can be used to calculate a 3-dimensional point [2]. Sending out thousands of these rays in multiple directions results in a collection of points that represent the world around the camera. A collection of these points can be called a LiDAR *point cloud* or a LiDAR *frame*.

In order to capture a given environment, a large amount of these LiDAR frames are collected at different timestamps, for example as a vehicle drives around an area. This can result in huge sets of data, consisting of potentially millions of points for each second of runtime [1]. Because of this, being able to reduce the data overhead, through compression or other techniques, is vital to utilize the dataset.

It can be of interest for the automotive industry to have a tool that can visualize multiple LiDAR frames in a 3-dimensional environment. We propose methods, and a proof-of-concept software that enable the visualization of a multi-minutes dataset of LiDAR frames in a common world-space. Visualization is the goal, which also enables the ability to compare the benefit of reducing the visual fidelity, for a decrease in the memory requirements of the visualizer. A smaller memory requirement will increase how large of a dataset can be visualized. Additional features such as segmentation of dynamic objects and ground should be implemented to remove visual noise.

The *Zenseact open dataset* (ZOD) is the primary dataset that the methods and software will be developed for and evaluated on. The ZOD contains 29 multiple-minutes sequences of a car driving through various environments. Each sequence contains LiDAR frames captured at a frequency of 9 Hz with corresponding *poses* (position and orientation) which position these frames in relation to each other [1].

One of the challenges to visualize the ZOD is the size of the data. A drive typically has 1000-2000 frames, with each frame containing on average $\approx 254\ 000$ points [1], this is in contrast to the famous KITTI dataset, mentioned in many papers (e.g.

[3], [4], [5], [6], [7], [8]) exploring methods concerning LiDAR data, which has an average of ≈ 120 000 points per frame [9] (captured at 10 Hz), roughly half of the ZOD.

1.1 Limitations

To achieve a successful merge of frames good poses are needed, it is beyond the scope of this project to implement a good *Simultaneous Localization and Mapping*, (SLAM) method that creates poses by analyzing LiDAR point clouds. The ZOD has good poses up to an accuracy of 0.01 m [1]. Issues could appear in cases where the LiDAR capturing car drives in a tunnel or similar area where the GPS accuracy decreases, which in turn creates bad poses. This is a problem which could be mitigated with SLAM techniques. Defects in the LiDAR dataset such as issues due to environmental conditions such as rain, snow are ignored. Issues such as reflections and windows causing LiDAR points to be not image the environment correctly are also ignored.

1.2 Similar work

There are existing techniques for point cloud compression, but they are not optimal for a merged point cloud which is necessary in this project. For example there are techniques which project a 3-dimensional point cloud onto a 2-dimensional raster plane to achieve good and fast compression [10]. This is not applicable to merged point clouds, as a merged point cloud can not be projected onto a plane as different points are captured from different perspectives.

There are also 3-dimensional methods which does not depend on projection, and should still work for merged point clouds [11]. This is could be used as an supplementary method for storing files on disk, but other techniques would need to be used to reduce the actual loaded data size of a merged LiDAR dataset, to enable the visualization of it.

1.3 Ethical and Environmental Considerations

There are no ethical concerns in this thesis, the methods introduced can not be used to harm or negatively affect any human or animal. Issues of privacy are not relevant, since the LiDAR images used are not detailed enough to be able to discern the identities of the people captured by the LiDAR camera.

The methods do not require large amounts of computational resources, contrary, it actually reduces the required storage-space and the necessary GPU performance to render a scene. The reduction in data could also enable the streaming of LiDAR data from a centralized storage location to rendering clients, which could eliminate redundant copies of the data, further decreasing the total used storage-space. Collectively, this all results in an improvement in environmental effects.

2

Theory

2.1 World partitioning

In both visualization and compression, world partitioning methods such as grids and octrees can be used [12], [13], [14]. World partitioning lets algorithms more efficient and to create a structure to guide the compression algorithms.

2.1.1 Grids

3-dimensional grids, shown in figure 2.1, serve as an easy-to-implement structure, consisting of splitting the region in equally sized regions. They are typically fast to build, but do however scale cubically in relation to their region size.

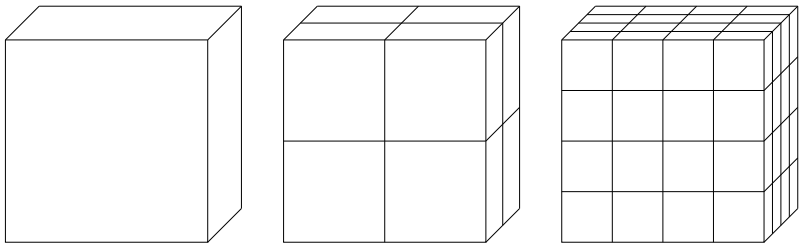


Figure 2.1: 3-dimensional grid structure of equally sized regions.

2.1.2 Octrees

An octree, illustrated in figure 2.2, is a hierarchical tree-structure used to partition 3-dimensional spaces. Octrees try to exploit sparsity within a structure. The algorithm begins with first placing the world in cubed region. Then the cube is split into 8 sub-cubes, split along the larger cube's middle. For each sub-cube, it is further subdivided only if the region contains data. This is recursively done until all data has been placed into their respective leaf node cube, at some user-given depth [15].

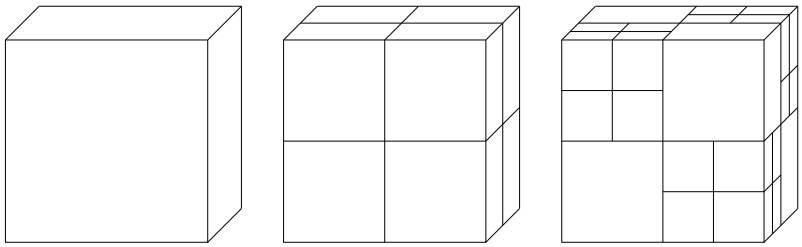


Figure 2.2: Octree structure at three different steps of recursion.

2.2 Downsampling and Data pruning

An aspect unique to the task is that a large degree of data pruning is not only acceptable, but encouraged, provided that the visual clarity of the data is kept or possibly even improved. This means that not only is lossy compression appropriate, more crude downsampling methods can be used as an early quantization method. Downsampling is typically used to reduce data for computationally exhaustive work, and there are a number of possible avenues to consider, for example, random-sample-based, grid-based, deep-learning-based, cluster-based, and voxel-based methods [14].

Grid-based methods are the most straightforward. For the purposes of this paper, it is any method performed for each cell in a grid, for example sampling random points from a cell (sec. 3.7.2) or taking the average point in a cell (sec. 3.7.3).

2.3 Compression

Compression can broadly be separated into two categories. Lossless and lossy compression. Lossless compression involves turning the set of data into a smaller, more compact, representation, but does not remove any actual information about the data. Lossy compression however, does not require the data to remain completely intact, which can be beneficial if exact precision is not crucial. Lossy compression methods typically makes use of lossless methods as well.

When working with compression, there is a set of common steps in any compression pipeline to take note of, though the order may differ.

Firstly any compression algorithm is divided in two, encoding and decoding. Encoding is representing the data in some way that takes up less space, and decoding is to reconstruct the data from the encoded representation. These mostly mirror each other. The encoding step can be further divided into: Mapping/Transforming, Quantization and Coding.

Mapping/Transforming refers to any act of reordering or rearranging the data to fit another data structure. This can be reordering the data into a grid, performing a fourier transform, or sorting the data in some other way [16].

Quantization is any method of removing excess/redundant information, and is exclusive to lossy compression algorithms. It is the only step which is not fully reversible,

consequently, the decoder may need reconstruct data in inverse quantization step.

Finally, coding refers to lossless methods to represent the bits in another way to save space, such as Huffman or arithmetic encoding.

2.3.1 Octree compression

Octree-based compression is a proven method used to compress LiDAR data. The G-PCC (Geometry Point cloud Compression) standard by the MPEG corporation, details a step-by-step process on how to compress data in this manner. It is made up of two parts, the S-PCC (Surface point Point cloud Compression) standard, which specifically targets and compresses point clouds meant to represent dense surfaces, and the simpler L-PCC (LiDAR Point cloud Compression) standard, which utilizes octrees as a base method. [11].

3

Methods

This chapter explains methods and techniques that are used to enable the visualization of a large LiDAR point cloud dataset.

Our project is split into three parts:

1. *Ground and dynamic object segmentation*

   It is necessary to identify the ground points, and the points belonging to dynamic objects that move between LiDAR frames. The segmentation information can be used to remove dynamic objects, and to modify the ground plane for a more clear visualization.

2. *Subsampling and compression process*

   This part merges the point cloud frames into coarse tiles and subsamples them to a given accuracy, this effectively reduces the amount of points. This new subsampled point cloud can then be further compressed by point cloud compression techniques to reduce the storage footprint on disk.

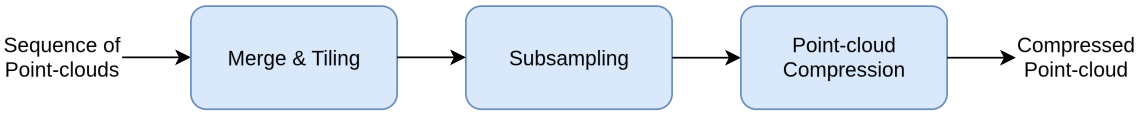


Figure 3.1: A flowchart of the compression pipeline.

3. *Visualization*

   Finally, the visualizer takes these compressed point clouds on disk, and visualizes them in a 3-dimensional real-time environment.

The project is implemented in `C++`, methods concerning point clouds are developed using functions defined in the *Point Cloud Library*, PCL [17], the renderer is implemented in *OpenGL*, *CUDA* is used for compute acceleration and the application is developed and tested for a Linux desktop system.

3.1 From camera-space to world-space

Each point captured by the LiDAR camera is given in coordinates defined in camera-space, the coordinate system which has its origin at the center of the camera. To

visualize all frames in a common space, called world-space, each frame must be transformed into the same coordinate system.

The ZOD [1] has for each frame a given *pose* (position and orientation) relative to a common origin. The pose is expressed in homogeneous coordinates in a 4×4 matrix.

Given a vector $\mathbf{u} = (x, y, z, 1)^\intercal$ it is transformed to the position and rotation given by a pose, $\mathbf{P}$, with a vector-matrix multiplication.

$$\mathbf{P} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

r_{ij} are the components that applies a rotation transform, and the components t_k are translation. The operation $\mathbf{Pu}$ applies the rotation defined by r_{ij} and the translation defined by t_k, to $\mathbf{u}$, in a single operation.

The ZOD poses are defined from the rear-wheel axle of the LiDAR capturing car [1]. All poses are transformed to the position of the camera on the vehicle to ensure that the LiDAR points are all defined as having the origin in the LiDAR camera. Henceforth, all poses point to the position where the LiDAR camera was when a frame was captured. The coordinate system for the LiDAR points are also aligned such that the z direction points upwards in the real world and the origin is the LiDAR camera.

