HAW
HAMBURG

MASTER THESIS
Sebastian Brückner

Surface reconstruction for mapping applications from LiDAR point clouds

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Sebastian Brückner

Surface reconstruction for mapping applications from LiDAR point clouds

Master thesis submitted for examination in Master´s degree
in the study course *Master of Science Informatik*
at the Department Computer Science
at the Faculty of Engineering and Computer Science
at University of Applied Science Hamburg

Supervisor: Prof. Dr. Tim Tiedemann
Supervisor: Prof. Dr. Peer Stelldinger

Submitted on: 17. August 2023

Sebastian Brückner

Title of Thesis

Surface reconstruction for mapping applications from LiDAR point clouds

Keywords

LiDAR, Point cloud, surface reconstruction, High Definition Maps, 3D Scanning, clustering, HDBSCAN

Abstract
The demand for high-definition (HD) maps has risen significantly, extending into diverse applications. This work presents an algorithmic approach to geometry generation for HD Maps by leveraging mobile LiDAR point cloud data. The algorithm integrates surface reconstruction, HDBSCAN-based clustering, spline fitting, and outline detection. Through rigorous evaluation against ground truth data, the algorithm's accuracy and efficiency are assessed on a purpose-built datasets with ground truth.

DE:

Thema der Arbeit

Oberflächenrekonstruktion für kartografie Anwendungen aus LiDAR Punktwolken

Stichworte
LiDAR, Punktwolken, Oberflächenrekonstruktion, hochauflösende karten, 3D Scanning, clustering, HDBSCAN

Kurzzusammenfassung

Die Nachfrage nach hochauflösenden (HD) Karten ist erheblich gestiegen und erstreckt sich auf verschiedene Anwendungen. Diese Arbeit stellt einen algorithmischen Ansatz zur Erzeugung von Geometrie für HD-Karten vor, welcher mobile LiDAR-Punktwolkendaten als Basis nutzt. Der Algorithmus integriert Oberflächenrekonstruktion, HDBSCAN-basiertes Clustering, Spline-Fitting und Umrisserkennung. Die Genauigkeit und Effizienz des Algorithmus werden anhand von eigens erstellten Datensatzes mit Ground Truth bewertet.

Contents

List of Figures

Abbreviations

ALS	Airborne laser scanner.
DBSCAN	DBSCAN.
FOV	Field of view.
GNSS	Global Navigation Satellite System.
GPS	Global Positioning System.
HAW	Hochschule für Angewandte Wissenschaften.
HD	High definition map.
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise.
IMU	Inertial measurement unit.
kNN	k-nearest neighbor(hood).
LiDAR	Light detection and ranging.
MLS	Mobile laser scanner.
MST	Minimum spanning tree.
PCA	Principal component analysis.
RANSAC	Random sample consensus.
TLS	Terrestrial laser scanner.

Symbols

H radii used in ball pivoting.

S smoothing factor for spline fitting.

d_s maximum distance between a cluster and a point to be considered for fitting.

L_3 normalized third eigenvalue of a PCA.

m_c minimum cluster size of HDBSCAN.

m_s minimum samples of HDBSCAN.

ng normal gain used to scale the normal influence in clustering.

$t_\triangle$ maximum distance between a spline and a point to be fitted to it.

1 Introduction

Recent LiDARs allow the capture of point clouds of urban environments with significant detail. At the same time, large areas can be scanned if the sensor is mounted to a mobile platform. LiDAR technology has improved significantly over the recent years. The Sensors have become higher resolution and are much cheaper. High Definition (HD) Maps are maps that go beyond traditional mapping methods. These properties made them become a valuable tool for a wide array of purposes, including urban planning, environmental monitoring, infrastructure management, and augmented reality experiences. The large point clouds captured with a mobile LiDAR presents itself as an excellent basis for such maps. Yet, this combination of LiDAR generated point clouds and HD Maps only starts to get traction in research when it comes to non-airborne mobile platforms. This work tries to make a contribution to generate the geometry for HD Maps, focusing on a sophisticated algorithmic approach that harnesses the potential of LiDAR point cloud data. The central objective of this research is to explore, refine, and optimize the algorithm's accuracy tailored to the special properties of mobile LiDAR data. Central to the algorithm is a comprehensive multistep process that entails surface reconstruction, intelligent clustering using HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise), precise spline fitting, and outline detection. By seamlessly integrating these processes, the algorithm aims to capture the intricate nuances of diverse landscapes, effectively addressing challenges arising from disparate point densities, noise contamination, and complex surface geometries.

First, the basis for the general problem is started. Then existing surface recovery and HD Map creation algorithms are portrayed. Following this, an algorithm is designed to the criteria of HD Maps and its implementation explained. To evaluate the algorithm, a Dataset is constructed that contains ground truth for the underlying surfaces in the point cloud. A comprehensive evaluation of the algorithm's accuracy and performance and its parts is undertaken, rigorously scrutinizing its outcomes against ground truth data and discerning its adaptability for real-world applications. Subsequent sections delve into the algorithm's outcomes, its inherent limitations, and avenues for future enhancement.

2 Basics

2.1 LiDAR

Light detection and ranging (LiDAR) is a remote sensing technology used to measure distances and create detailed point clouds. LiDARs are Time-of-Flight sensors, that work by sending out laser pulses and recording the time until the pulse is returned by a reflection. The electromagnetic spectrum used is mostly near-infrared, which enables much higher resolution than for example ultrasound sensors or Radar, while achieving higher and lower ranges respectively. Minimum and maximum ranges can vary depending on the used LiDAR and its configuration from a few centimeters to several hundred meters. The accuracy of LiDAR also depends on the LiDAR sensor used, but also on the reflectance of the scanned objects surface and other common sources of inaccuracy, for example scattering caused by bad visibility (e.g. rain, fog or dust). Most LiDARs deliver a range accuracy in the range of a few centimeters, with some even in the millimeters. The area a LiDAR can scan is called its Field-of-vied (FOV), having a horizontal and vertical component. The LiDAR beams diverge, and the scan density of LiDARs decreases with distance. Some LiDARs also record the intensity the returned signal. Typical common LiDAR resolutions are from $0.2°$ upwards, with vertical FOV ranging from a single line up 60 degree.

Two main forms of LiDAR need to be distinguished: Rotating LiDARs and solid state LiDARs. Rotating LiDAR employs a spinning mechanism to scan the environment. The LiDAR sensor typically consists of a laser emitter and a rotating mirror, or the whole scanning assembly rotates. Higher rotation speeds enable higher scan frequencies, but reduce angular resolution. The laser emits a pulse of light, which is then directed by the rotating element to cover the entire 360-degree horizontal FOV. As the sensor rotates, it captures a series of individual distances, which can be translated to a position in which the scanner uses its current angle of rotation. Rotating LiDARs can feature multiple scan lines to expand the vertical FOV. Solid state LiDARs use an array of lasers and detectors to send out multiple beams at once. With no moving parts, these tend to be more reliable, while having a smaller FOV and higher scan frequencies of the covered area. Often, especially for solid state LiDAR with smaller FOV, multiple LiDARs are used on one vehicle and their measurements are fused.

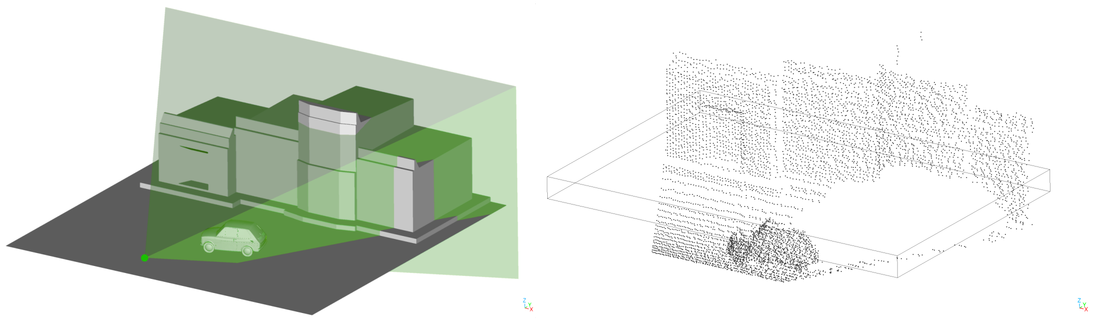
The technology used can often be seen in the resulting point cloud. Every LiDAR produces a different scan pattern. These are mostly the scan lines of the LiDAR, but also the scan frequency and rotation speed can be important.

A critical role for the shape of the generated data is also the mounting of the LiDAR. [49] categorizes LiDAR mounting into four categories:

1. **Terrestial Laser Scanner** (TLS) LiDARs mounted to a fixed, stable position, for example a tripod. Usually high accuracy LiDARs with low resolution where many frames are taken in different directions from the same position. Often used for surveying a single structure. Multiple fixed positions can be combined to one point cloud to avoid scan shadows

2. **Mobile Laser Scanner** (MLS) LiDARs mounted to mobile platforms such as cars or robots on the ground, or carried by hand. Used on (partially) autonomous systems or to generate data of a wide area. Usually less accurate data than TLS. However, they are limited in the height of recorded structures by the LiDAR FOV.

3. **Airborne Laser Scanner** (ALS) LiDARs mounted an Arial vehicle. Can Partially overlap with MLS when talking about low altitude platforms like small drones. Used to survey very large areas, for example, on a plane. By its very nature of being recorded from above, its mostly missing vertical structures.



(a) (b) The produces point cloud with scan shadows

Figure 2.1: Example Scene: A solid state LiDAR (green ball) with FOV 60° with resolution 120 times 60 (green frustum) points at a car in front of a building.

2.2 Point Clouds

LiDARs produce measurements of points on the surfaces of objects. These measurements are recorded as 3D points in Cartesian coordinates. A single LiDAR measurement is called a LiDAR frame. For a rotating LiDAR, this is often one full rotation. Since one LiDAR frame does not contain much information due to the limited resolution, multiple LIDAR frames are often fused to one point cloud in a process called registration or scan matching. Many algorithms exist to solve this problem, for example Iterative Closest Point [23]. These algorithms also apply the help of one or all of the GNSS, IMU or the odometry of the system carrying the LiDAR. Point clouds can be relatively small, for example to show the immediate surroundings of an indoor scene for a robot, or they can be extremely large, for example, the 3D scan of an entire city.

2.3 High Definition Maps

High Definition (HD) maps offer many advantages over more classic digital maps used in car navigation system since the advent of GPS in the early 2000s and found in navigation apps on almost all smartphones. The most significant difference is their usage of third dimension. While most digital maps are simply flat, HD Maps contain more information than a flattened top-down view. Common elements are man-made structures such as signage and buildings, vegetation and the general terrain features.

HD Maps offer higher accuracy then previous maps. Often, accuracies in the centimeter range are desired. Such a high accuracy enables usages of such maps not previously possible, for example, navigation of autonomous systems by map matching, a technique that determines the position of a system by comparing the features detected using its sensor systems with the given map.

HD Maps are not only used for navigation and autonomous driving on the road, but also map more areas that were previously not available in maps or just very coarsely. These areas usually include footpaths, plazas and all other open space not reachable by car. Sometimes, even the insides of buildings are contained.

HD Maps are often split into different layers [30]. While different layer models exist, most of them can be categorized into two different types. Geometric layers and semantic layers. Geometric layers define the shape of the objects in the map, like buildings, landscape etc. These can range from relatively simple shapes like boxes or simple planes for man-made structures, to fully detailed models for every tree in the map. Semantic layers annotate these geometries. For example, a map suited for autonomous driving will contain information on the lanes of the roads and their relation with each other. With these relations, correct behavior planning of the autonomous vehicle can be deduced. For example, which lane changes are valid. HD Maps are also used on smaller systems, like drones and small robots or IoT devices, making the memory efficiency of these geometric definitions a concern. If the HD Map lives on the device, its size must be limited to the available space, or when streaming, it has to work with slow connections.

3 Literature Review

3.1 Surfaces

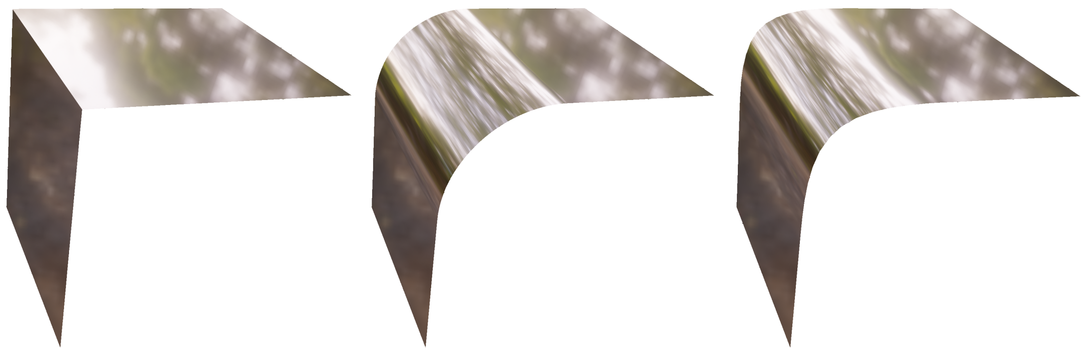
Surfaces describe the boundary between two regions of space. The model that is used to describe a surface is important for the actual structure of the object represented. The surface models describe a two-dimensional object in three-dimensional space. Both extend and surface area are often finite for most surface models and are the useful cases for most application, but this must not always be true. For example, 3D fractals have a limited extend but infinite surface area, while a simple euclidean plane is infinite in both aspects.