3.2 Overlapping Point clouds

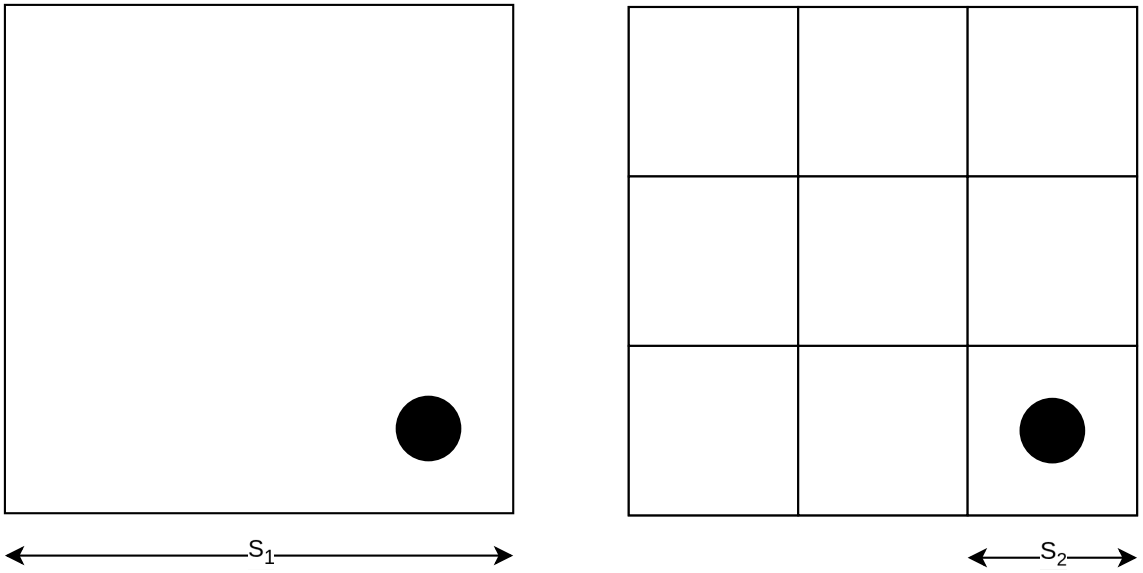
A consequence of placing all LiDAR frames in the same world-space is that the density of points is much higher. This is due to the frequency of the captured LiDAR frames is high enough to cause a substantial overlap of the geometry that is captured. The results of this is explored in section 4.1.

By limiting the maximum point density for the world point cloud, a large amount of data can be pruned. This could achieve a similar effect to simply combining every fifth or seventh or tenth frame without losing the information of the in-between frames that are skipped.

3.3 Point cloud density

We propose a method to compare the density of different point clouds. The density is determined in two dimensions to make visualization and computation easy. The density for a point cloud is determined for each cell in a 2-dimensional grid. The 2-dimensional grid is superimposed on the xy-plane, and the density in a cell is computed as

$$\rho = \frac{\#\text{points in cell}}{S^2} \tag{3.2}$$



(a) A region with one point covered by one cell.

(b) A region with one point covered by nine cells.

Figure 3.2: Different cell sizes can over- or underestimate the density.

, S is the cell size, given as the side length of the cell.

Empty cells are ignored, as this effectively results in a grid structure that only contains the point cloud.

This measure can be useful to compare different point clouds when using the same cell size. The density measure is not an accurate measure to determine the amount of points in a given area.

The inaccuracy of the measure is the tendency to overestimate or underestimate the density due to the points being sparsely distributed in some areas. Consider figure 3.2, where the left figure with a cell with one point has a density of $\rho_1 = \frac{1}{S_1^2}$, while the right figure has a cell with one point and density $\rho_2 = \frac{1}{S_2^2}$. Assuming $S_1 = 3S_2$, we get $\rho_2 = 9\rho_1$. One point in both cases, but nine times higher density in the second case. Due to sparse points the density changes dramatically depending on the cell size.

The measure is still useful to describe the overlapping nature of point clouds. Two point clouds capturing the same environment, with a small difference in pose, should have roughly double the average cell density if measured together, rather than individually, if measured using the same cell size.

3.4 Segmentation of the ground

Segmentation consists of identifying which points in a LiDAR frame is identified as ground. There exists multiple methods but a fast, simple and fairly good method is to adapt the first steps shown in *Fast Ground Segmentation for 3D LiDAR Point*

Cloud Based on Jump-Convolution-Process [5].

The process begins by taking a frame, $F = \{\mathbf{p}^0, ..., \mathbf{p}^i, ...\}$, containing points $\mathbf{p}^i$ and depositing all the points into a ring-shaped map containing multiple bins (figure 3.3). Each ring segment is ΔR long, and each angular section is $\Delta\alpha$ wide. The points are deposited as

$$B_{i,j} = \{\mathbf{p} \in F \mid i \cdot \Delta R \leq \sqrt{\mathbf{p}_x^2 + \mathbf{p}_y^2} < (i+1) \cdot \Delta R, \\ j \cdot \Delta\alpha \leq \text{atan2}(\mathbf{p}_y, \mathbf{p}_x) < (j+1) \cdot \Delta\alpha\} \tag{3.3}$$

, $B_{i,j}$ are the bins and $\mathbf{p}$ is a point, with coordinates $\mathbf{p}_x, \mathbf{p}_y$ and $\mathbf{p}_z$, in frame F. atan2 is the 2-argument arctangent function.

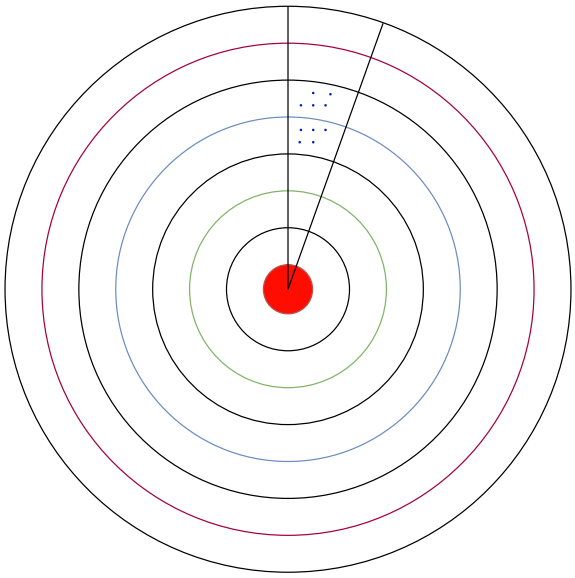


Figure 3.3: Ring-shaped map consisting of multiple bins.

Ground points are identified as the points in a bin that are within a threshold of the lowest height value in a bin. The set of ground points G in a frame is

$$G = \bigcup_{\text{All bins}} \{\mathbf{p} \in B_{i,j} \mid \mathbf{p}_z < E(B_{i,j}) + \tau\} \tag{3.4}$$

, where τ is a threshold parameter typically set to around 0.2 m and $E(X)$ is defined as the z value of the point with the lowest z value in a set X.

This method fails where an obstacle with a flat top, such as a car, is placed on the ground. This would cause the roof of the car to be labeled as ground. To mitigate this an improvement is made to $E(X)$.

A gradient $k = \arctan(\frac{E(B_{i,j}) - E(B_{i-1,j})}{\Delta R})$ between bins is computed.

Given k an adjustment is done to $E(X)$.

$$E'(B_{i,j}) = \begin{cases} E(B_{i,j}), & k \leq \delta_{\max} \\ E(B_{i-1,j}) + \Delta R \cdot \delta_{\max}, & k > \delta_{\max} \end{cases} \tag{3.5}$$

$\delta_{\max}$ is a parameter set as the greatest slope of the road. This adjustment simply linearly extrapolates the $E(X)$ value from the previous bin if the slope between two bins were too large.

Finally the set of ground points are

$$G = \bigcup_{\text{All bins}} \{\mathbf{p} \in B_{i,j} \mid \mathbf{p}_z < E'(B_{i,j}) + \tau\} \tag{3.6}$$

3.5 Segmentation of dynamic objects

Dynamic objects introduce a lot of visual clutter to the scene when viewing many frames together. A visual example is shown in figure 3.4.

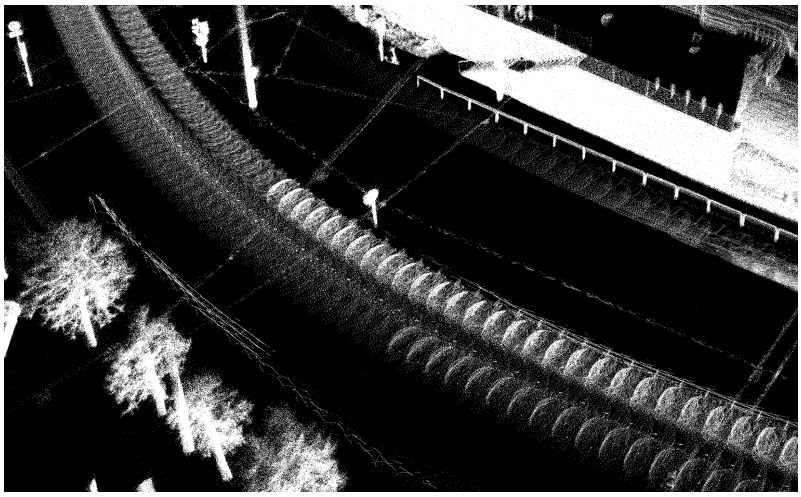


Figure 3.4: Cars "smeared" out across multiple frames. The picture is created from a visualization of LiDAR frames 1050 to 1150 from drive 000010 included in the Zenseact Open Dataset [1].

The method for removing dynamic objects is based on the paper *ERASOR: Egocentric Ratio of Pseudo Occupancy-Based Dynamic Object Removal for Static 3D Point Cloud Map Building* [4]. The method assumes that each frame has a high quality pose associated with it.

The idea behind this method is to compare the region around a given pose in world-space, with the same region around the same pose, but with individual frames. The region in world-space will include the points of all frames, potentially including dynamic objects that appear in different locations in different frames. By comparing this region from world-space with individual frames, one can determine if an area was, at some point in time in some frame, not occupied by an obstacle. If this area was free from obstacles in a frame but not in world-space, then that area must contain a dynamic object, and can be segmented accordingly. Figure 3.5 shows how a collection of frames only containing two obstacles in each frame can result in a world-map containing many obstacles. It also shows how discrepancies between frames can inform which obstacles are dynamic.

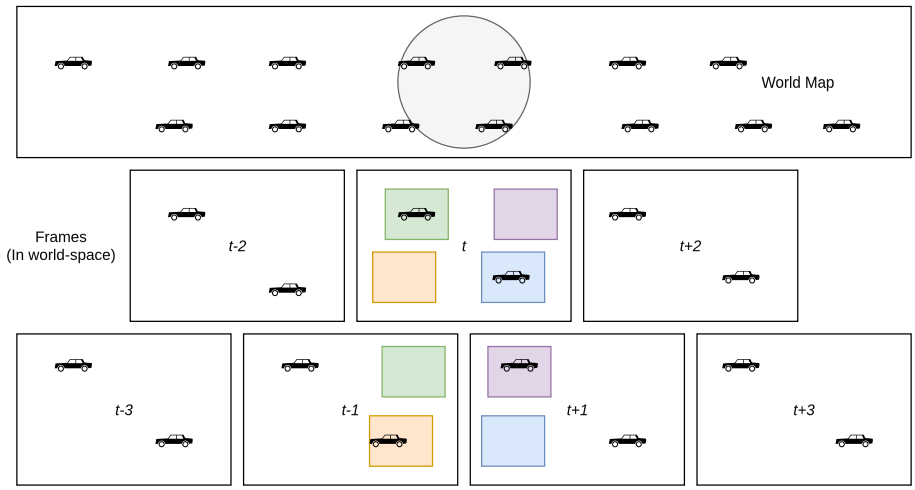


Figure 3.5: The seven lower boxes are illustrations of what obstacles a LiDAR frame could have captured at poses $\{t-3, ..., t+3\}$. Each frame only contains two obstacles, but when combined in the world-map, it results in the obstacles appearing many times at different locations. The circle in the world-frame shows a region in world-space where the information of LiDAR frames captured at poses $\{t-1, t, t+1\}$ can be used to determine which areas were obstacle at some point in time. The colored regions shows locations that were obstructed in one frame, and free in another frame.

Assume a frame $F_t^C = \{\mathbf{p}^0, ..., \mathbf{p}^i, ...\}$, captured in camera-space at timestamp t with points $\mathbf{p}^i$. This frame has an associated pose $\mathbf{A}_t$ given as a homogeneous 4×4 transformation matrix that transforms the frame into world-space.

The world map W can be expressed as

$$W = \bigcup_{t \in T} F_t^W \tag{3.7}$$

, where $F_t^W = \bigcup_{\mathbf{p} \in F_t^C} \mathbf{A}_t \cdot \mathbf{p}$, is the frame, F_t^C, in world-space and T is the set of all timestamps.

To efficiently compare regions around a specific pose a *volume of interest*, VOI, ν, is created. The VOI around a point, $\mathbf{q}$ is defined as

$$\nu(P, \mathbf{q}) = \{\mathbf{p} \in P \mid \sqrt{(\mathbf{p}_x - \mathbf{q}_x)^2 + (\mathbf{p}_y - \mathbf{q}_y)^2} < L_{\max},\ h_{\min} < \mathbf{p}_z - \mathbf{q}_z < h_{\max}\} \tag{3.8}$$

, P is a set of points, $L_{\max}$ is the maximum radius of the extent of the VOI, $h_{\min}$ and $h_{\max}$ controls the extent of the VOI in the z dimension. By assuming that all dynamic objects are located within $h_{\min}$ and $h_{\max}$, the total amount of points that are potentially dynamic are reduced, which reduces the computational burden. A GPU implementation was used to speed up the creation of VOIs. If a GPU is not available a kd-tree nearest neighbor algorithm is used instead [18].

Then for a VOI, bins $B_{i,j}$ are created and filled as defined in eq. (3.3). For each bin the maximum height difference, Δh, is recorded.

$$\Delta h = \max(z(B_{i,j})) - \min(z(B_{i,j})). \tag{3.9}$$

, where $z(\cdot)$ takes the z coordinates from a set, as a set. This set of bins is called a *Region-Wise Pseudo Occupancy Descriptor* or R-POD.

To remove points around a pose $\mathbf{A}_t$ in the world-map W, a VOI, $\nu(W, \varrho(\mathbf{A}_t))$ is created around that pose. $\varrho(\mathbf{A})$ returns the position given from a pose. To center $\nu(W, \varrho(\mathbf{A}_t))$ around the origin, the VOI is transformed into frame-space with $\mathbf{A}_t^{-1}$. VOIs are then also created around the same pose for a set of individual frames that are captured close to the pose $\mathbf{A}_t$, e.g $V = \{\nu(F_{t-i}^W, \varrho(\mathbf{A}_t)), ..., \nu(F_{t+i}^W, \varrho(\mathbf{A}_t))\}$. Then all VOIs in V are transformed with $\mathbf{A}_t^{-1}$ to ensure that all points are compared around the same origin. It is beneficial to center everything around the origin to enable easy creation of the R-PODs

This yields a set of VOIs, V, created from frames concerning the same region as the VOI created from the world-map $\nu(W, \varrho(\mathbf{A}_t))$. Then for each VOI an R-POD is created. These bins will concern the same volume in space, but from the perspective of individual frames, and from the perspective of the world-space.

Each bin created in the world-space R-POD is compared to the bin in each frame's R-POD. If the ratio $\xi = \frac{\Delta h_F}{\Delta h_W}$, is smaller than a threshold ξ_{th}, (typically $\xi_{\text{th}} = 0.2$ [4]) then all points in that bin is classified as dynamic. Here Δh_F and Δh_W is the maximum height difference for a R-POD bin in a single frame and in the world-map, respectively.

This process is then repeated for every pose, $\mathbf{A}_t$, to segment dynamic objects for the entire world-map. The size of the set V must be sufficiently large to ensure that there are frames that "see" every static space of the environment.

3.6 Dynamic object segmentation for large datasets

The dynamic object segmentation algorithm (refer to section 3.5) requires both LiDAR frames in camera-space, and the merged frames in world-space. Loading the frames and creating the world-space for a complete drive requires up towards 10-20 GB of memory. This can especially be a problem if the point cloud needs to be loaded on the GPU to speed up calculations, as the GPU memory is often insufficient. To mitigate this a rolling window process is proposed.

Under the assumption that the dynamic object segmentation around pose A_i requires no information from a frame captured at pose A_{i+k}, then the frames associated at these poses does not need to be loaded at the same time. Consequently, these two frames can be done independently of each other.

The dataset is split up into sections, called windows which fit into the memory of the host computer. But the windows will have to be overlapping, as the final frame in a window will only have frames before itself were captured. Therefore the windows are interleaved with each other to ensure that all frames have enough information to make a good segmentation. Figure 3.6 shows an example of the interleaved windows.

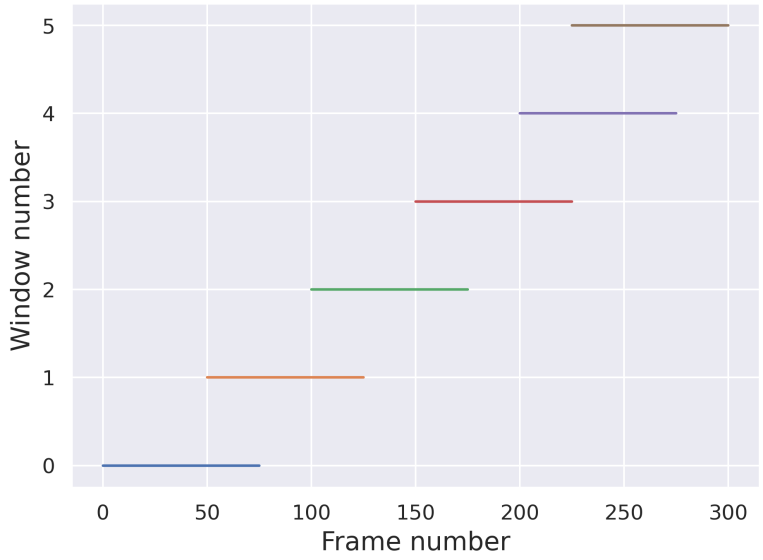


Figure 3.6: Interleaved windows to ensure enough frame information to segment dynamic objects.

The suggested segmentation of points from each window are merged by always segmenting a point as dynamic if any window recognized it as such.

This creates a process than can load an arbitrary large dataset, provided that the memory required for processing a single window fits in the host computer's memory.

3.7 Downsampling point clouds

We propose several methods for downsampling. The main ones are: random sample grid, average point grid and hybrid grid-octree average (often referred to as octree average in this paper). The downsampling will be evaluated on number of removed points, runtime and visual quality.

3.7.1 Grid structures

In order to downsample effectively, a mixture of grids are used. A grid with large cells are created first, then for each large cell a new grid structure is created with smaller cells. This is because generating and iterating over small cells at once quickly becomes unmanageable from a memory-standpoint. Thus, the grid is first split into a coarse grain grid, with finer grids inside. This partitioning into coarse grids creates a natural opportunity to parallelize the downsampling process as each algorithm operates on the fine grain grid structure level. The size of the fine grid size becomes the *resolution* of the downsampling algorithm.

Additionally, in order for future work to be able to efficiently support multiple downsampled versions of the point cloud at once, as a form of level-of-detail implementation, multiple coarse grain cells are put together to form tiles. Each tile

contains the same area but at different degrees of fidelity, ready to be switched out dynamically. This tiling is done in 2 dimensions, rather than 3 dimensions, as it has no connection or effect to the downsampling algorithms. This should not affect the final result, but it could create artifacts along tile edges and affect runtime.

All grid methods are implemented by first generating a grid based on a cell-size, then by indexing each point. In the case of the 3-dimensional grid, this is done by taking its x, y, and z-coordinates, shifting them to positive values, and dividing by the amount of cells in each direction. The cell size is considered an important factor, and different cell sizes will be evaluated.

The number of cells in each direction is noted as n_x, n_y, n_z and determined as

$$n_x = \frac{\max(x(W))}{S} \tag{3.10}$$

$$n_y = \frac{\max(y(W))}{S} \tag{3.11}$$

$$n_z = \frac{\max(z(W))}{S} \tag{3.12}$$

, where S is the cell size, $x(\cdot),\ y(\cdot),\ z(\cdot)$ takes the x, y, z coordinates from a set, as a set, respectively. W is collection of points.

A set of cells, G are created

$$G = \{C_{ijk} | 0 < i \leq n_x,\ 0 < j \leq n_y,\ 0 < k \leq n_z\}. \tag{3.13}$$

Each point $\mathbf{p} \in W$ is deposited into cell C_{ijk} by calculating the indices as

$$i = \text{floor}(\frac{\mathbf{p_x}}{S}) \tag{3.14}$$

$$j = \text{floor}(\frac{\mathbf{p_y}}{S}) \tag{3.15}$$

$$k = \text{floor}(\frac{\mathbf{p_z}}{S}) \tag{3.16}$$

.

This yields the set G containing all points in W but partitioned into cells ordered in a grid structure.

3.7.2 Random sample grid

Random sampling requires all points to be stored in their respective cell in the grid. Then iterating over all cells containing points, it samples a fixed amount of points uniformly. If the cell already contains fewer than the given sample count, it samples the same amount of points the cell contains [14].

The set of sampled points, C', in a cell can be expressed as

$$C' = \bigcup_N (\mathbf{p} \in_R C) \tag{3.17}$$

, C is the set of all points, $\mathbf{p}$, in a cell, and N is a set with cardinality equal to the number of points that should be sampled or the number of points in a cell, whichever is smaller. $\in_R$ denotes that an element is selected at random from a set. The process is illustrated in figure 3.7.

The number of sampled points per cell will be adjusted according to density of points sampled compared to the density of the original point cloud.

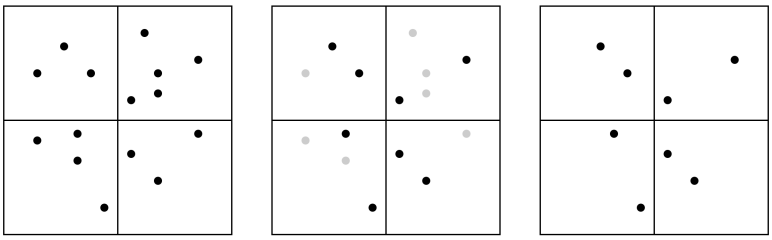


Figure 3.7: Visual example of random sampling in a grid.