3.1.1 Features

These features can include a variety of geometric and topological traits that provide important information about the nature and behavior of the surface. For example, this can include things like its curvature, roughness, edges or holes.

3.1.1.1 Continuity

In mathematics, a singularity describes a point or region where a mathematical object is not well-behaved. Surface continuity is a property of a surface. It describes the smoothness in the transition between any two neighboring points on a surface. C_0 continuity simply means that the surface is connected, for example a cube. To have C_1 continuity the two tangent vectors of all neighboring points on a surface must be equal in both magnitude and direction. This implies that the first derivative of the surface has no singularities. C_2 continuity means that the curvature of the object does not change abruptly, see fig. 3.1. Generally, C_N continuity means that the N derivatives of the surfaces has no singularities, e.g. the tangent vector of all neighboring points are equal both



(a) C_0 surface with edge (b) C_1 planes connected by circular arc (c) C_2 cubic B-Spline surface

Figure 3.1: Different kinds of surface continuity on mostly specular reflective materials. (a) has a singularity at the edge. Note how both (b) and (c) appear smooth, but the reflection on (b) changes abruptly at the end of the circular arc while the reflection on (c) stays smooth. The abrupt change marks the singularity in the second derivative of (b)

in magnitude and direction. Geometric continuity relaxes this relation by only taking the vector direction into account. Continuity is consecutive. So a C_3 surface must also be C_0, C_1 and C_2.

3.1.1.2 Principal Component Analysis

Principal Component Analysis (PCA) is a method to reduce the dimensions of a data set. To achieve this, the principal components P of the data are computed. This is done by Eigenvalue decomposition of the covariance matrix of the data. The first principal component is the axis on which the data has the highest variance and the other principal components are always perpendicular to the first one. The eigenvalues L of the eigenvectors indicate how dominant a principal component is. The eigenvalues determine the accuracy of the transformation to a lower space. In practice, the eigenvalues and vectors are often calculated by singular value decomposition.

A PCA on a Dataset in $\mathbb{R}$ will result in three eigenvalues $L1$,$L2$ and $L3$. If $L1$ is dominant, the feature has a roughly one dimensional extend. If $L1 \approxeq L2$ are similar, the feature is flat. If $L3 \approxeq L1$, the data is spread out over all principal components. [45]

This can be applied to a surface, by calculating the PCA for a point on that surfaces certain radius r, to determine the flatness of the surface within that region. Also, a vector perpendicular to the first two components can be a good estimate for the surface normal at that point. The eigenvalues can be normalized to make them comparable between points. A limitation of PCA is, that it is very susceptible to noise. For PCA to work, the data must be centered and normalized. There are multiple methods available that make PCA perform better with noisy data. These so called RPCA methods are mostly computationally much more expensive [47].

3.1.2 Models

3.1.2.1 Meshes

Meshes represent surfaces of objects by repeating flat surface primitives that are connected. Most commonly used are triangles and quadrilaterals. A mesh consists of the following components:

- **Vertices:** Individual points in 3D space that define the positions of the corners of the piecewise surfaces.
- **Edges:** Straight-line segments connecting two vertices. They form the boundaries of the faces and define the shape of the mesh.
- **Faces:** Flat surfaces that connect a set of vertices and edges.
- **Normals:** Normal vector standing on the surface or vertex of the mesh

The edges and faces are normally defined as a list of vertex indices that form this face. Face normals can be made implicit by defining them by the order of the vertex indices.

A mesh is called manifold when he adheres to the following conditions:

1. The absence of T-Junctions. A T-Junction forms when more than two edges connect.
2. No open Boundaries. If an edge connects to no other edge, the edge is open. These boundary edges are sometimes allowed. These meshes are then sometimes called manifold with boundaries.
3. No self intersections

Meshes offer great flexibility, but since they are always constructed of flat surfaces, they can only approximate the underlying object geometry [26].

Another feature of a mesh is its quality. The quality of a mesh can be determined by many metrics [27]. Generally speaking, most of them favor meshes that have:

1. Roughly equally sized triangles
2. The amount of triangles fits the objects curvature (for example, not to many triangles on a flat surface)
3. Triangles that do not feature very sharp or flat angles between edges

Meshes that follow these conditions are generally easier to work with.

3.1.2.2 B-Splines Surfaces

Splines are curves that are defined by piecewise joint polynomials. The degree of the polynomials in the segments that are joint together is called the degree d of the spline. For example, a polyline could be expressed with a spline of $d = 1$, since all connections are linear. The points where the segments are joined are called knots. All splines have in common that they are defined by a set of n control points $P = P_0, P_1, \dots P_n$. The curve is a combination of these control points. The way these control points influence, or respectively, are combined to the curve determines the type of the spline. Depending on the type of the spline, the curve can pass through these points, but it is not mandatory. Many kinds of splines exist. The splines type is determined by how the control points influence the shape of the resulting curve. The different kinds of splines have different continuity properties and therefore, different use cases.

A common spline is the cubic B-Spline. As the name suggest, it uses polynomials of $d = 3$. For this kind of spline, the curves generally do not pass through the control points P. The biggest advantage of cubic B-Spline is, that they are C^2 smooth [26]. With higher degrees, for example a quadratic B-Spline, the continuity increases, following the rule C^{d-1}. A technique to make the spline behave nicely at the end, is to add two virtual control points V_0 and V_1 at the beginning and end of the spline, see fig. 3.2. V_0 is a mirror of P_1 mirrored around P_0. The same principal is applied to V_1 with P_{n-1} and P_n. The virtual points are then added to P forming V with size m A cubic B-Spline B can be defined by [26]:

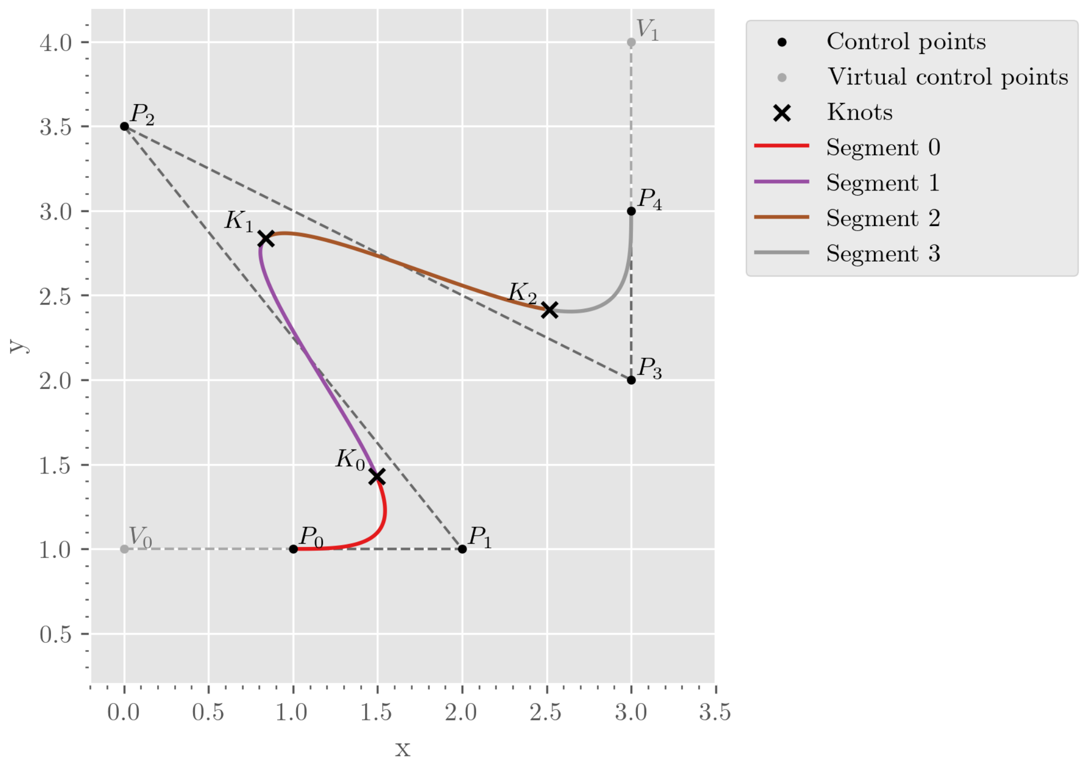


Figure 3.2: A spline with five control points forming 4 segments with 3 knots

$$B(t) = \sum_{i=0}^{m} V_i * b_3((t-3)-i) \tag{3.1}$$

$$b_3(s) = \begin{cases} 0 & \text{if } t < 0 \\ \frac{1}{6}t^3 & \text{if } 0 \leq t \leq 1 \\ \frac{1}{6}(-3(t-1)^3 + 3(t-1)^2 + 3(t-1) + 1) & \text{if } 1 < t \leq 2 \\ \frac{1}{6}(3(t-2)^3 - 6(t-2)^2 + 4) & \text{if } 2 < t \leq 3 \\ \frac{1}{6}(-(t-3)^3 + 3(t-3)^2 - 3(t-3) + 1) & \text{if } 3 < t \leq 4 \end{cases} \tag{3.2}$$

B is defined in the range $0 \leq t \leq n-1$. The function b_3 can be derived by the constraints that a spline must fulfill for C^2 continuity. b_3 basically limits how much a control point can pull on the line. This is called the Basis function. There are other, more computationally efficient ways to define a spline, but this is the easiest to comprehend.

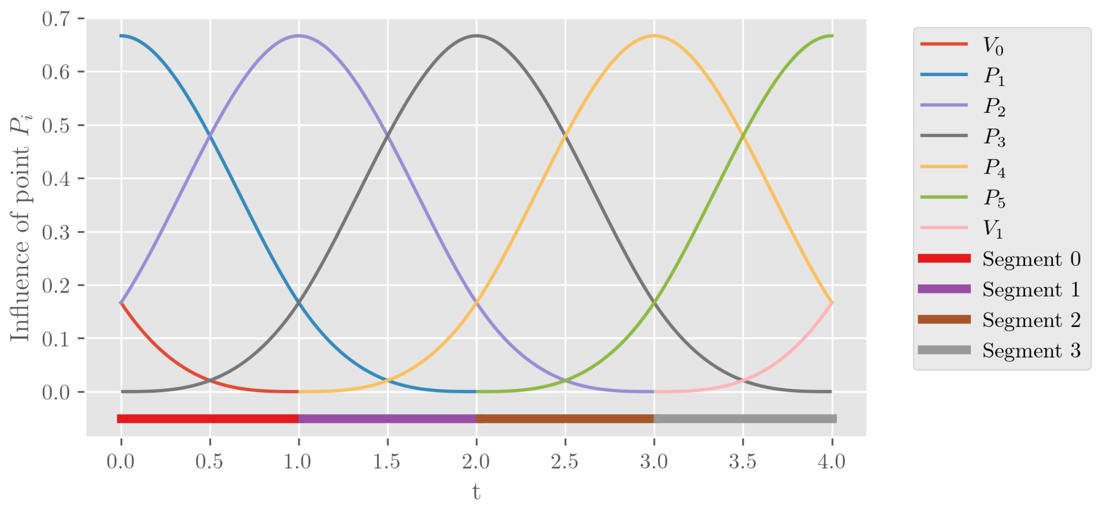


Figure 3.3: The influences of the five control points in fig. 3.2 with raising t values. The shape of the individual influence curves for each control point are equal to the shape of b_3