3.7.3 Average point grid

Averaging creates one point per cell. In our case all points are first deposited in their respective cells. Once stored, the average point is calculated for each cell [14]. This is illustrated in figure 3.8.

For each cell the average point of a cell is determined as

$$\mathbf{p}_{\text{mean}} = \sum_{\mathbf{p} \in C} \frac{\mathbf{p}}{|C|} \tag{3.18}$$

, C is the set of all points, $\mathbf{p}$, in a cell, $|C|$ is the cardinality of the set.

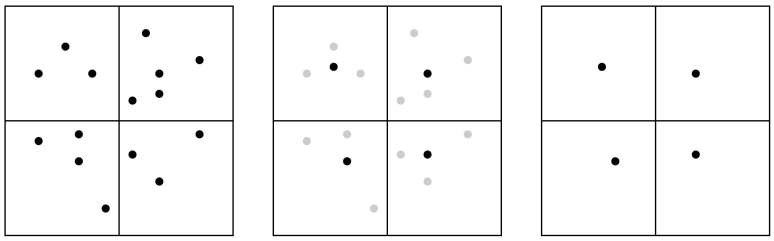


Figure 3.8: Visual example of average sampling sampling in a grid.

3.7.4 Octree-based average point

Octree-based averaging is the same as average point grid , only the data structure used to store and iterate over the cells (or "leaves" in this case) is an octree structure rather than a grid [13]. The purpose of testing this method is to possibly allow the same octree to be used when implementing point compression, as well as serving as an alternative way to decrease the grid generation cost when dealing with very small cell sizes. This method will often be referred to simply as octree downsampling.

However, in order to combat the extreme memory requirements when dealing with larger datasets, the octree is still employed in tandem with with the coarse grain grid-structure. That is to say generating an octree at each coarse grid cell. This allows for lower depth sizes for the given octree, possible parallelization of work, as well as splitting the task into separate chunks, so that less of the data has to be processed at once.

3.7.5 Estimating optimal cell-size and point count

The density measures (sec. 3.3) can be most effectively used when looking at the random-sample method, as it provides an easy way to set up a fixed upper limit density given a cell size. This can give an indication of both optimal cell sizes and point count given a cell. However, as we use 2-dimensional cells for calculations, they can only give a vague idea.

For the other methods, where the result is several points being collapsed to a single point, it is instead better to consider just the point count per cell calculations in order to estimate the most optimal cell-size.

3.8 Downsampling large datasets

The PCL library [17] has efficient methods to work with point clouds. It is employed to assist with all methods. The average sample has a voxel-grid based approach. For all methods a rough coarse grid structure is used as a base, and then the octree averaging or the grid averaging is applied on each rough coarse cell. The rough coarse grid approach limits the data input when utilizing the provided functions.

Following the same logic as in 3.5, the dataset cannot be fully loaded at once. Thus another rolling windows process is used. However, unlike in segmentation, the windows are not overlapping. For each window, the coarse-grid structure is applied. Finally, once all windows have been downsampled, they are merged and downsampled a final time.

3.9 Storage Compression

The PCL library [17] is used in an additional final compression when storing the files to disk. An octree compression is applied to the already subsampled point cloud, then the compressed cloud is saved to disk. This final disk representation will be stripped of visually unnecessary information such as, timestamps of points, intensity of points and which sensor captured a point to reduce the final data size as much as possible.

4

Results

All methods were evaluated on a *HP ZBook Fury 15 G7* laptop with a 16-core Intel Core i9-10885H, 32 GB of memory and a Nvidia Quadro RTX 4000. The laptop was running Ubuntu 24.04.4 LTS.

The methods were evaluated on drives 000000, 000004, 000008 and 000010 from the *Zenseact Open Dataset* [1], which contains a diverse set of environments. Table 4.1 presents the locations of each drive. Drive 000000 is a highway environment with many cars. Drive 000004 includes a drive over a bridge with some elevation change included. Drive 000008 is a slow drive through a city environment with lots of points. Drive 000010 is a medium fast drive through a city environment with trams and cars.

Table 4.1: Summary of the locations of a few drives.

Drive ID	Location	Mean points per frame
000000	Highway around Paris	$239.5 \cdot 10^3$
000004	Älvsborgsbron, Gothenburg	$199.7 \cdot 10^3$
000008	Järntorget and Oscar Fredriks Church, Gothenburg	$287.0 \cdot 10^3$
000010	Sahlgrenska to Wavrinskys, Gothenburg	$266.8 \cdot 10^3$

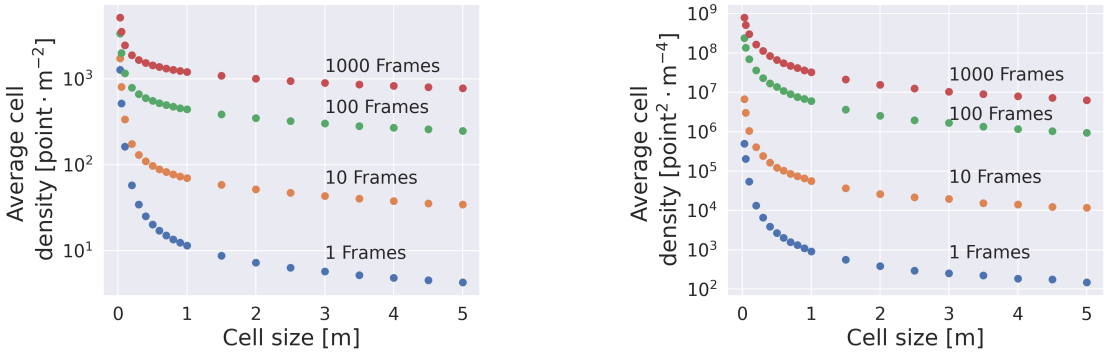
The ground segmentation was initialized with parameters $\Delta R = 2\,\text{m}, \Delta\alpha = 0.4°$, while the dynamic object segmentation had $\Delta R = \frac{1}{3}\text{m}, \Delta\alpha = 2.4°, L_{\max} = 30\,\text{m}, h_{\max} = 4.3\,\text{m}, h_{\min} = -3.7\,\text{m}$. The threshold ratio to determine if a region is dynamic, $\xi_{\text{th}} = 0.2$.

The downsampling parameters were set as: The tile-structure used for downsampling-maps were set to 50 meters. Each window loaded 50 frames at a time in the initial pass. For the coarse grid tiles, the size is set to be 32 times the given resolution (fine cell size) for both of the average-based methods. Since the random sample method used larger cell sizes overall, it did not need to make use of the internal coarse-grain structure, other than the downsample-map tile structure.

4.1 Density of points

The density of points were determined for a grid of cells as described in section 3.3. The average cell density was plotted, in figure 4.1, for different cell sizes and for

point clouds created from various amount of frames.



(a) The average cell density for different cell side-lengths, and different number of frames.

(b) The variance of the cell density for different cell side-lengths, and different number of frames.

Figure 4.1: Average cell density and variance of cell density. Measured starting from frame 0 from drive 000010 included in the Zenseact Open Dataset [1].

Figure 4.1a shows the problems of over- and underestimating the amount of points in a cell as described in section 3.3. But comparing the density difference for different point clouds measured with the same cell size reveal that there is a large overlap of points, as the density, for all cell sizes, increases when the number of frames in sequence increases.

This is further supported when plotting the density cells as an image. Figure 4.2 shows the same location with a different amount of frames to build the world point cloud. It is clear that there is a strong overlap between frames and that the density of points in one area increases with the amount of frames.

4.2 Segmentation

The runtimes and the effectiveness of ground and dynamic object segmentation was measured and evaluated.

4.2.1 Ground Segmentation

The runtime for the ground segmentation, processed on different drives is shown in table 4.2. There seem to be some correlation with the average execution time of a frame and the mean points per frame.

Table 4.3 shows that $20 - 30\%$ of the points are typically recognized as ground. There does not seem to be a clear correlation in how many points are labeled as ground, and which environment the drive was captured in.

The ground segmentation does accomplish its goal of removing the ground. Figure 4.3a shows a render where the ground plane has been removed. Although, it does

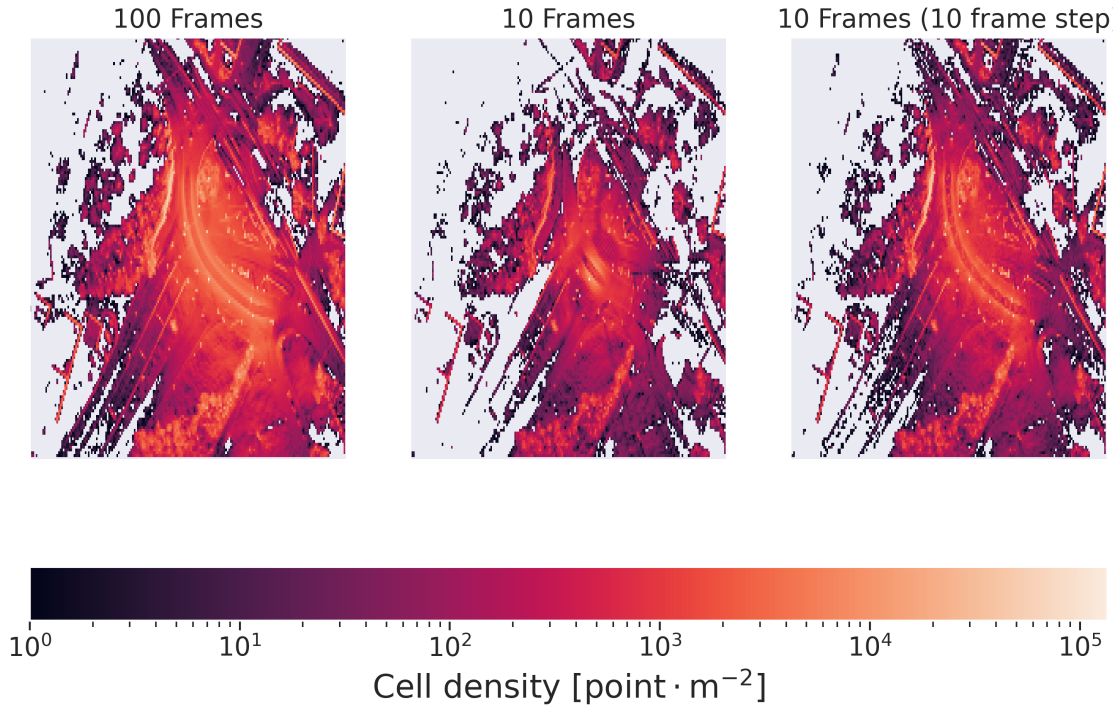


Figure 4.2: The density as a heatmap with 100 frames (left), 10 frames (middle) or 10 frames with 10 frames between every frame (right), around frame 2330 from drive 000010 included in the Zenseact Open Dataset [1]. The cell size was 1 m.

Table 4.2: The average time to segment the ground from a frame for different drives. The average speed of the car in a drive and the mean points per frame is also given.