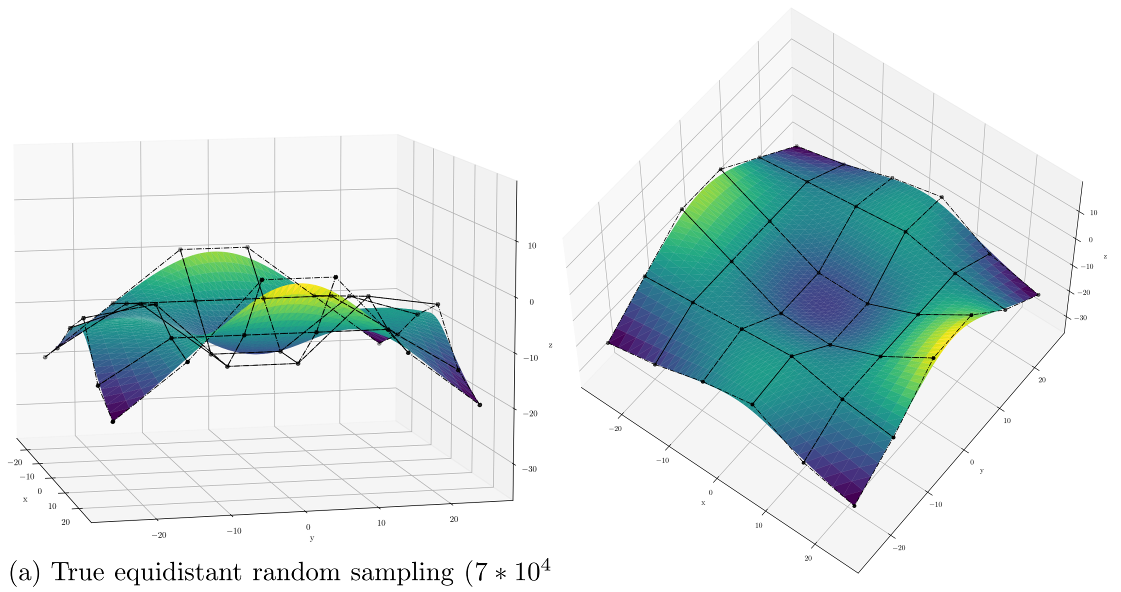
To form a surface, a network of control points G quads in $\mathbb{R}^3$ is constructed, see fig. 3.4. With a control grid G of size $m * o$ the surface B_s is then defined by the function:

$$B_s(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{o} b_3((u-3)-i) b_3((v-3)-j) V_{i,j} \tag{3.3}$$

Note that the knots are implicitly defined in Equi. 3.3 and 3.1. This is called a uniform B-Spline and uniform B-Spline surface respectively. Non-uniform B-Splines have an explicit knot vector. These explicit knot vectors change the basis function of each control point individually. Spline functions can also be defined by the knot positions and coefficients, since the knots and control points with coefficients can be transformed into each other [13]. This has three main advantages. It gives another and even finer way to control the shape of the curve or surface, Knot repetitions can form sharp corners and the usage of the virtual control points becomes superfluous.

3.1.3 Reconstruction

Since point clouds are essentially a collection of isolated points without explicit information about the continuous surfaces they represent, surface recovery techniques are used



(a) True equidistant random sampling ($7 * 10^4$ points)

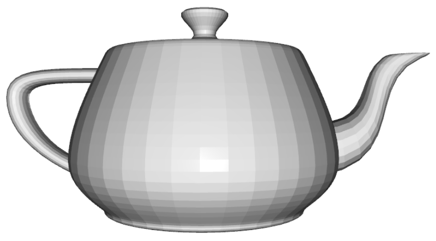
(b)

Figure 3.4: A cubic B-Spline surface with 36 equidistant control points defined from $x, y = (-20, 20)$ to $(20, 20)$ from two perspectives. The control points form a quad mesh

to create a more comprehensive and usable representation of the object or environment. There are several methods for surface recovery from point clouds, and the common approaches are mainly separated in two types. Firstly, mesh generation approaches. These techniques involve creating a mesh that represents the surface. Triangulation is often used to form the mesh with the points acting as vertices of the triangles. Secondly, direct surface fitting. In this method, mathematical models are employed to fit surfaces to the point cloud data. These models can be simple, like planes, spheres, or more complex, like B-splines.

3.1.3.1 Plane Fitting

One of the simplest geometric surface definitions is a plane. Many methods to fit planes to data points exist [31]. Very common are PCA and RANSAC based approaches. The PCA normal calculation can also be seen as fitting a plane to the cloud and using an orthogonal vector to that plane as the normal.


(a) Original

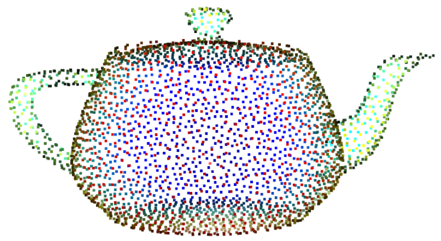

(b) Random sampling point cloud with 3000 points

Figure 3.5: The Utah Teapot. Used to demonstrate surface reconstruction techniques.

3.1.3.2 Bivariate Spline Fitting

Fitting a bivariate spline to a surface hast multiple advantages. It is able to smooth the data to reduce noise. For most surfaces it is safe to assume that the number of points is larger than needed to construct a surface model, (the surface fitting is overdetermined), meaning the number of resulting spline parameters, be it knots and coefficients or control points, is smaller than the number of data points, resulting in data reduction. Functional representation of the surface by the spline offers easy interpolation and limited but possible extrapolation. Fitting a spline poses multiple problems.

Fitting a spline to a surface can be defined by a special case of fitting any surface to scattered data following these conditions given by [22]:

1. fitting is significantly easier when the surface can be defined in a surjective function $f(x, y) \rightarrow - > z$, meaning one z value for every x, y pair.
2. the less steep the derivatives of this hypothetical function, the more accurate the fit.

The higher the order of the spline, the more complex the computations. Generally, [13, Chapter 8.2]suggest using splines of degree three for general purpose approximations. [12] defines a heuristic to place knots using a smoothing condition based on a least squares condition and a smoothing factor sf. Multiple suggestions for determining a good sf value exist depending on the data. [13, Chapter 9.2.4] suggest using $m \pm \sqrt{2m}$, taken from [40], with m being the number of data points, as a start value and then using trail an error. If sf is chosen to small, the spline will be overfit and oscillate strongly by picking up too much noise. While a small sf value can result in underfitting and too much smoothing. Spline extrapolation works by continuing the curvature of the last patch defined by the knots. The larger the patches in general, the more points of the original data influence the extrapolation. Since the patches get larger with more smoothing, this needs to be taken into account when choosing an sf value.

3.1.3.3 Poisson Surface Reconstruction

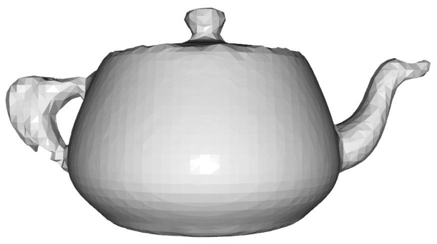
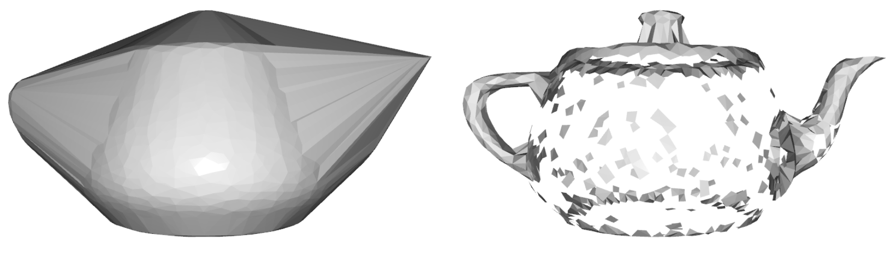


Figure 3.6: Poisson Surface Reconstruction with $o = 10$.

Poisson Surface Reconstruction is a mesh surface reconstruction technique (fig. 3.6). Poisson Surface Reconstruction needs the oriented points of a point cloud as input, e.g. the point surface normals pointing outwards of the surface describing the point cloud. The goal is to construct a function that indicates if a point is inside a model or outside. This function is constructed using the relationship between the flipped point normal (pointing inwards) and the gradient of the indicator function as a fitting condition. When interpreting the gradient of the indicator function as vector field, it will have vectors of size zero everywhere, since it has only two values, inside and outside, except for the boundary of the object. There these vectors need to align with the inverted normals. As a first step to fit the indicator function, the point cloud is sorted into an octree. The function is piece wise fitted in every octree level o and then joint together. o determines the fidelity of the indicator function.

3.1.3.4 Alphashapes

Alphashapes [16] is mesh generation technique based on the Delaunay triangulation. The mesh M_α, resulting from a valid Delaunay triangulation for a point cloud P_α has the property, that no vertex v_x can lie within a triangle t_y circumcircle if v_x is not used to construct triangle t_y. The convex hull is part of the Delaunay triangulation. Delaunay triangulation minimizes the sum of all triangle angles within M_α. [10]



(a) low α values create a convex hull (b) $1/\alpha = 0.2$

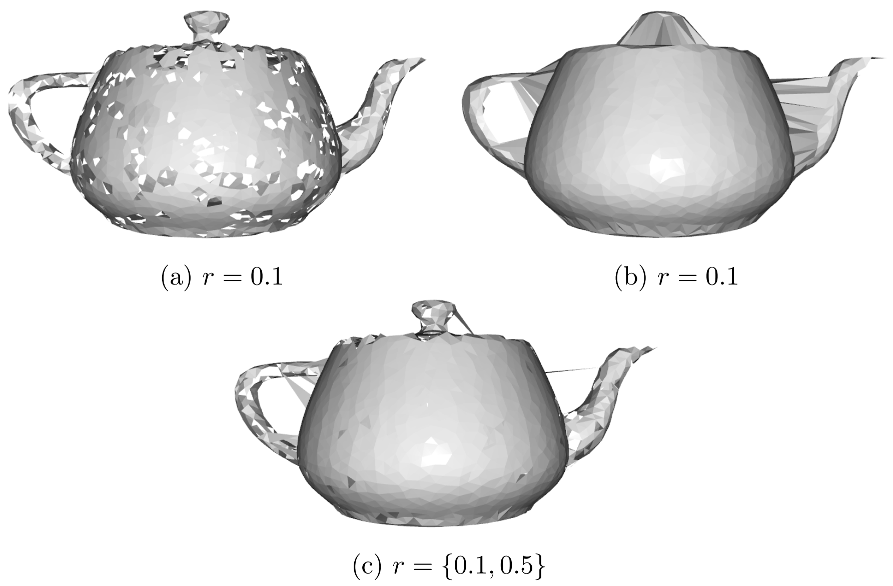
Figure 3.7: Aplphashapes: Low values capture the handle, spout and knob, but fail to capture the rest of the surface.

The fidelity of an alphashape is determined by its α parameter. An alphashape can be constructed by inspecting all triangles in M_α. Triangles, whose circumcircle radius r_t is larger than $1/\alpha$ will be dropped (Note that some implementations drop the reciprocal of α, reversing the effect of larger values). The union of all remaining triangles is the alphasahpe of P_α. A low α value will result in the convex hull, see fig. 3.7. Alphashapes suffer from a few limitations. The quality of the triangles is determined by the underlying point clouds and the meshes are not guaranteed to be manifold, even if boundaries are allowed.

3.1.3.5 Ball Pivoting

The basic idea behind ball pivoting is to approximate the surface by constructing triangles that connect neighboring points within a certain radius, simulating the idea of rolling balls along the point cloud and creating a surface whenever the ball falls between three points. The algorithm starts by selecting a seed point. A ball is created around the seed point with a specified radius r_b. The ball is rotated around this point, trying to find other points within its radius that can be connected to the seed point to form a triangle. When a valid point is found, a triangle is formed, and the newly added point becomes the next pivot point. Points that are already part of the surface are marked as "visited" and are not considered as pivot points again. The process is repeated by creating new balls around the newly added points (pivot points) and pivoting them to find additional points to form more triangles. The process continues until all points are visited, and the entire surface is reconstructed. [2]

A known limitation of the Ball Pivoting algorithm is, that it is very sensitive to the parameter for r_b. This problem can be partially mitigated by using multiple radii in


(a) $r = 0.1$

(b) $r = 0.1$

(c) $r = \{0.1, 0.5\}$

Figure 3.8: Ball Pivoting with a small radius (a), a large radius (b) and combined radii (c). While the small radius captures more detail, it also creates more holes in the model. The larger on the other hand, looses the holes of the teapot handle, spout and knob. Letting the radii pass over the model after each other gives a better middle ground result.

ascending order. When a pass with one radius finishes, a pass with the next bigger radius is started, using only the open edges from the previous path. Another problem can be, that Ball Pivoting will not always create well behaved meshes. The quality of the triangles is determined by the underlying point clouds and the meshes are not guaranteed to be manifold, even with boundaries allowed. But this can also be an advantage. It is able to correctly find holes in surfaces or deal with missing pieces of the input point cloud [14].

3.2 Clustering

Clustering is a data analysis technique used to group similar objects based on their intrinsic characteristics, for example, how close togehter a group of datapoints is in a euclidean space. It aims to identify patterns and relationships within a dataset by organizing datapoints in such a way, that datapoints within the same cluster exhibit higher similarity to each other than to items in other clusters. This process assists in uncovering underlying structures in the data and can aid in tasks such as classification, anomaly detection, and pattern recognition. Many different clustering methods exist, based on different principals, like fitting mathematical models, grid based, partitioning of data into known number of groups or density based [41]. Hierarchical Clustering methods produce a tree of clusters, enabling to choose finer or larger clusters that get merged together the higher up the tree one goes.

3.2.1 Density-Based Spatial Clustering of Applications with Noise

DBSCAN is a density based Clustering algorithm. It propagates point labels based on how dense a point is. The density of points is determined by the number of its neighbors.

To cluster a point cloud P, the point cloud is partioned into three types of points. Core, reachable and noise points. Core points have a minimum of min_{pts} neighbors points within a radius of ϵ. All core points are connected to its neighbors. Points that are connected to a core point but do not not fullfill the core point condition are reachable points. Points that are neither core of reachable are marked as noise. To label the points, a random Core point and a new label gets created. The label is then propagated over all connections to other core and reachable points, but never further then to a reachable point. Reachable points might be reachable from multiple core points belonging to different clusters. The label of those points then depend on the random choice of label start points. This flood fill like process is repeated until all Core points have a label. This enables DBSCAN to determine the number of clusters, they do not need to be known beforehand.

When not taking into account the nearest neighbor search and for non-ill-formed data and parameters, DBSCAN has a complexity of $\mathcal{O}(n)$. DBSCAN can only handle data with roughly the same density throughout the data, since min_{pts} and the ϵ radius are fixed. Meaning when they produce the desired clustering for one density, they mostly can not for another.

3.2.2 Hierarchical DBSCAN

Hierarchical DBSCAN (HDBSCAN) [7] is an evolution of the DBSCAN algorithm. HDBSCAN tries to takle the main problem of DBSCAN, that it can only work with roughly same density data throughout the dataset. It does this, by getting rid of the fixed ϵ

and min_{pts} value. Internally, HDBSCAN works different from DBSCAN. It is however related in the results it produces and which logic these results follow.

Since HDBSCAN is also density based, a measurement of density needs to found not based on a single radius ϵ. This is done using the mutual reachability distance d_{mreach} d_{mreach} is supposed to give a cheap estimate of a points density while simultaneously penalizing points with a lower density. Let m be the euclidean distance metric (other metrics would also be valid) and $knn_(p)$ be a function that extracts the k-Nearest Neighbors (kNN) of p with k equal to the parameter m_s. Then d_{mreach} is defined as:

$$d_{mreach}(p0, p1) = max(d_{mcore}(p0), d_{mcore}(p1), m(p1, p0)) \tag{3.4}$$

with:

$$d_{mcore}(p) = \max_{ninknn(p)} m(p, n) \tag{3.5}$$

d_{mcore} is called the core distance of a point. It is defined by the distance between the point p and the point in its kNN that is the furthest away by metric m. This is larger for sparse points, because their kNNs are more spread out. Since the kNN depends on the parameter m_s, the larger the parameter m_s, the further the points get spread apart. So the distnace between points in the mutual reachability distance is always at minimum spread apart to the larger of the core distance values, if $m(p1, p0)$ is not larger on its own.

The second step in HDBSCAN is to construct a minimum spanning tree (MST) of the dataset using the mutual reachability distance. Multiple algorithms exist to construct such a tree, for example Prims algorithm and the Dual Tree Boruvka (DTB)[9] which is significantly faster [34]. The MST is often the most expensive operation, depending on the shape of the data, with non-ill-formed data and parameters with a complexity of $\mathcal{O}(n \log n)$. A MST can only be constructed if all points are in some way connected to each other. This means, a distance matrix that is too sparse or has sparsity in exactly the wrong way or a distance matrix that results in two many infinity distance values might prevent the construction of MST, making the algorithm fail.

The third step is to perform a single linkage clustering on the MST minimum spanning tree. Single linkage clustering uses a botton up approach to construct a hierarchy of clusters. This is done by iterating over the edges of the tree. Every point starts as its own cluster. At each iteration, two clusters containing the nearest pair of elements p_s and p_t not yet belonging to the same cluster are merged together, creating a hierarchy of connected components. Let the distance between these pairs called δ_{st}. A hierarchy of clusters is not a satisfactory result. To find a definitive set of clusters, the hierarchy needs to be cut at some point. By doing this at a single level throughout the entire hierachy, one would replicate the results of DBSCAN which is not desireable.

The forth step is a reduction of the hierarchy into one with less branches. Every split in the hierarchy is connected to a certain δ value. Every cluster in the single linkage hierarchy has a number of containing points, with the root having all points and the leafs having one point. Starting from the root, drop all child nodes that contain less than m_c points. Instead, mark the δ value where this split would be and subtract the points from the cluster from this point onward. Only split the hierarchy when the split would create two clusters with minimum m_c points. Proceed with these clusters. The resulting hierarchy is called the condensed hierarchy.

With this condensed hierarchy the problem of cluster selection remains. To compute its clustering result, HDBSCAN sums up the lifetimes of points within their cluster. A point's lifetime in a cluster ends when it is dropped in the condensed hierarchy or the cluster is split. Using the δ values for the calculation is not possible. Instead, the reciprocal δ^{-1} is used. This problem becomes apparent when taking two hypothetical dense clusters that are very far apart as an example. Using the distance as a measurement, the cluster would live very long with all points in it. When using δ^{-1}, it will become basically zero leading to the desired result. The overall weight w_c of a cluster C can now be calculated:

$$w_c = \sum_{p \in C} \delta_p^{-1} - \delta_C^{-1} \tag{3.6}$$

With δ_C^{-1} being the split where the cluster is created and δ_p^{-1} the split where the point is dropped from the cluster. To create the cluster result, the cluster with the greatest weights whose child clusters in the hierarchy do not have a greater sum of weights gets seletced. The leave nodes are treated as they would have children with a weight sum of zero. Every point not selected in a cluster is marked as noise.

3.3 State of the Art

The problem of reconstructing surfaces from point clouds has been tackled in many ways and for very different applications. While methods like ball pivoting or Poisson surface reconstruction focus on reconstruction the point cloud of a single object, the techniques trying to recover objects in urban environments need to filter out regions that are of no interest for the application, for example, parked cars while surveying and separate multiple objects.

The focus in the literature reviewed in [51] has been mostly on TLS data, since it makes up arround 38% of the data used in suface reconstruction. MLS points clouds are used in 20% of the cases and ALS also arround 20% percent. The missing percentage are based off other technologies such as, for example, depth cameras.

The proposed methods are based on many principals for segmenting the object in a point cloud. [51] differentiates between model-based, region growing based, clustering based, energy optimization based and hybrid based. Model-based methods group points based on specific mathematical representations, such as spatial locations and normal vectors. Points fitting the same model are extracted as a segment. Region growing methods iteratively analyze neighboring points to determine if they belong to a region. Seed selection and growing criteria influence this process. Clustering-based methods relate adjacent positions based on spatial coordinates and geometric characteristics. They are then clustered using existing or modified clustering algorithms Clustering relies on criteria like Euclidean distance, normal vector angles, and density consistency. Energy optimization methods treat segmentation as energy minimization. They assign points to clusters to minimize costs by some function. Region growing approaches are widely implemented and are computationally inexpensive, while clusters approaches tend to better with noise and clutter in the data [32].

Many techniques are targeted to building reconstruction and are not suitable to general surface recovery from points clouds. When it comes to large points clouds to reconstruct large areas, most techniques proposed are limited to ALS point clouds, because they make certain assumptions about the structure of the point cloud, mostly that it is strictly 2.5D.

[29] uses a neural network to estimate the number of planes in a point cloud of arbitrary origin. The planes are then clustered with a method called hybrid k-means. It combines the standard euclidean k-mean clustering algorithm with the spherical k-Means based on

the cosine distance of the point normals. The method performs no reconstruction. [42] works similarly, but only clustering the surface normals. For this to work, the buildings in the data are previously separated. [39] also uses a clustering approach based on the normals. Each point is represented using a six-dimensional vector describing the local geometry using the components of the point normal and the height of the point, as well as the local height variance and the normal variance around the neighbors of the point. The points are then clustered using a custom clustering algorithm that takes the point position into account by only connecting points within a certain kNN.

Many algorithms first apply a classification step. Generally, this is done to separate out anything that cannot be recovered as a surface very well. Most algorithms use some combination of the categories ground, building (or roof for ALS), clutter and vegetation. [32] uses complete linkage with point postilions and normals to segment planar structures from point clouds. The normals are determined by a RPCA method. [28] uses such a segmentation step to separate these classes and treat them differently. Buildings are reconstructed using region growing with the angle between the point normal being the criteria to grow the region between two points. The found regions are then fitted with planes, sphere or cylinders based on what fits best. Regions fitted with planes are then used to close gaps between them. For this, the 2.5D propertie of ALS data is assumed. The planes are then extended iteratively using a grid structure on the x, y plane. If two planes meet in a grid, the intersection is calculated in this cell. [52] also uses such a classification step to remove the ground between buildings. To separate the buildings itself, a connected component clustering is used. This also segments different height regions of a roof belonging to a single building in the used ALS data. Now, the building outline is detected using alphashapes on the x, y components of the points. If the outline is curved or not fine enough by a certain metric, the edges are upsampled similarly to Edge-Aware Point Set Resampling [24] and the algorithm repeated. The found outlines are then elevated to the average height of the region they stemmed from. This produces only flat rooftops. [46] uses multiple passes of a clustering algorithm to get all Roottops from ALS data. Every rooftop is then aligned to a two-dimensional grid. The occupancy of this grid determines the covered area of the rooftop.

A user interactive method called SmartBoxes is given in [36]. It is specifically tailored to MLS point clouds with a lot of missing data. Planes are fitted to the point cloud strictly vertically or horizontally, which can than be combined to models by the user by construction edges and filling gaps.

PolyFit [37] and City3D [25] both extract planes from the point cloud. They than calculate the intersection of all planes within a region treating them as they would have infinite extend. Planes with infinite extend always intersect if they are not perfectly parallel. Finding the correct intersections to limit the plane extend is then performed by solving and optimization problem, which is very computationally intense. While Poylfit can handle MLS point clouds, City3D is making ALS assumptions to optimize the intersection finding and construct vertical walls.

3.4 Previous Works

This work uses the ideas of [5] as its foundation. [3] used a modified DBSCAN to cluster together points with similar normals. First, the point cloud was segmented into noise, flat and one dimensional points using PCA and the resulting L_1, L_2 and L_3 eigenvalues like described in Sec. 3.1.1.2.

The normals were calculated using the first two eigenvectors. Instead of using the normals in its distance metric, the DBSCAN algorithm was modified in its connection building behavior. For points to be connected, the connected point and core point condition were checked normally. Then, as an added restriction, the point normals must be at least within a specified angle β. This effectivly changed the distance metric m_a between two points $p1$ and $p2$ from euclidean to:

$$m_a(p1, p2) = \begin{cases} \infty & \text{if } \angle(p2_n, p1_n) > \beta \\ \|p2_{xyz} - p1_{xyz}\|_2 & \text{if } \angle(p2_n, p1_n) <= \beta \end{cases} \tag{3.7}$$

While it worked on a small synthetic dataset, it had problems working on the KITTI [11] dataset. Especially in the presence of noise the algorithm produces unsatisfacotry results. Also, tuning the DBSCAN parameters for the uneven resolution of LiDAR point clouds turned out to be very difficult to impossible. The clustering results were also highly dependent on the quality of the normals. PCAs susceptibility to noise was tackled by increasing the neighborhood radius of the point the PCA was calculated for. The increased radius of the PCA also tends to smooth the normals over sharp corners and edges. For features in the point cloud that are close together, a large radius can have the unintended effect of "merging" the features together, resulting in a large L_3 values, and thereby classifying the point as noise. The found clusters were not converted into a real surface description.

An effort to fix these issues was made in [5]. First, DBSCAN was swapped out for HDBSCAN to tackle the problem of DBSCAN not being able to work with the different data densities within one point cloud. Since HDBSCAN needs a well-behaved distance metric, using the DBSCAN like distance metric Equi. 3.7 would not work. Instead, a new metric was formed, which added the normalized angle between the normal with a weight factor ng to the euclidean distance.

$$\Delta_{op}(p1, p2) = \|p2_{xyz} - p1_{xyz}\|_2 + ng * \angle(p2_n, p1_n)/\pi \tag{3.8}$$

This is called the oriented point distance. This posed the problem, that the distance could not be calculated efficiently using existing HDBSCAN implementations and spatial acceleration data structures. To solve this, the original metric Δ_{op} was approximated with Δ_e by also using the euclidean metric for the distance between the normals:

$$\Delta_e(p1, p2) = \|p2_xyz - p1_xyz\|_2 + ng * \|p2_n, p1_n\|_2 /2 \tag{3.9}$$

With PCA being so susceptible to noise, a different noise robust PCA method was used, Dual Principal Component Persuit (DPCP) [15]. This greatly improved the problem with points close to surface singularities being classified as noise. The clusters found by HDBSCAN were converted to a real surface description by fitting a plane using DPCP and finding the outline of this plane using alphashapes. These improvements gave overall usable and better results then the DBSCAN variant.