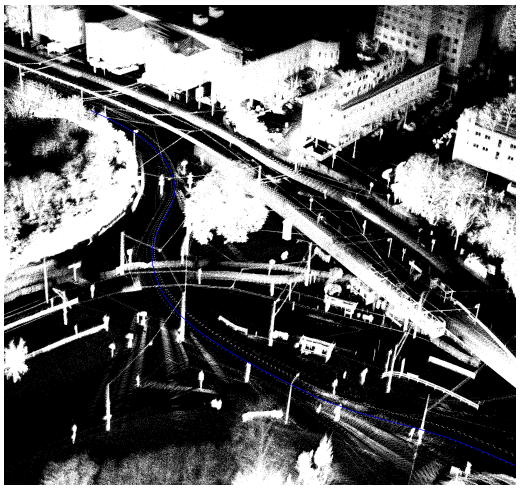
Drive ID	Mean time to segment one frame	Mean speed	Mean points per frame
000000	4.241 ms	85.14 km/h	$239.5 \cdot 10^3$
000004	3.746 ms	61.30 km/h	$199.7 \cdot 10^3$
000008	4.646 ms	18.32 km/h	$287.0 \cdot 10^3$
000010	4.715 ms	29.73 km/h	$266.8 \cdot 10^3$

Table 4.3: Point statistics for ground segmentation.

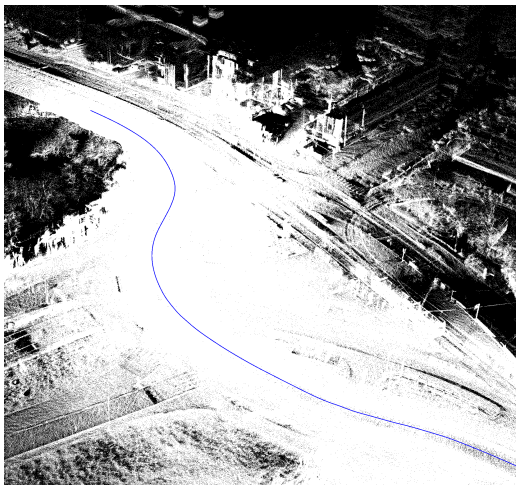
Drive ID	Points	Ground Points	Percentage Ground Points
000000	$6.86 \cdot 10^8$	$1.97 \cdot 10^8$	28.74 %
000004	$2.84 \cdot 10^8$	$7.71 \cdot 10^7$	27.14 %
000008	$5.69 \cdot 10^8$	$1.17 \cdot 10^8$	20.51 %
000010	$6.44 \cdot 10^8$	$1.99 \cdot 10^8$	30.87 %

extract the ground, it also incorrectly labels other static points as belonging to the ground.

Figure 4.3b shows how a lot of points are still included in the ground segmentation when they clearly are not part of the ground. For example parts of the building in the top of the image and trees in the left side of the image. But, this overestimation of the ground points does not seem to create any noticeable artifacts or holes



(a) A render of the points not labeled as ground.



(b) A render of the points labeled as ground.

Figure 4.3: Drive 000010 included in the Zenseact Open Dataset [1]. The blue path indicates the driven path of the LiDAR capturing car.

in the environment rendered in figure 4.3a. Regardless of overestimation, it does successfully segment the road, which is important for visual clarity.

4.2.2 Dynamic object segmentation

The average runtime of identifying dynamic objects around a single point seem to be related to the average speed that the LiDAR capturing car is traveling at. Table 4.4 shows how drive 000008 and 000010 is notably slower to process than drives 000000 and 000004. From looking at memory utilization it seems that the memory on the machine was insufficient and instead memory pages had to be shuffled into swap memory. The poses and frames are captured on the vehicle at a fixed rate, therefore the density of points around a pose in the merged world-frame increases if the car is moving at a slower speed, compared to a faster speed.

Table 4.4: The average time to segment around a pose for different drives. The average speed of the car and a cell density measure (1.0 m cell size) is also given.

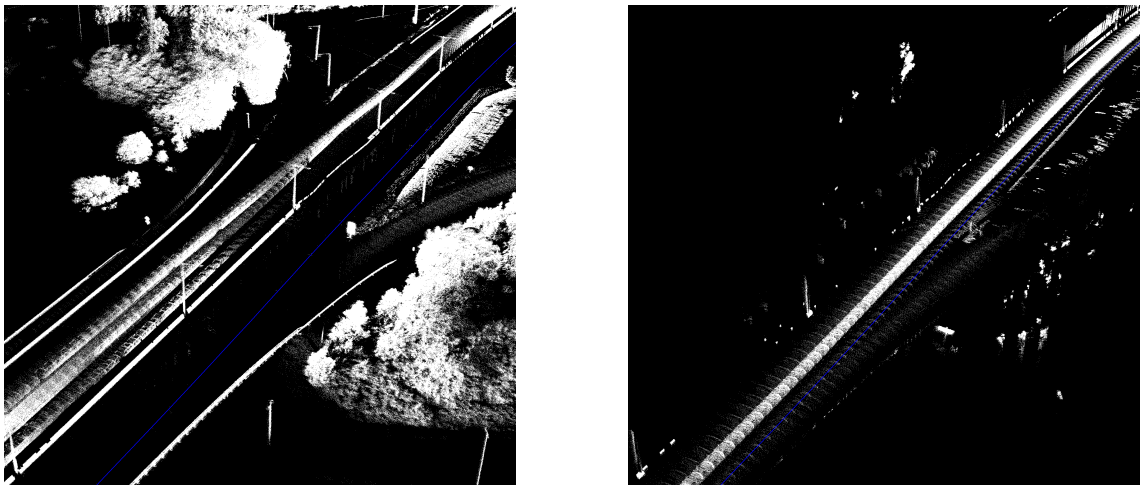
Drive ID	Mean time	Mean speed	Mean cell density	Mean points per frame
000000	0.61 s	85.14 km/h	896.6	$239.5 \cdot 10^3$
000004	0.63 s	61.30 km/h	881.9	$199.7 \cdot 10^3$
000008	7.70 s	18.32 km/h	4351.6	$287.0 \cdot 10^3$
000010	2.75 s	29.73 km/h	1582.6	$266.8 \cdot 10^3$

From table 4.5 it seems that there is generally more points recognized as dynamic in the city environments of drives 000008 and 0000010, while the highway environments seem to segment less. Even though the highway environments have more cars in them. The segmentation seem to work better in highway environments compared to city environments, which is further supported by figures 4.4 and 4.5.

Table 4.5: Point statistics for dynamic object segmentation.

Drive ID	Points	Dynamic Points	Percentage Dynamic Points
000000	$6.86 \cdot 10^8$	$5.62 \cdot 10^7$	8.20 %
000004	$2.84 \cdot 10^8$	$1.73 \cdot 10^7$	6.08 %
000008	$5.69 \cdot 10^8$	$1.21 \cdot 10^8$	21.34 %
000010	$6.44 \cdot 10^8$	$1.13 \cdot 10^8$	17.56 %

Figure 4.4a shows a highway without dynamic points, while figure 4.4b shows the same area, but only points recognized as dynamic. The ground was removed for clarity. In this relatively simple environment the dynamic segmentation works well and successfully segments cars on the capturing car's side of the road. It fails to segment the cars on the other side of the highway. But it does not seem to over segment and instead mostly only includes clearly dynamic points.



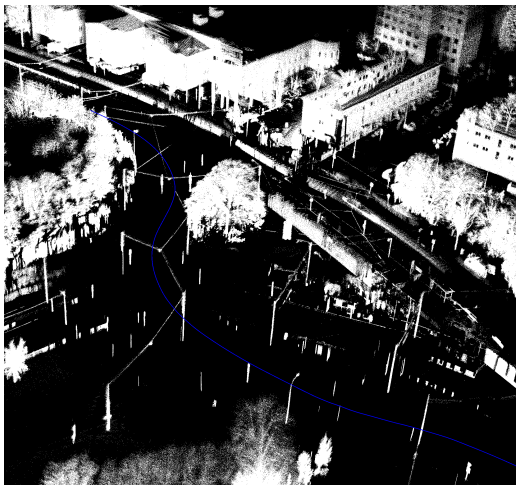
(a) A render of the points labeled as static.

(b) A render of the points labeled as dynamic.

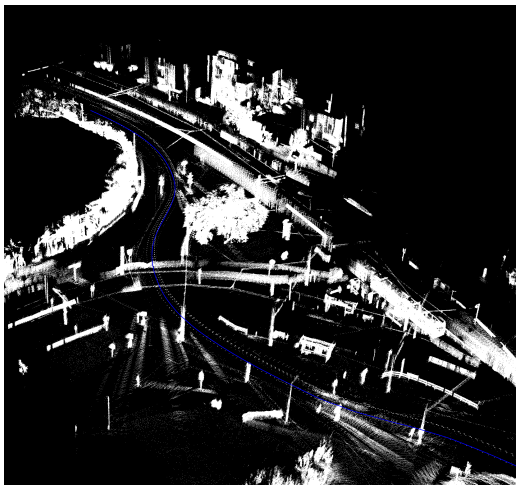
Figure 4.4: Drive 000004 included in the Zenseact Open Dataset [1].

Similarly to the ground segmentation, the dynamic object segmentation is also not accurate. It does eliminate many dynamic objects, but it also removes objects that are clearly static. This is showcased in the figures 4.5, the left image shows the points labeled as static, and without the ground points for clarity. The right figure only contains points labeled as dynamic.

Figure 4.5b overestimates the dynamic segmentation and includes objects that are clearly not dynamic, such as the wall to the left in the image, and the tree in the center. But it does manage to remove some dynamic objects such as: the smeared out persons walking in the center, the cars driving "upwards" in the image, parts of the tram that is leaving the tram stop in the center. It also seem to remove the top of some signposts.



(a) A render of the points labeled as static.



(b) A render of the points labeled as dynamic.

Figure 4.5: Captured from Drive 000010.

4.3 Downsampling

The different methods produced similar statistical results but with different runtime, and visual quality.

The octree and averge point grid methods were tested at 4 different cell sizes, 0.05 m, 0.10 m, 0.30 m and 0.50 m. Random sampling was done with 50 points with larger cell sizes, 0.50 m, 1.0 m, 3.0 m, 5.0 m.

With two iterations of downsampling, we observed that the octree version showed less visual artifacts compared to the average point grid version.

4.3.1 Runtimes

The first 300 frames of drive 000010 was used to evaluate the runtime of the different downsampling algorithms. The result is shown in table 4.6. The runtimes of octree and average point grid can not be compared to random sampling as the measure of resolution does not mean the same for both types. They are thus not compared together.

Table 4.6: Time to downsample for different algorithms. The runtimes are given in seconds per million points processed. The random sampling method sampled 50 points per cell.