There were still some problems with the approach. The performance of DPCP and HDBSCAN were not great. Point clouds needed to be downsampled to make it work, costing accuracy. A large radius for DPCP also proved to be a significant performance bottleneck. Also, HDBSCAN sometimes still clustered planes with different orientation together. Trying to correct these issues by increasing ng resulted in over segmentation elsewhere in the data. Since all clusters were fitted to planes, only surfaces with roughly planar structure were accurately represented. The alphashapes approach to find the outlines of the fitted plane often resulted in low density areas of the data being mangled into single polygons, while still failing to find the accurate concave outlines in the high density area.

To check if preprocessing the point cloud with different preprocessing techniques like smoothing, noise reduction, upsampling near singularities, better normal calculation, the HDBSCAN based method was combined with such methods in [4]. The HDBSCAN method pairs were then optimized using a parameter optimization framework [1] and an objective function based on the hausdorff distance between found and ground truth planes. This revealed no significant advantages of one method. It even showed, when analyzing the parameter importance, that the HDBSCAN parameters were way more important than any other parameters and that HDBSCAN was able to handle the noisy normals and corners produced by PCA, showing that using DPCP instead produces not significantly better results.

3.5 LiDAR Point Cloud Datasets

LiDAR point cloud datasets are the basis for developing and algorithm for surface reconstruction. There are two main types of datasets that can be used for evaluation. Synthetic datasets and real world recorded datasets. To create a synthetic dataset, a LiDAR gets simulated in a virtual environment. This virtual environment can be static or dynamic, with simulated cars, pedestrians or weather. The realism of the datasets is largely determined by the quality of the lidar simulation. Many different models to simulate a lidar exist [33]. The virtual envoironments are usually defined using meshes. This enables easy ground truth data generation from these meshes.

Real world datasets are recorded using real LiDARS, mostly in uncontrolled real environments. Datasets created in controlled environments, for example in a lab or other large indoor space, are usually small in scale, since the complexity of setting up such an environment is rather large. Normally, no exact 3D model of the environment exists. Ground truth data for these dataset is created in a tedious manual process.

3.5.1 Datasets and Tools

Virtual Kitti Virtual Kitti and Virtual Kitti 2 are datasets of a simulated car using onboard cameras, created with unity, see fig. 3.9b. Sadly, it offers only camera data with object segmentation, classification, optical flow and depth ground truth. While the ground truth depth data could be transformed into something like a LiDAR point cloud, the assets used to create the simulation are not given, meaning no ground truth is available for the surfaces. Virual Kitti offers 4 different scenes, urban and highway, captured in short sequences of driving during different weather conditions like rain and fog, as well as different times of day. The scenes are supplied as a sequence of camera frames.

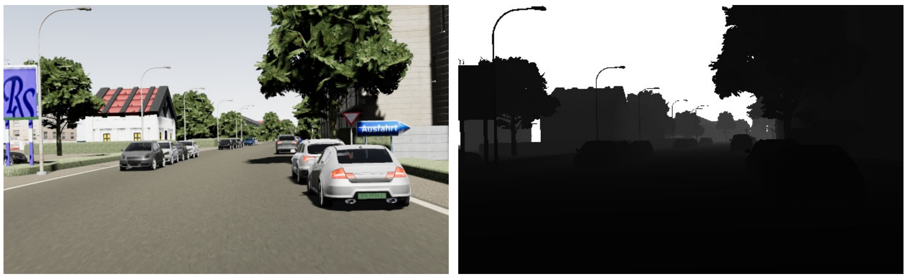

(a) Camera view (b) Depth View

Figure 3.9: Virtual KITTI 2 Dataset (Scene 01, Clone). Camera picture with corresponding depth image

Carla Simulator Carla is an Unreal Engine based simulator for automotive autonomous driving. It can simulate traffic and supports controlling one or multiple cars, but the traffic can also be turned off. The simulator supports a wide range of sensors, including LiDAR. The sensor can be configured with a wide range of parameters. The position of a sensor must be given fixed to a car. Carla offers multiple maps from urban to rural environments. The maps generally scales from a few city blocks to a small town. Ground truth data can be generated using CARLA and the whole editor is available with a project for a purpose modified Unreal Engine. LiDAR datasets have been created using the CARLA Simulator [11].

Blensor Blensor [21] is a modified version of the free Blender 3D modelling program. Blensor has 4 build in sensors modeled after real LiDAR models. There is no test data included, meaning an urban scene would need to the supplied to blensor.

SynthCity SynthCity [19] is a synthetic LiDAR dateset generated from a realistic high resolution 3D mesh city model. The LiDAR data was generated using Blensor. While the ground truth city model for SynthCity exists, it is not available for free and must be bought from its original creator. The area covered by the synth city dataset is rather large with the entire point cloud being nearly 30 GB of data.

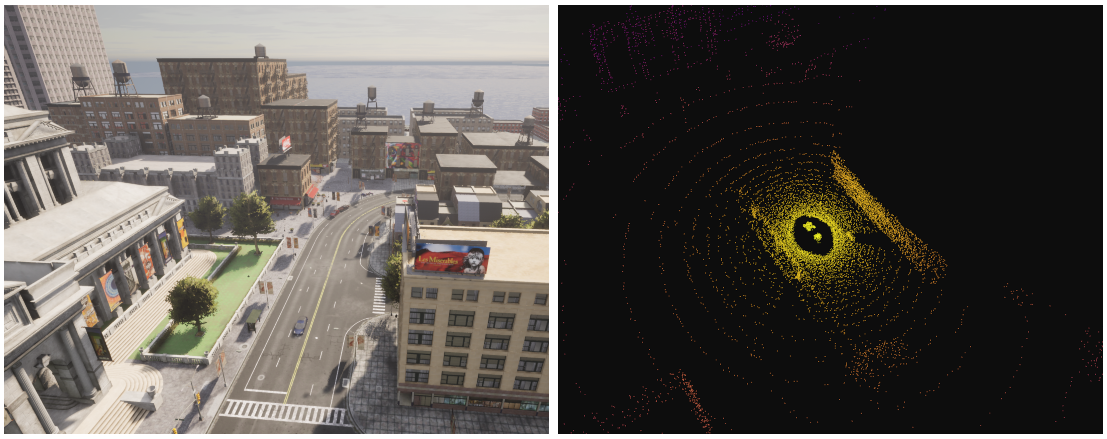

(a) Rendering of CARLA map (b) Default CARLA LiDAR setup, single frame

Figure 3.10: Render of a CARLA map and a LiDAR frame of a car entering the scene from below the camera

The Newer College Dataset The Newer College dataset [20] is a small dataset recorded using a handheld LiDAR traversing parts of the New College of the University of Oxford. It features mainly historic facades and parkland. The dataset offers a low resolution point cloud generated by a millimeter precise TLS as ground truth data, with the high resolution low accuracy MLS LiDAR data matched to the low resolution high accuracy point cloud by iterative closest point methods.

SynLiDAR SynLiDAR [50] is a very large synthetic dataset. It covers mostly suburban american style streets, captured from a simulated car. The models used for the simulation are not supplied. The used models are low resolution and the virtual environment relatively empty.

HoliCity HoliCity[54] is a dataset containing panoramic views of London. It is aimed at training machine learning approaches for image segmentation and depth estimation. These panoramic views are mapped to a high resolution, high accuracy model of downtown London. The panoramic images are single location and single image dotted over the map. The views contain no depth information.

HAW Hamburg Dataset [43] generated a large dataset with a handheld LiDAR mainly containing the main campus of the University of Applied Sciences Hamburg. The

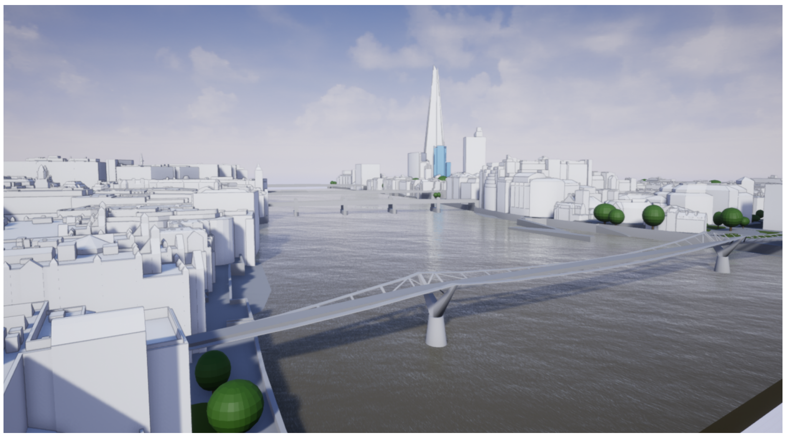


Figure 3.11: Used by the HoliCity dataset, rendered with Unreal Engine 4

datasets show vegetation, streets, buildings and areas not accessible by car. The set is purely LiDAR data with no other information given.

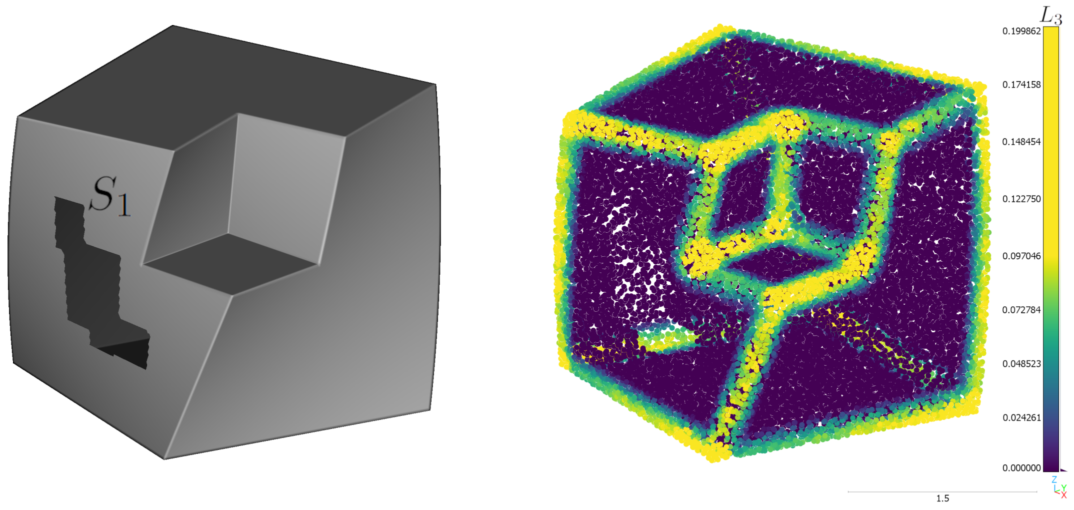
Data with HD Maps ArgoVerse 2 [48], NuScenes [6] and 3DHD City Scenes [38] all contain an HD Map and LiDAR data. They are all focused on atunomous driving cars. ArgoVerse 2 and NuScenes contain lane-level geometry. ArgoVerse 2 additionally has a rasterized ground truth map. 3DHD City Scenes contains streets, lanes and annotated signage and line markings. None of the datasets contains buildings.

4 Method

To close the research gap considering the surface recovery from MLS LiDAR data for usage as HD Map geometry, an algorithm is developed tuned to the specific problems of MLS LiDAR data. This algorithm results shall then be evaluated against some kind of ground truth data, to see if the generated data fits the requirements for High Definition Maps. Since no known good LiDAR dataset with ground truth exists, such a dataset shall be constructed with one of the available tools. Because there always exist discrepancies between such synthetic and real world datasets, the algorithm shall also be tested on a real world dataset and the results be visually inspected for their quality.

4.1 Algorithm

The algorithm shall recover a well-behaved geometric representation from a point cloud representation of the surfaces that were captured by a MLS LiDAR. To achieve this, the algorithm needs to be tolerant to the aforementioned problems of such datasets like noise, incomplete data and scan shadows. This representation shall be significantly smaller in size. Also, the algorithm shall filter out any non-surface data such as vegetation or outliers caused by reflections, dust or other effects. The main problem of large MLS data is its size. While algorithms dealing with such big datasets exits, they focus mainly on ALS data and make certain assumptions about the data, for example that it is 2.5 dimensional [51] [46]. While the produced outputs have good visual appearance with watertight meshes and simple forms [25], the accuracy can be quite bad for surfaces not fitting the used model ideally. This is quite common, since many buildings are not composed of simple shapes or do not adhere to them as tightly as it sometimes seems. Older buildings for example, seem rectangular, but are usually deformed in many ways. Fitting more complex models to large MLS datasets directly is nearly impossible and requires some kind of segmentation beforehand. Watertightness is of no concern for HD Map applications, so holes in the data are acceptable. Other algorithms find single



(a) Example cube with non planar surfaces, and a hole in surface $S1$ (b) The cube with normals calculated with PCA with $r = 0.2$ and the L_3 displayed

Figure 4.1: Sample twisted cube used to demonstrate the algorithm basics

surfaces and then combine them to one big model by solving an optimization problem. The results are very appealing, but the runtimes are too high to be considered for large datasets. The runtime and memory requirements are a general problem for large datasets, since they often can not be treated all at once, at least on a computer with of the shelf hardware, even if it is considered high performance.

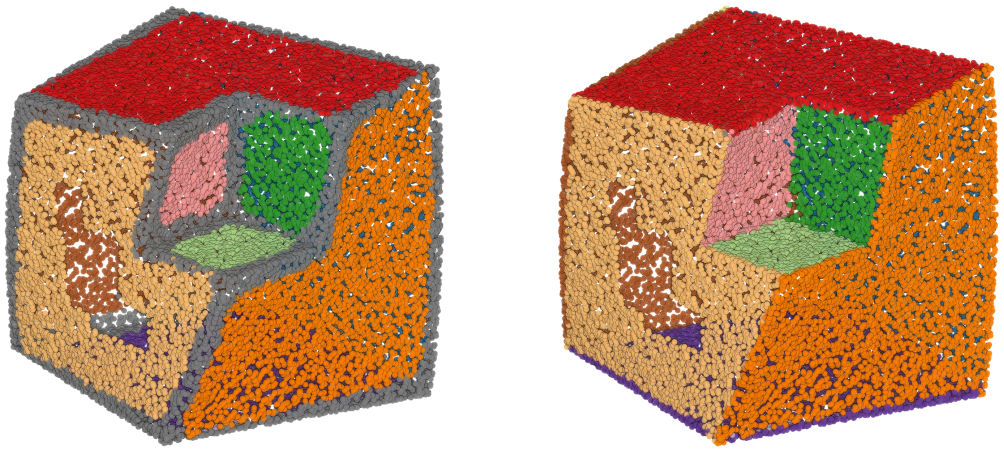
The first step to designing an algorithm that can deal with these problems, is the filtering of all non-surface like data. Points can be classified by computing surface features of these points. This has the advantage, that many algorithms need some kind of surface normal to work, which can also be calculated in this step. A problem when filtering non-surface like data are edges (C_1 discontinuities). These often present itself also as non-planar. Another problem with such discontinuities is that they are inherently undersampled [24]. These two problems can be solved by starting a surface segmentation. This segmentation will seperate all surface like structures into at least $C1$ continuous patches, that will have some distance to the next edge at their maximum extent. Then, a surface representation that is able to be extrapolated can be fitted to these patches. By extending these surfaces, the edges can be recovered. Different strategies for this exist [24] [37] [25]. Usually, a surface that can be extrapolated has either an unlimited extent or at least one that overrepresents the original surface area. The area can be limited to an approximation of the original surface area by finding an outline by using the points associated with it.

4.1.1 Non-Surface Filtering

The filtering of non-planar object and the calculation of the surface normals will be performed by PCA. For this, all points within a radius r around one point will be used to calculate the eigenvectors and principal components of this point. The normal will be determined by the principal components and oriented using the vector pointing towards the LiDAR position that recorded that point. Non-surface structure will be removed by L_3 filtering. An example that is often present is foliage. The fine structure can not really be captured by LiDAR and presents itself as a volume of points with more or less random distribution. The points in these areas have a higher L_3 value. By defining a L_3-theshold t_{L_3}, these areas can be filtered out. This filtering also removes edges and corners, see fig. 4.1b.

4.1.2 Surface Segmentation

The segmentation principal is the same as used in [3], with HDBSCAN and the oriented point distance Δ_e. HDBSCAN offers many advantages over DBSCAN, especially for point clouds with uneven distribution of points. Two disadvantages of the HDBSCAN approach compared to DBSCAN are, that the produced clustering is less local and the higher runtime complexity and overall runtime. The HDBSCAN clustering depends more on the overall structure of the data while DBSCAN depends more on the local structure of the data. This effect is created by the cluster selection algorithm of HDBSCAN. If a big cluster containing multiple smaller ones is more stable than the sum of these smaller clusters, it is selected. This can be imagined as varying the DBSCAN parameters to favor clusters of a certain size. Since these parameters are fixed in parametrization of DBSCAN, the cluster scale does not depend on the cluster neighborhood. This and the runtime issues make it hard to tune the HDBSCAN parameters m_s and m_c to produce sensible results when clustering extremely larger point clouds, especially since the clustering can take hours. Both the runtime issue and the locality issue can be solved by chunking the point cloud. The parameters can then be tuned on one, or multiple to be more exact, chunks of the point cloud. When a fitting parameter set is found, it can be applied to the other chunks. An important factor when chunking the data is the chunk size. The chunks should be as large as possible, since smaller chunks tend to cut clusters in half more frequently.



(a) Clustered PCL of fig. 4.1b with $t_{L_3} = 0.2$, $r = 0.2$

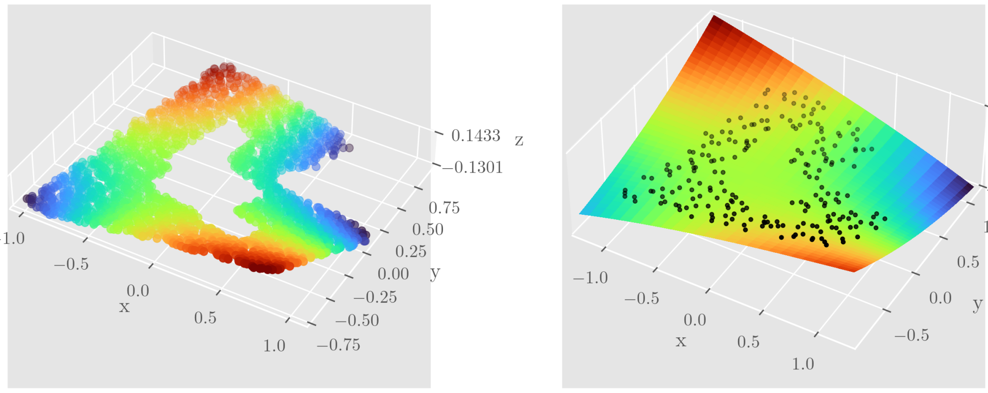
(b) Surface patch extended via fitted spline, with $t_\Delta = 0.05$ and $d_s = 0.3$

Figure 4.2: Twisted cube clustering and cluster extension step

To segment not only planar surfaces but also C_1 and above, the usage of HDBSCAN becomes critical. For planar surfaces, when ignoring outliers and noise, the points of the surface do not get spread apart by Δ_e, since their normals are aligned anyway. For any other non-planar C_1 and above continuous surface, the normals are not equal across the entire surface, meaning they get spread apart by Δ_e. This could limit ng, since non-planar surfaces should not be spread apart too much. Here, the L_3 filtering is beneficial. Since it removes edges, it creates gaps between surfaces separated by a C_1 discontinuity. These gaps lower the overall density of the surrounding points, spreading them further apart. For fitting only planar structures, this could be done by increasing ng, which is not possible here because of the aforementioned problem.

4.1.3 Surface Fitting

While many surface representations exist, splines have the advantage that they can represent non-planar and planar surfaces with relatively few parameters as long as they do not contain sharp corners and no changing steep curvatures, which is mostly not the case in urban environments. In contrary, a mesh representation would need many vertices and triages to accurately capture a slightly curved surface of, for example, a facade.



(a) Surface $S1$ cluster $C1$ transformed to spline space V_{C1}

(b) The spline B_{C1} fitted to the $C1$ with smoothing factor $sf = 1$, every tenth point of $C1_B$ displayed

Figure 4.3: Spline fitting and expansion

To fit B-splines accurately, the two conditions mentioned in Sec. 3.1.3.2 must be met. Adhering to these conditions requires transforming the found clusters. Imagine a cluster representing an open half sphere, segmented as a continuous surface by the HDBSCAN step, with the open side of the half spehere standing vertically. The underlying function of this half sphere has multiple z values for one x, y pair, breaking the surjectivity condition. The breakage of this condition can accure with many different curved surfaces and surface orientations. A possible way to avoid this is, for example, to lay the sphere flat onto the x, y-Plane.

This transformation, a change of basis for any given cluster C, can also be calculated using PCA. When calculating the PCA without dimensional reduction, the principal component in matrix form a transformation P_c that achieves this. Transforming back to the original point cloud space can be done using the inverse of this matrix P_c^{-1}. For the sake of simplicity, the centering and normalization of the data required by PCA is assumed to be part of P_t and vice versa for P_c^{-1}. The space where P_c transforms to is here called the spline space V_c. Note that this will flatten pole like structures. But since they are not the main interest here, this is deemed acceptable.

With C transformed to spline space, a spline B_c can be fitted using the method in Sec. 3.1.3.2. For this, a smoothing factor must be determined. If no smoothing factor is given, noise will be present in the fitted spline and basically not reduction of data will be achieved, since every single point in C would basically need a knot definition to be exactly on the spline. Since the sf suggestions from [13] are data size depend, here

$sf = S * m$ defines a smoothing factor by multiplying a smoothing aggressiveness value S with the number of points m. This smoothing factor generally controls how true the fit to the original data shall be. While HDBSCAN is robust to noise, it does only remove outliers but not noise. Since B_c was fitted in V_c, it can only be evaluated in V_c. Noise removal is achieved by transforming C to V_c with P_c (fig. 4.3), evaluating the B_c at the transformed clusters x, y-positions and transforming back with P_c^{-1}, calling the result C_B. The combination of B_c, C_B is here called a surface patch S_c.

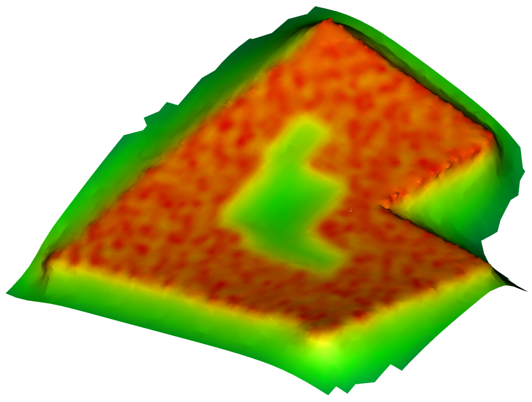
$$C_B = B_c(C * P_c * \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}) * P_c^{-1} \tag{4.1}$$

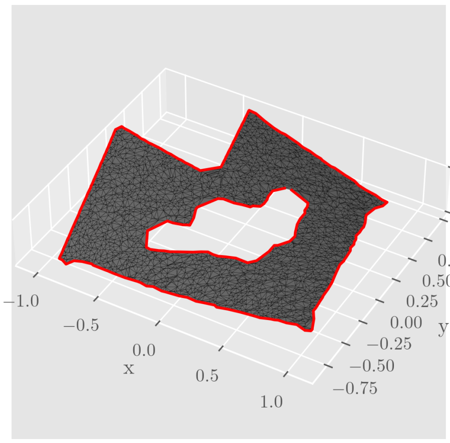
4.1.4 Surface Patch Expansion

The $L3$ filtering and HDBSCAN produces a lot of points that are classified as not belonging to any cluster. This can be seen in fig. 4.2a. While this is by design, it would lead to huge gaps between the found surfaces and would underrepresent their surface area. To mitigate this, the clusters are extended using the fitted splines. Let N be the set of points not beloging to any cluster. Now, for every cluster C_b find the points within N that are within a distance d_s from any point in C_b. The set of these points N_{C_b} are the candidate points for cluster expansion. For every point n_i, transform it to spline space and calculate its distance from the spline by evaluating the spline at its x, y-component and calculating the z difference. For every point in N, propagate the label of the surface patch, who's spline was the closest to that point. To avoid propagating labels to points that are too far off, a maximal distance t_Δ that a point can have to the spline is defined to be considered for label propagation. These points where the labels were propagated to are also mapped onto a spline using equi. 4.1. The effect of propagating the label is seen in fig. 4.2b.

4.1.5 Outline

Multiple algorithms exists to find the concave outline of a polygon in 2D. These could be applied to C_b mapped to V_c with the z component dropped. This approach would have the disadvantage that it would introduce a large error in the outline when the curvature of B_c is large, since the distance between the points would be distorted by the projection.





(a) Poisson reconstruction of $S1$ showing the problems of the method with open edges. Color density estimate of the points.