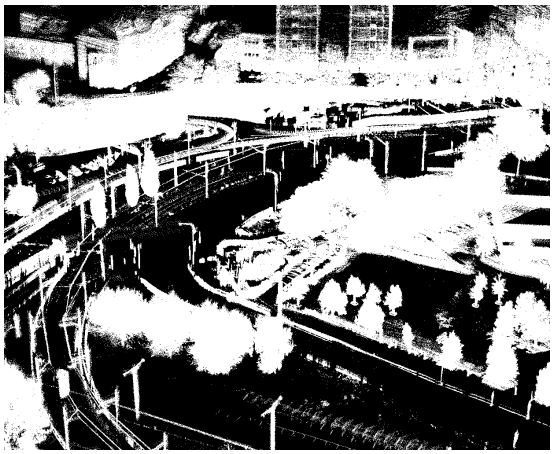
Resolution [m]	Octree [s]	Average grid [s]	Resolution [m]	rand. sample [s]
0.05	0.16	0.16	0.50	0.117
0.10	0.14	0.20	1.00	0.118
0.30	0.12	0.16	3.00	0.122
0.50	0.12	0.15	5.00	0.128

The runtime of the octree and average grid approach seem to be roughly similar. The runtime does not seem to vary much across different resolutions. All runtimes are not representative of the times it takes to perform a full drive, when storing a full drive time spent on storing and loading data in between stages affects the results.

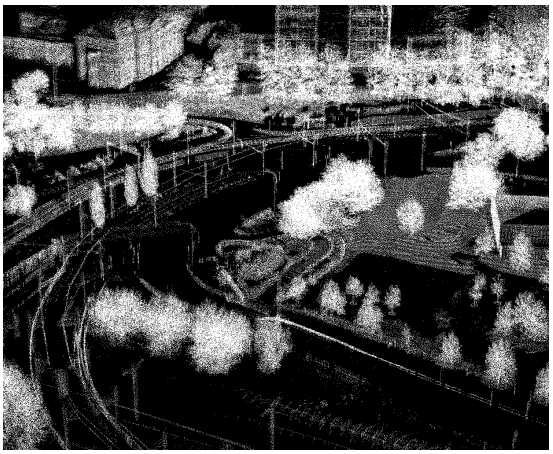
4.3.2 Visual quality

Drive 000010 has been selected as example to evaluate based on visual quality, as it has a varied city environment including trams, cars, pedestrians, houses and trees.

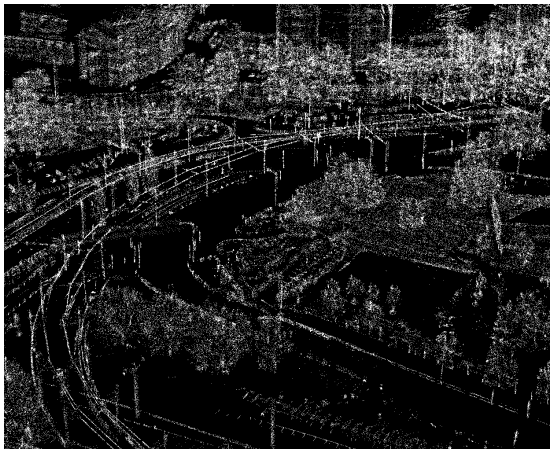
Images shown in this paper are not perfect representatives of the final renderer, as they have a level of image compression, which causes a smoothing effect when stored as a digital document.



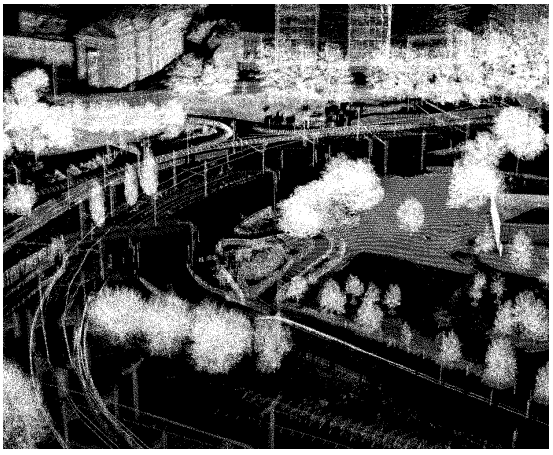
(a) Reference image with all points.



(b) Octree averaging example at 0.3 m resolution, ≈ 1.54% of points remaining



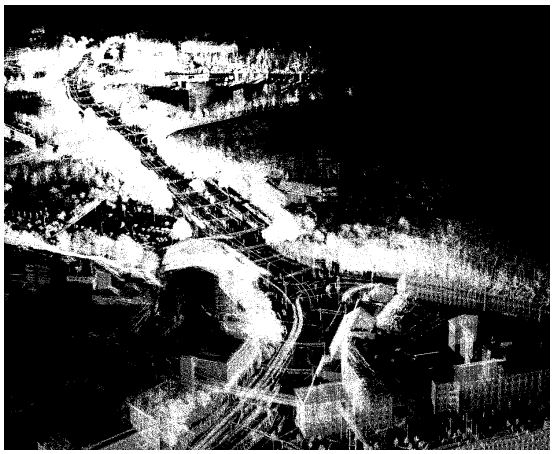
(c) Random sample example at 3.0 m resolution, 50 point count per cell, around ≈ 0.69% of points remaining



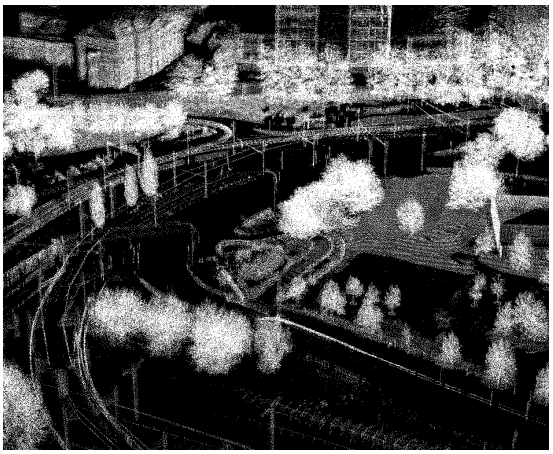
(d) Average point grid example at 0.3 m resolution, ≈ 1.99% of points remaining

Figure 4.6: Comparison between different subsampling algorithms (ground and dynamic points are removed). Captured with all frames, drive 000010.

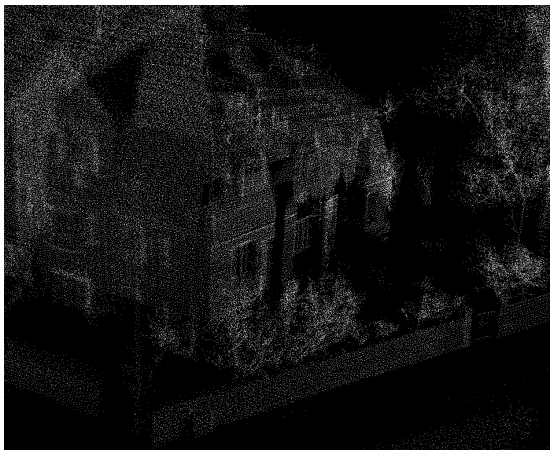
The images in figure 4.6 presents the different algorithms at roughly the same amount of points removed. The original point cloud (fig. 4.6a) has too many points and overwhelms the scene. The subsampling methods achieve a more clear image by removing a lot of points. The octree averaging (fig. 4.6b) has a clear and soft image. The random sampling method (fig. 4.6c) less points than other methods, has very different visual artifacts. Areas that had fewer and more concentrated points form bright spots in the image, such as the telephone lines or the edges of objects, while areas such as trees or the ground are more equally reduced and harder to see. The average point grid method is very similar to the octree method, but has stronger visual artifacts along certain surfaces, such as the ground in (fig. 4.6d), appearing as clear lines in the render.



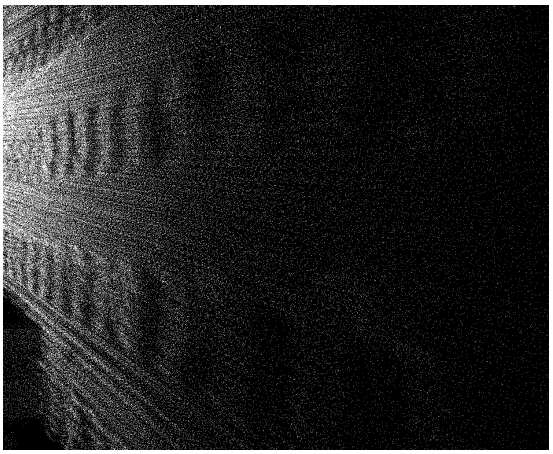
(a) 0.5 m resolution, far away



(b) 0.3 m resolution, medium distance



(c) 0.1 m resolution, close up



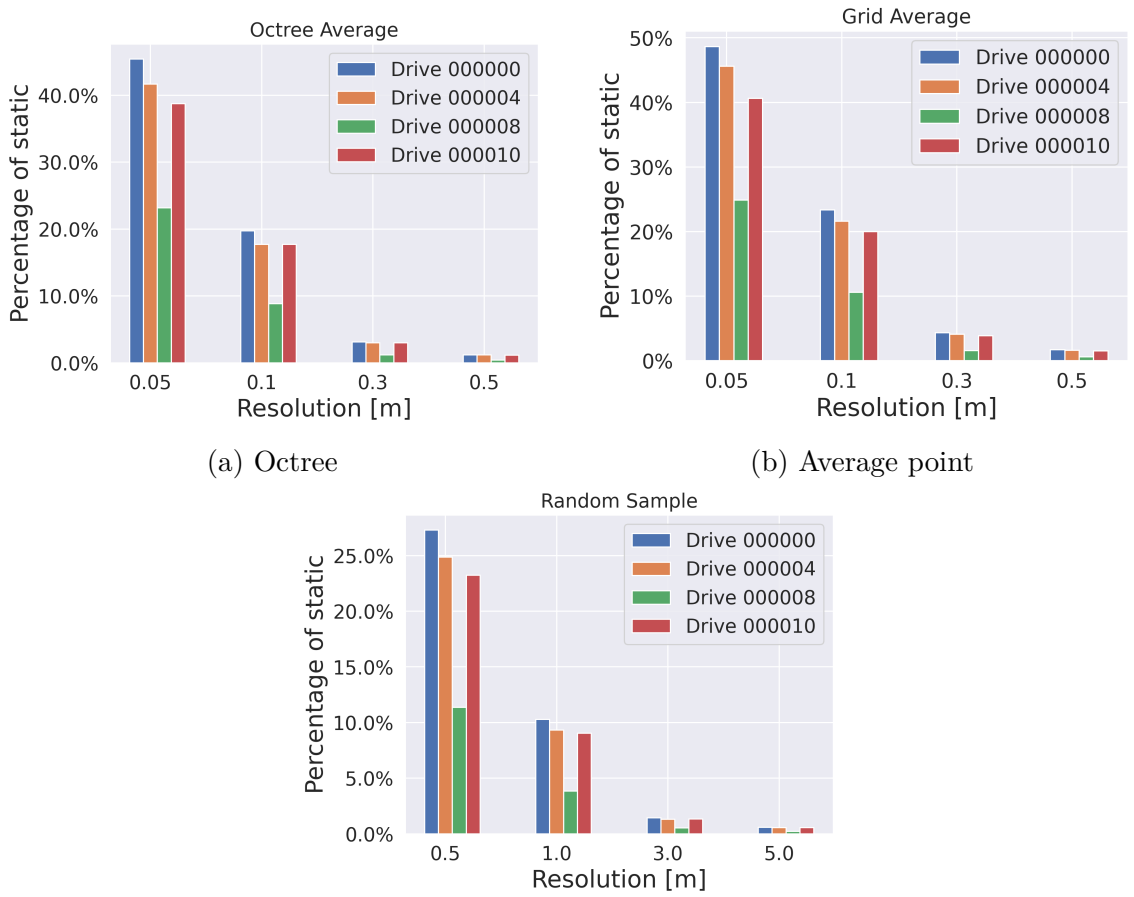
(d) 0.05 m resolution, extremely close

Figure 4.7: Octree downsampling at different resolutions with different views, Drive 000010, ground and dynamic objects have been removed.