(b) Ball pivoting reconstruction with $H = \{5.0, 6.0\}$. The non manifold boundary edges in red

Figure 4.4: Poisson reconstruction and ball pivtoting result to find outlines

This could be avoided by applying such algorithm on the 2D surface defined by the spline instead of a flattened version, but this is computationally and mathematically much more difficult. An approximation of doing this to perform a mesh generation on the original C_b and use the non-manifold boundary edges as the outline. Poisson surface reconstruction is often superior to alphashapes and ball pivoting, but it performs best and is designed for point clouds of closed surfaces without holes. This makes it less suitable to find the outlines of a surface patch, see fig. 4.4a. Aplphashapes was not applied successfully in [5]. Ball pivoting is robust to the aforementioned issues. Parameterization of ball pivoting is the numbers of radii and their values. To estimate this parameter for surface patch, the following technique is applied. The lower bound value for any point is the distance to its closest neighbor to from any edge. Taking the average of these distances for all points in C_b results in a lower bound estimate a for the entire cluster. The number and size of radii is then determined by a factor set $H = \{h_1, \dots, h_n\}$, resulting in radii of the following form $r = \{h_1 a, h_2 a, \dots, h_n a\}$. The H set allows tuning the ball pivoting globally while the a estimate adapts this tuning to each cluster.

The boundary edges produced by ball pivoting are an unordered list of edges. To transform them to outlines, the edges are split into connected components (an object can have multiple outlines, for example the hole in $S1$) and an eulerian circuit [17] is constructed on the points of the edges with loop detection, since outlines can share a vertex sometimes.

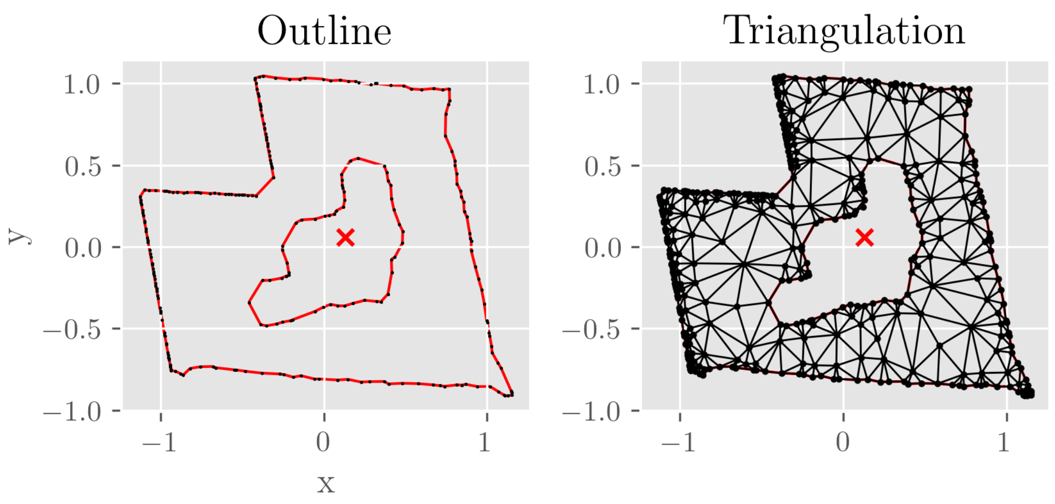


Figure 4.5: Triangulation of $S1$

4.1.6 Triangulation

Triangulation of the result is only needed for the visualization. The triangulation from the ball pivoting step is not used. This triangulation has multiple problems. It is not necessarily well-behaved, also areas with low density of points, like clusters with a big extent and sparse points, might be triangulated way too coarse. See sec. 4.2 for details.

4.2 Implementation

The algorithms described in Sec. 4.1 was largely implemented using *Python* 3.10.6. The PCA calculations for the point normals and L_3 values were performed using *Cloud-Compare* v2.12.4 [1]. To chunk the points clouds, a script was written that performs the chunking of the point cloud from its minimal x, y coordinate onwards with a fixed grid size given. The chunking was only performed in x, y direction, because the point clouds are dominated by these dimensions anyway. The clustering was performed using the Python package *hdbscan* v0.8.32 [35]. It offers a multithreaded implementation of HDBSCAN where possible in the HDBSCAN steps.

The clustered chunks were then split into their individual clusters but kept grouped together based on their origin cluster. The mapping of the clusters to the spline space was

[1] https://www.danielgm.net/cc/, visited 11.08.2023, 18:14

done using the PCA component of the *sklearn* 1.3.0 library [2]. The fitting an expanding the surface patch was done single threaded per chunk groups of clusters. First, for every cluster, there was a spline fitted. This was performed using the *SciPy* v1.11.1 [3] interface to the FITPACK library described in [13]. The spline evaluation was also done using these libraries. These splines were then saved with the cluster ID into a hashmap.

Now, for every point in the chunk that did not belong to a cluster, it was tested if one of their neighbors belong to a cluster. This was accelerated with the kd-tree of *SciPy* For every point that did, the spline was looked up in the hashmap and the point tested for the best fitting spline. The point was then added to the surface patch of the spline expending it. This was accelerated using the Python multiprocessing library, treating each Chunk in its own thread using a processpool to avoid the Python global interpolator lock.

The outline determination is now independent of all other surface patches, so the surface patches were degrouped and put into one list. The outline computation was then done in a processpool using the Python multiprocessing library again. To perform the ball pivoting, the algorithm from [14] was used. It was used single threaded, because the multithreaded variant had a tendency to crash and usually there are so many surfaces patches, that more than 20 process pool workers are active anyway. The eulerian walk was done using the *networkx* v3.1 package [4]

The Triangulation was done using the triangle library supplied by [44]. The triangulation serves no other purpose than to display the results.

4.3 Dataset

To evaluate the proposed algorithm, two different things are needed:

1. A point cloud that is recorded or similar in structure to one recorded with a ground based LiDAR

2. Ground truth geometry in form a well-behaved surface representation

[2]https://scikit-learn.org/, visited 11.08.2023, 19:00
[3]https://scipy.org/, visited 11.08.2023, 18:40
[4]https://networkx.org/, visited 11.08.2023, 18:56

The first condition is needed since it is the base input the algorithm is designed towards to handle. Point cloud generated from depth cameras or equidistantly sampled geometry pose different problems. The second condition is needed to evaluate the algorithm. The ground truth data should fit the goal of the algorithm, to extract surface features but leave out relatively fine details, to be a good base for evaluation. Also, the points of the point cloud should be in some kind connected to the ground truth data. This makes checking the clustering step of the algorithm much easier. The dataset must also fit the definition of an HD Map. It should have a relatively large extent and shall not only feature data recorded from roads, since HD Maps should offer more than maps for car centric transportation.

4.3.1 Selection

None if the synthetic or real world Datasets offer the needed ground truth data. While the The Newer College Dataset [20] has very accurate data to compare against, this data in not dense enough to evaluate how much of a surface was recovered, it would, however, by suitable to check how well a surface description fits the datapoints when ignoring the extent. To achieve a good evaluation, it was decided to construct a synthetic dataset with existing tools and ground truth data. For a real world dataset, the HAW Hamburg Dataset was chosen. The reason being, that with not ground truth data present, the evaluation must be performed visually, which is easier if one has access to the space being mapped.

4.3.2 Construction

A suitable real world dataset was found. But this dataset can not be used to measure the quality of the results quantitatively, since no ground truth for the recorded surfaces exists. No dataset of the synthetic evaluated ones provides the needed ground truth geometry to evaluate the proposed algorithm. To evaluate the algorithm, such a dataset needs to be constructed. Ideally, such dataset is constructed from some kind of well behaved surface representation. Not only can the point cloud be constructed from such a surface representation, but also the ground truth geometry. Constructing such surface representation, and in a realistic manner, so the results are comparable to real world data, is very complex and time consuming. CARLA provided good point cloud data, but had some issues. To avoid starting from zero, the geometry provided by CARLA was

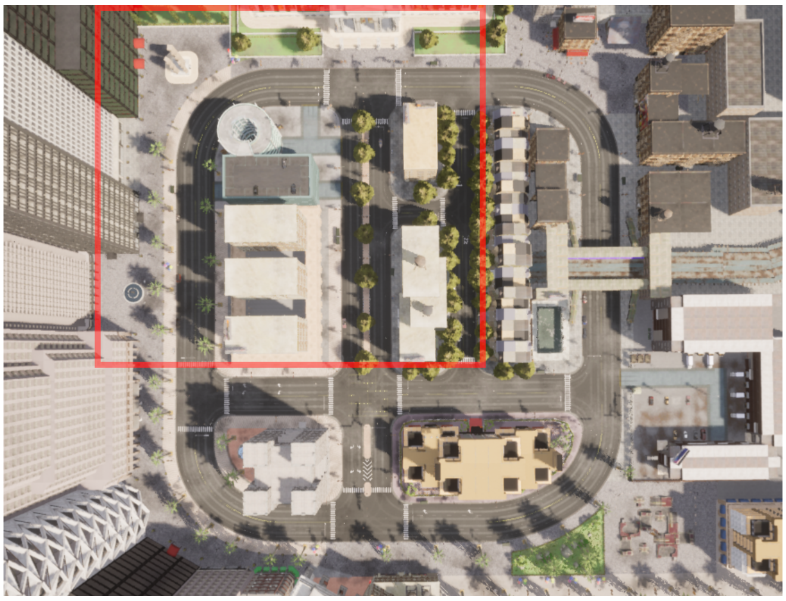


Figure 4.6: Map 10 of the CARLA simulator viewed from above. The region of interest marked red

chosen. Map 10 of CARLA is an interesting urban environment to test the algorithm. It contains interesting geometries that are not simply flat, for example a round, spiraling parking garage ramp, a helix shaped statue, and a combination of modern and historic style building. The entire map is quite large, so only the most interesting part was chosen to be the main focus, to keep the evaluation simple and amount of data manageable.

Since the point clouds of CARLA contained errors and there was no way to get a point cloud ground truth relation from the simulator, the simulator geometry need to be exported.

The maps of CARLA are given as Unreal Engine 4 levels. These levels can be exported from the custom Unreal Engine Editor of CARLA into a mesh representation. Sadly, the CARLA Unreal Engine editor contains a bug that crashes the export if certain types of objects and foliage, noticeable trees, are also exported. This is why they are missing in fig. 4.7.

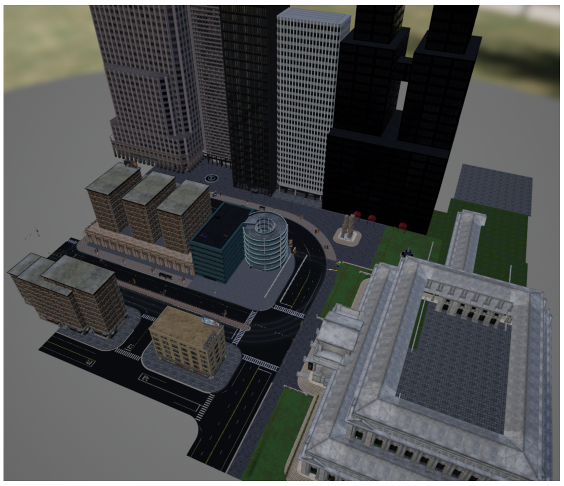


Figure 4.7: The red area from fig. 4.6 exported as a single mesh

4.3.2.1 Point Cloud

To construct a point cloud from the Map 10 mesh, a LiDAR traversing the map needs to simulated. Here it was decided to let the LiDAR traverse four paths around the most interesting structures in the map, such as the sculpture, the old style building and the round parking house ramp, see fig. 4.8. These structures should show the advantages of using non-planar surfaces and at the same time, be challenging to the algorithm. The paths were chosen to be collateral to each other to encurage the formation of the typical MLS LiDAR data defect in the point cloud, such as scan shadows and different densities of data which stem from limit LiDAR view angles and FOVs. The LiDAR is moved along these paths once with a speed of $10\,\mathrm{m\,s^{-1}}$. The speed determines the density of the point cloud together with the sampling frequency of the LiDAR.

The simulated LiDAR is modeled after a very common real one. The chosen LiDAR is the Velodyne Puck, formerly called VLP-16. It is a very common rotating 360° LiDAR, so the simulated LiDAR can be checked against real world data. Also, it is one of the few LiDARs where independently measured accuracy data is available [8] [18]. The characteristic of the Velodyne Puck are seen in the table below.