Figure 4.7 instead contains four images of octree compression at different locations, at different distances to the viewed object at different resolutions. Different resolutions are more or less appropriate at different distances. At a closer distance, a more detailed resolution is needed to keep visual quality, while at a further distance a less detailed resolution is needed to not be overwhelmed by the point count.

4.3.3 Point Count

Figure 4.8 shows what fraction of static points, i.e. points not previously labeled as dynamic or ground, remain after the various downsampling methods, octree-, average point grid and random sample grid, have been applied. The most detailed resolutions remove at least ≈ 50% of the static points for all methods. In all cases drive 000008 have substantially less points remaining than drive 000000, drive 000004 and drive 000010. This is consistent with the average cell density shown in table 4.4.



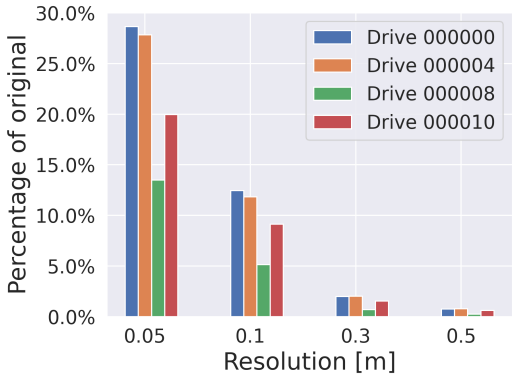
(a) Octree

(b) Average point

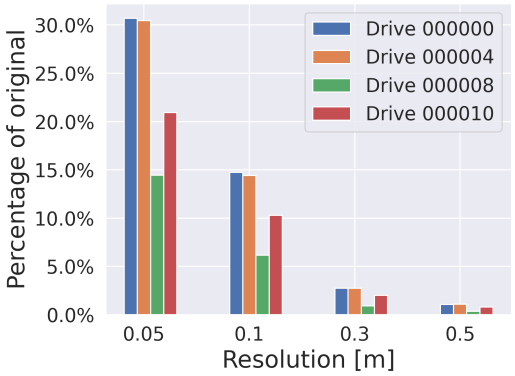
(c) Average sampling

Figure 4.8: The percentage of static points remaining for different drives and different downsampling methods.

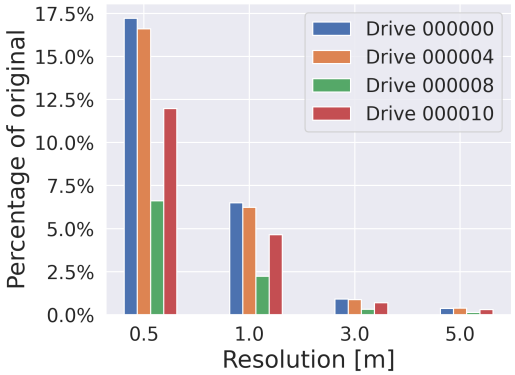
Figure 4.9 shows the final point count percentage, with ground and dynamic objects removed, compared to the original point count. The most detailed resolutions result in a final point count reduction of ≈ 30%. The various resolutions for the octree-, average point grid and random sample downsampling methods, follow a pattern similar to measuring the number of static points remaining. However, drive 000010 generally shows a higher final point reduction compared to drive 000000 and drive 000004, despite having similar static point count reductions, which aligns with the previous results regarding dynamic and ground points shown in table 4.3 and table 4.5.



(a) Octree



(b) Average point



(c) Average sampling

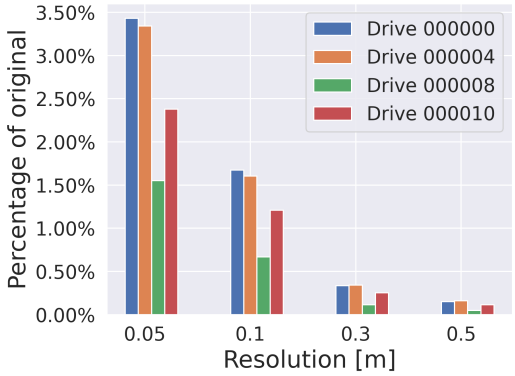
Figure 4.9: The percentage of points compared to the original point count for different drives and different downsampling methods.

4.3.4 Storing downsampled drives on disk

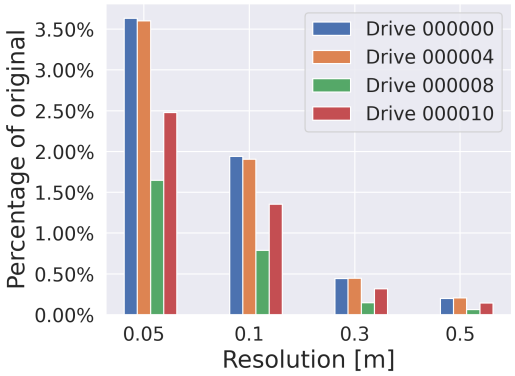
In the original dataset, each drive is is composed of a large set of frames (around 1000-2000 frames). The frames themselves are stored in as Python Numpy (`.npy`) files, and average 5 MB per frame. A resulting drive takes up 6-15 GB of memory (table 4.7). This data also contains data not needed for our visualization implementation, such as timestamps, emitter information, and intensity. That not stored in the final compressed files, which only contains point coordinates. The total final file size given in percentage of the original file size is shown in figures 4.10 for each downsampling method and drive.

Table 4.7: Original size on disk for the LiDAR datasets for different drives.

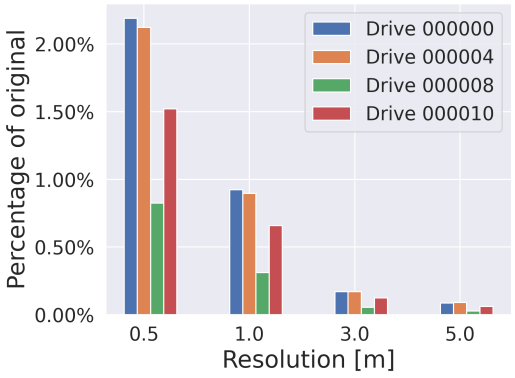
Drive ID	File size [GB]
000000	15.09
000004	6.25
000008	12.51
000010	14.16



(a) Octree



(b) Average point



(c) Average sampling

Figure 4.10: The final file size given in percentage of the original file size for the different drives and downsampling algorithms.

4.4 Renderer

There are some limitations when rendering point clouds. A typical point cloud does not include any surfaces, only points with no relation to each other. For example, a surface that is captured by a LiDAR camera only results in points that were calculated from light beams reflecting from that surface. Consequently, in our `OpenGL` renderer all points are rendered as points and not as surfaces. This creates problems as rendered points typically does not become smaller at distances, and they are not able to be occluded by other points. Two surfaces in a row would just appear as one surface, that is more dense in points, if viewed straight on.

5

Conclusion

5.1 Discussion

Overall, the methods perform fairly well. The downsampling methods massively reduce the size of the dataset which enabled the visualization of entire multi-minute LiDAR sequences at a visual quality that makes it easier to discern the environment, and the stored files are significantly smaller than in the original dataset. The segmentation methods does work to a fairly good degree, and does manage to improve the visual quality to a satisfactory degree given the limit in scope concerning object segmentation.

5.1.1 Segmentation

The segmentation process seem to overestimate, both for ground segmentation and dynamic object segmentation. In the case of the ground segmentation it successfully extracts the ground, and parts of non-ground, but it does not seem to over-segment to such an extent that artifacts appear in the non-ground segmentation.

On the other hand, the dynamic object segmentation does over-segment. For example, signposts are often segmented as dynamic, causing these to be removed from the scene. Which is bad, as signposts are an important quality for visualization software used in the automotive sector. Most likely the signposts are segmented as ground, because from some directions the signposts are quite thin, and may not appear in the LiDAR image, while from other directions where it is captured face on it appears clearly in LiDAR. This would cause it to be a classified as a dynamic object as it "disappears" in some frame.

The dynamic object segmentation also fails in cases were it cannot "see" the ground of an area that was previously occupied by an dynamic object, e.g a car on the other side of the highway with a divider in between (figure 4.4). It fails because in some LiDAR frames an object occupies an area, in others there are simply nothing there as something blocks the LiDAR rays from finding the ground the object stood on. This creates no discrepancy in occupation, therefore the object can not be marked dynamic. This could possibly be solved by interpolating a ground plane from known ground points to create points that can be used to identify discrepancies in object occupation across frames.

But, these are fairly simple methods, implemented because they are fairly fast and

easy to implement. For the purposes of exploring compression techniques, as in this thesis, the requirement for good segmentation is not that strong. But, for a more polished product, a better ground- and dynamic object segmentation should be used.

The runtime of the ground segmentation is very fast and it is not dependent on what environment a drive is running through, as the difference in number of points between environments does not affect the runtime substantially.

The runtime of the dynamic object segmentation on the other hand seem to vary depending on the nature of the given drive. From table 4.4 a clear correlation between average speed and the time to segment a pose is shown. A slower speed causes more LiDAR images per length unit moved, which causes a higher density in points. Compared to higher speeds were the opposite occurs, less images per length unit, therefore less density of points. Slow speed are often associated with urban environments which also typically has more objects for the LiDAR camera to image, such as many buildings, taller buildings and people, which results in more points per LiDAR frame.

The problem that occurs in the dynamic object segmentation process is that the memory requirements are much higher for a slower moving drive as there are more points to manage.

A heuristic based on average drive speed that creates smaller windows, or uses less parallelism may be useful to decrease memory usages in the cases where the point density is higher. This would reduce the need for swapping memory to disk, and reduce the large slowdowns caused by swapped memory. Another method would be to simply increase the memory of the compute device.

The dynamic object process is also sensitive to which parameters are chosen, such as the physical real world size of the bins that points are deposited into. For example, a bin of too large physical size will cause the objects identified as dynamic to remove other, non-dynamic, parts with it, as the algorithm removes entire bins.

For future work, other methods should be explored. A machine learning approach [3] could work well. Or a method which combines various non machine learning methods [19] seem to give good results. There could possibly also be an effective sensor fusion method which employs visual cameras in conjunction with LiDAR to identify which LiDAR points are dynamic [20].

5.1.2 Downsampling

Between the different downsampling methods, it is hard to make direct comparisons other than visual quality. The visual quality was compared to the best abilities of the authors of this paper. A more thorough survey, taking into account the opinions of a larger population, should be conducted to achieve the best possible visual consensus.

The random sample are given different cell sizes compared to average point and octree. This makes them difficult to compare, because a number of randomly sampled

points can not meaningfully be compared to 1 point given the same cell size.

A side effect of the visualization method being purely point-based is that certain visual attributes appear visually better in certain context but fail in other. One such case is plain surfaces, which generally behave quite differently to more natural shapes such as trees, ground. Surfaces appear most visually consistent when the points are slightly standardized, such that the lines of the plane are readable, but not too dense, and such that borders are distinguishable from the plane. This is in opposition to shapes like trees, which contain far more edges to keep track of, and a standardized approach makes the image harder to interpret.

Another problem is close up images, which fail to show up properly when looking at large resolutions, while far away images would appear cluttered if the resolution is too low, see figure 4.7 as an example. If the visualizer is to be kept as point based, then a level-of-detail approach with different downsample resolutions would need to be necessary to ensure readability regardless of distance.

For future work the surfaces could also be identified and replaced by a geometry such as a mesh or a voxel. This would enable a surface of points to be represented by a much more efficient data structure, and would solve some of the issues with visualizing points.

Another thing to note is the quality of the poses given in the dataset. If the poses for a given drive, or even just part of a drive, would be less accurate, this downsampling algorithm would not be delete points correctly, as the points would not be overlapping as much. Future work could be towards extending the pipeline to include the SLAM-segment into the pipeline, as the work necessary to determine accurate poses should overlap with the ability to detect redundant points.

5.1.3 Storing processed clouds on disk

Although storing the clouds via octree compression gives smaller files than storing them directly, there is an added time spent to later decompress the clouds for rendering. Keeping the data in a compressed cloud may not be the most optimal way to keep the data if the goal is quick render time. But may still be preferable for transporting point clouds over networks, or for saving storage.

5.2 Future applications

There could be an interest to evaluate if it is possible to create a pseudo-frame from a world-map. If a downsampled world-map could be stored at an equivalent point density to a single LiDAR frame, then massive amounts of storage could be saved. Instead of storing every frame containing overlapping points, a pseudo-frame could instead be created as necessary at a specific point in the LiDAR world-map, by means such as closest neighbor search or similar. It would be necessary to evaluate if a pseudo-frame would perform worse or comparable to a normal LiDAR frame in the relevant applications.

5.3 Conclusion

Different methods were used and developed to create a proof-of-concept software that enables the visualization of a dataset containing multiple minutes of LiDAR frames. Downsampling techniques were used to reduce the necessary data overhead needed to visualize a large LiDAR point cloud dataset. Simple ground and dynamic object segmentation methods were implemented to improve the visual quality of the visualizer. From downsampling and removing unnecessary data for visualization and compression, the data size could be reduced to 0.06%-3.6% of the original file size depending on chosen resolution.

Bibliography

[1] M. Alibeigi et al., "Zenseact open dataset: A large-scale and diverse multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[2] J. Carter et al., *Lidar 101: An Introduction to Lidar Technology Data, and Applications*. Charleston, South Carolina: National Oceanic and Atmospheric Administration (NOAA) Coastal Services Center, 2011.

[3] X. Chen et al., "Moving Object Segmentation in 3D LiDAR Data: A Learning-based Approach Exploiting Sequential Data," *IEEE Robotics and Automation Letters (RA-L)*, vol. 6, pp. 6529–6536, 4 2021, ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3093567. [Online]. Available: http://www.ipb.uni-bonn.de/pdfs/chen2021ral-iros.pdf.

[4] H. Lim, S. Hwang, and H. Myung, "Erasor: Egocentric ratio of pseudo occupancy-based dynamic object removal for static 3d point cloud map building," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2272–2279, 2021. DOI: 10.1109/LRA.2021.3061363.

[5] Z. Shen, H. Liang, L. Lin, Z. Wang, W. Huang, and J. Yu, "Fast ground segmentation for 3d lidar point cloud based on jump-convolution-process," *Remote Sensing*, vol. 13, no. 16, 2021, ISSN: 2072-4292. DOI: 10.3390/rs13163239. [Online]. Available: https://www.mdpi.com/2072-4292/13/16/3239.

[6] B. Mersch, T. Guadagnino, X. Chen, I. Vizzo, J. Behley, and C. Stachniss, "Building Volumetric Beliefs for Dynamic Environments Exploiting Map-Based Moving Object Segmentation," *IEEE Robotics and Automation Letters (RA-L)*, vol. 8, no. 8, pp. 5180–5187, 2023, ISSN: 2377-3766. DOI: 10.1109/LRA.2023.3292583.

[7] B. Mersch, X. Chen, I. Vizzo, L. Nunes, J. Behley, and C. Stachniss, "Receding Moving Object Segmentation in 3D LiDAR Data Using Sparse 4D Convolutions," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 3, pp. 7503–7510, 2022.

[8] G. Kim and A. Kim, "Remove, then revert: Static point cloud map construction using multiresolution range images," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 10 758–10 765. DOI: 10.1109/IROS45743.2020.9340856.

[9] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013. DOI: 10.1177/0278364913491297. eprint: https://doi.

org/10.1177/0278364913491297. [Online]. Available: https://doi.org/10.1177/0278364913491297.

[10] Y. Cao, Y. Wang, and H. Chen, *Real-time lidar point cloud compression and transmission for resource-constrained robots*, 2025. arXiv: 2502.06123 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2502.06123.

[11] S. Schwarz et al., "Emerging mpeg standards for point cloud compression," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 133–148, 2019. DOI: 10.1109/JETCAS.2018.2885981.

[12] R. Roriz, H. Silva, F. Dias, and T. Gomes, "A survey on data compression techniques for automotive lidar point clouds," *Sensors*, vol. 24, no. 10, 2024, ISSN: 1424-8220. DOI: 10.3390/s24103185. [Online]. Available: https://www.mdpi.com/1424-8220/24/10/3185.

[13] G. Riegler, A. O. Ulusoy, and A. Geiger, "Octnet: Learning deep 3d representations at high resolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6620–6629. DOI: 10.1109/CVPR.2017.701.

[14] W. Lyu, W. Ke, H. Sheng, X. Ma, and H. Zhang, "Dynamic downsampling algorithm for 3d point cloud map based on voxel filtering," *Applied Sciences*, vol. 14, no. 8, 2024, ISSN: 2076-3417. DOI: 10.3390/app14083160. [Online]. Available: https://www.mdpi.com/2076-3417/14/8/3160.

[15] D. Meagher, *Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer*. Oct. 1980.

[16] G. Wallace, "The jpeg still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992. DOI: 10.1109/30.125072.

[17] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[18] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Application VISSAPP'09)*, INSTICC Press, 2009, pp. 331–340.

[19] A. Gupta, S. Jain, P. Choudhary, and M. Parida, "Dynamic object detection using sparse lidar data for autonomous machine driving and road safety applications," *Expert Systems with Applications*, vol. 255, p. 124636, 2024, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2024.124636. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417424015033.

[20] W. Chen et al., "Slam overview: From single sensor to heterogeneous fusion," *Remote Sensing*, vol. 14, no. 23, 2022, ISSN: 2072-4292. DOI: 10.3390/rs14236033. [Online]. Available: https://www.mdpi.com/2072-4292/14/23/6033.

A

Appendix 1

A.1 Drive dataset size

The data size of different drives.

Table A.1: Dataset sizes of various drives

Drive ID	LiDAR dataset size [GB]
000000	15.09
000003	12.05
000004	6.25
000006	14.33
000008	12.51
000009	13.84
000010	14.16
000011	15.22

A.2 Downsampled dataset size

Dataset sizes for downsampled drives at various resolutions.

Table A.2: Downsampled dataset size for octree downsampling

Resolution [m]	Drive 000000	Drive 000004	Drive 000008	Drive 000010
original	15.09 GB	6.25 GB	12.51 GB	14.16 GB
0.05	517.7 MB (3.43%)	208.8 MB (1.38%)	194.2 MB (1.29%)	337.0 MB (2.23%)
0.10	252.7 MB (4.04%)	100.2 MB (1.60%)	83.8 MB (1.34%)	171.4 MB (2.74%)
0.30	50.5 MB (0.40%)	21.4 MB (0.17%)	14.3 MB (0.11%)	35.9 MB (0.29%)
0.50	22.8 MB (0.16%)	10.0 MB (0.07%)	6.1 MB (0.04%)	16.0 MB (0.11%)

Table A.3: Downsampled dataset size for average point grid downsampling

Resolution [m]	Drive 000000	Drive 000004	Drive 000008	Drive 000010
original	15.09 GB	6.25 GB	12.51 GB	14.16 GB
0.05	548.2 MB (3.63%)	225.0 MB (1.49%)	205.9 MB (1.36%)	350.6 MB (2.32%)
0.10	293.0 MB (4.69%)	119.1 MB (1.91%)	98.5 MB (1.58%)	191.6 MB (3.07%)
0.30	66.6 MB (0.53%)	28.0 MB (0.22%)	18.2 MB (0.15%)	44.9 MB (0.36%)
0.50	30.1 MB (0.21%)	12.9 MB (0.09%)	7.6 MB (0.05%)	20.0 MB (0.14%)

Table A.4: Downsampled dataset size for random sampling

Resolution [m]	Drive 000000	Drive 000004	Drive 000008	Drive 000010
original	15.09 GB	6.25 GB	12.51 GB	14.16 GB
0.5	330.5 MB (2.19%)	132.7 MB (0.88%)	103.1 MB (0.68%)	215.5 MB (1.43%)
1.0	139.3 MB (2.23%)	56.0 MB (0.90%)	39.0 MB (0.62%)	93.1 MB (1.49%)
3.0	25.6 MB (0.20%)	10.6 MB (0.08%)	6.9 MB (0.05%)	17.4 MB (0.14%)
5.0	12.9 MB (0.09%)	5.7 MB (0.04%)	3.3 MB (0.02%)	8.5 MB (0.06%)

A.3 Downsample Point Counts

The number of points remaining after downsampling.

Table A.5: Octree

Drive ID	Resolution	Point Count
000000	0.05	196525752
000000	0.10	85388682
000000	0.30	13624504
000000	0.50	5171036
000004	0.05	79104215
000004	0.10	33651019
000004	0.30	5720080
000004	0.50	2251570
000008	0.05	76657489
000008	0.10	29268606
000008	0.30	3995759
000008	0.50	1444353
000010	0.05	128548563
000010	0.10	58756123
000010	0.30	9939692
000010	0.50	3864099

Table A.6: Average point

Drive ID	Resolution	Point Count
000000	0.05	210461631
000000	0.10	101078432
000000	0.30	18701946
000000	0.50	7293193
000004	0.05	86517626
000004	0.10	40962657
000004	0.30	7796043
000004	0.50	3079118
000008	0.05	82160988
000008	0.10	35059394
000008	0.30	5212567
000008	0.50	1907330
000010	0.05	134747754
000010	0.10	66278801
000010	0.30	12786350
000010	0.50	5031233

Table A.7: Random sampling 50 points

Drive ID	Resolution	Point Count
000000	0.5	118082414
000000	1.0	44509442
000000	3.0	6233111
000000	5.0	2505251
000004	0.5	47209542
000004	1.0	17717878
000004	3.0	2502092
000004	5.0	1081293
000008	0.5	37580784
000008	1.0	12712792
000008	3.0	1761055
000008	5.0	720755
000010	0.5	77040824
000010	1.0	29952696
000010	3.0	4435326
000010	5.0	1838012