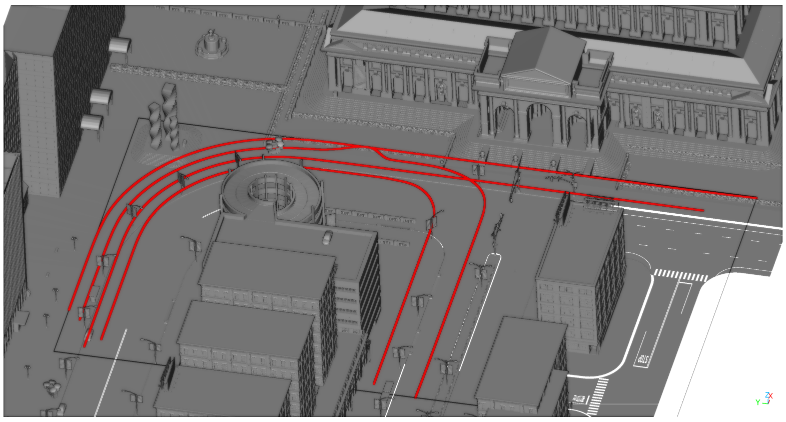


Figure 4.8: LiDAR path trough the test data

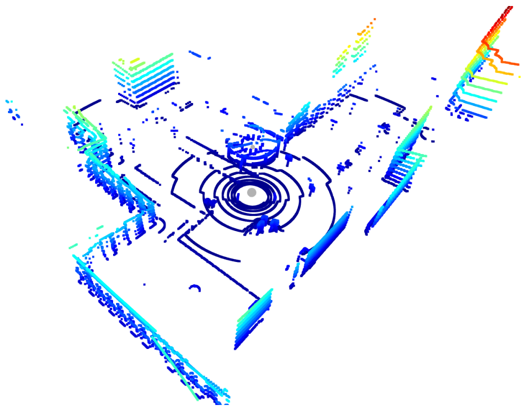


Figure 4.9: Full rotation of the simulated LiDAR 4.3.2.1 on the mesh shown in fig. 4.7

Velodyne VLP-16 / Puck			
Accuracy	3 cm	Vertical FOV	30°
Range	100 m	Vertical resolution	2°
Type	rotating	Horizontal FOV	360°
Rotation speed	5 Hz to 20 Hz	Horizontal resolution	0, 1° to 0, 4°

[8] and [18] both find that the accuarcy claims of 3 cm are mostly true for medium ranges, forming the typical normal distribution of the error. Simulation of the LiDAR is done by raycasting and intersection the CARLA Map 10 mesh. The raycasting is done by using the raycasting functionalities of the Open3D python library [53]. The Velodyne Puck hast 16 vertical channels (2° angular resolutin at 30° FOV). These 16 measurements are always taken at the same moment. The LiDAR is simulated by calculating the exact

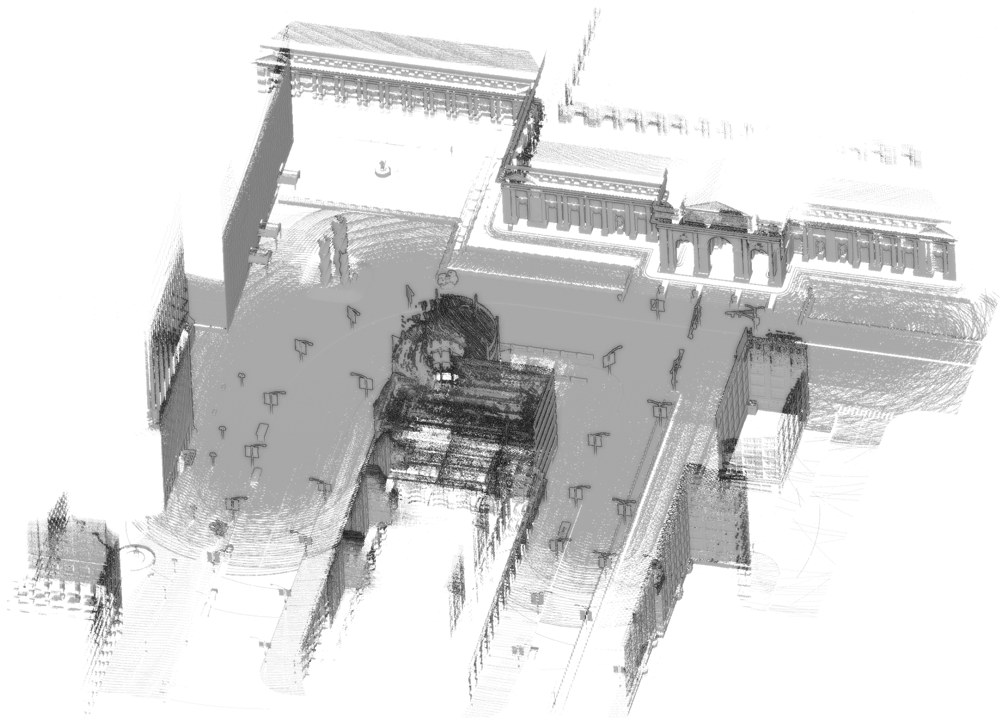


Figure 4.10: Point cloud simulated with a virtual LiDAR modeled after a Velodyne Puck (tab. 4.3.2.1) and the paths given in fig. 4.8

location for all these groups of 16 rays to be cast. The 360° degree FOV can not be simulated at once at a single point, because the car moves during the rotation of the LiDAR. One full rotation of the LiDAR is shown in fig. 4.9. From the given accuracy data of the VLP-16, it can be assumed that applying noise with a standard deviation of $\sigma = 0.027$ to the distance in meters gives a relatively good result for noise. In real world scenarios, the accuracy is often dependent on the distance, but this factor is ignored here. No atmospheric effects are simulated. Also, no effects of rotational error in the LiDAR are simulated. All points measured further than 100 m are dropped. This process results in the point cloud seen in fig. 4.10. All points are then combined into one large point cloud. Because the ray origin is always known, the points can be placed absolutely and not relative to the LiDAR, making a process such as point cloud registration to combine them superfluous.

4.3.2.2 Ground Truth

With the point cloud for testing generated, a crucial part is missing, the ground truth data. While the original mesh is a good starting point to generate these, it cannot be used directly to evaluate the quality of the results. Using the mesh directly would cause multiple problems. Firstly, not all surfaces present in the mesh have been in the FOV of the LiDAR, so no data about them was collected, making it impossible for the algorithm to recover them. Also, some of the surfaces that have been seen are very large and were not seen fully. Assuming full recovery for those would skew the results. The first problem can be tackled relatively easy. Open3D raycasting can return the primitive IDs of triangles of meshes when raycasting. These can then be used to filter the mesh in such a way, that only triangles remain that have been hit. However, the problem with large surfaces not fully covered remains. This effect can be seen when comparing the left side of fig. 4.11 with fig. 4.10. While some surfaces have barely any points, they would still be large areas to compare against in the ground truth data. To mitigate these overrepresented surfaces, a sphere is constructed arround every hitpoint on the primitives with the radius 0.25. The union of these spheres with the filtered mesh produces 4.12. The radius determined by try and error with the goal of producing a ground truth that has not many holes, but also does not overextend the ground truth from captured surface pieces. The surfaces generated by the algorithm will then be evaluated in two ways.

Distance calculations between all points of two different surface sets are done by determining the minimal euclidean distance from one to another. This relation is not symmetric, making it necessary to calculate it both ways. The distance between the generated surface results $d_{r \to gt}$ to the ground truth data. $d_{r \to gt}$ is a metric that measures how accurate the fitted surfaces are. Reversing this calculation to $d_{gt \to r}$ measures how much of the ground truth surface way actually recovered.

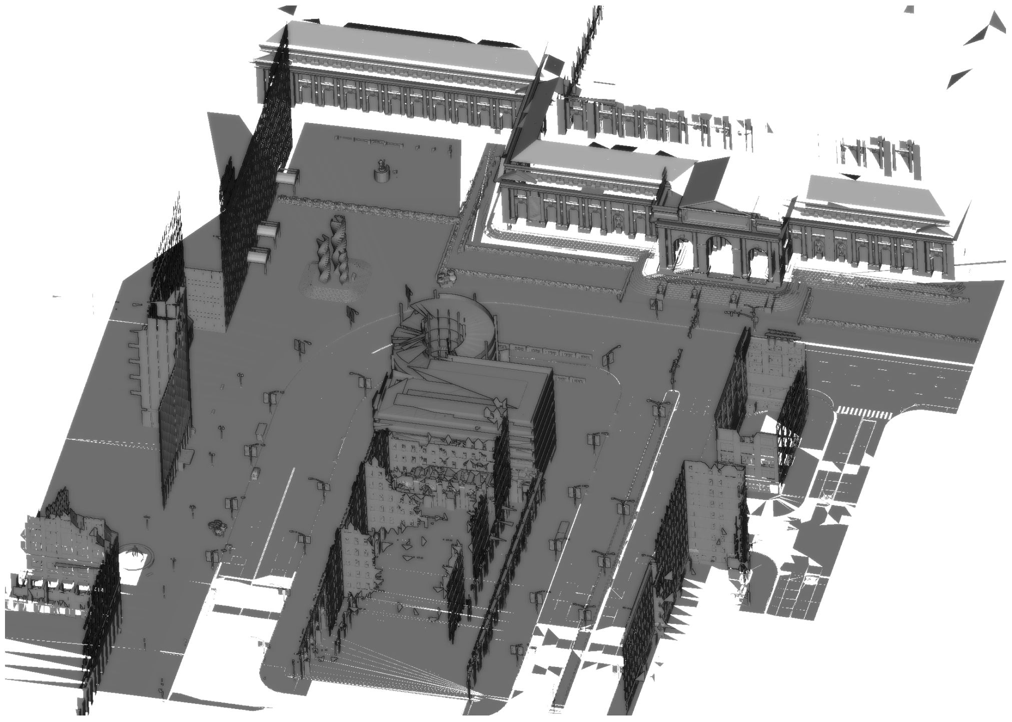


Figure 4.11: The red area from fig. 4.6 exported as a single mesh

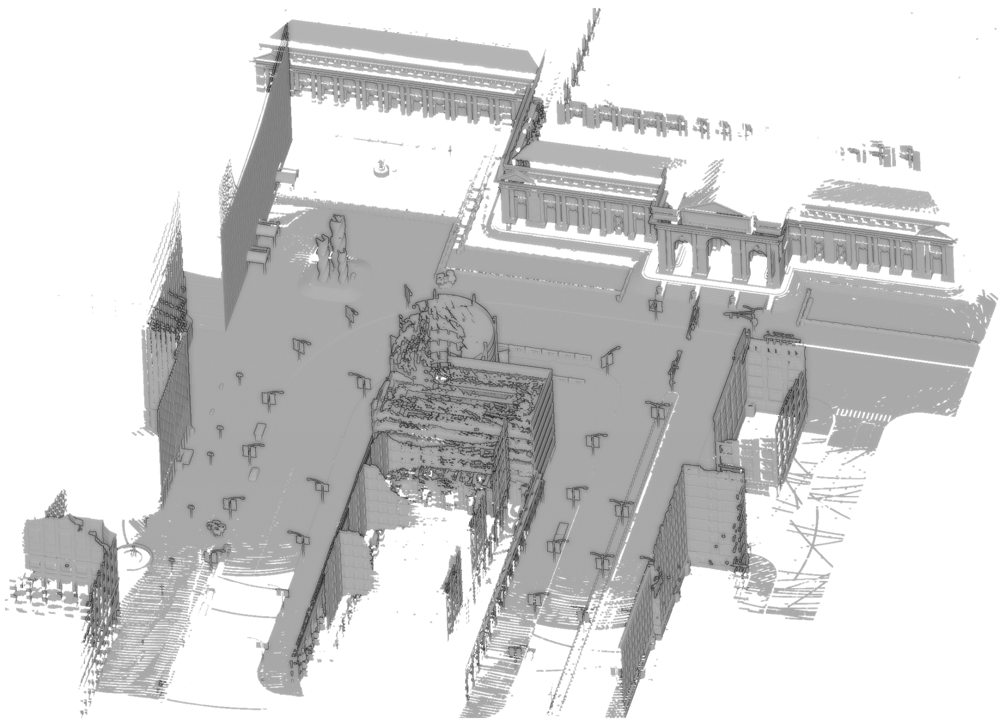


Figure 4.12: The filtered mesh transformed to ground truth data by removing all areas that were not sampled by the simulated LiDAR

5 Evaluation

The algorithm is evaluated on the HAW dataset and the synthetic datasets. For the synthetic datasets, the focus is on the accuary of the algorithm. For the real world dataset, the run time and algorithm steps will be evaluated. This will be followed by an analysis of the the distance metric used in the clustering step.

5.1 Synthetic Dataset

The ideal chunksize was determined using the data in fig. 5.1. The jump in the time data is somehow caused by the implementation. Around a million points per chunk were determined to be a good size because it keeps the time complexity low. Lower chunk size values resulted in too many seams in the data. To cut the point cloud into chunks of roughly these numbers of points, the chunksize was chosen to be squares with 35 m side length.

The parameters that worked best for the dataset were found by trial and error. The used values are: $t_{L_3} = 0.08$, $m_c = 200$, $m_s = 5$, $ng = 5$, $t_\Delta = 0.3$, $d_s = 0.4$, $S = 1$, $H = \{2.0, 4.0, 6.0\}$. The surface recovery created by the algorithm with this parameter set can be seen in fig. 5.2

The algorithm was able to recover most of the surfaces. When coloring the results by their surface patch, the chunking can be clearly seen in the recovered surfaces, producing small gaps.

The surfaces look well-formed. Especially in areas of the map that were more densely sampled by the LiDAR. In areas with such high sampling rates, even very small surfaces were recovered. Here, the limiting factor is the radius of the PCA calculation. This can be seen on the curbs of the streets, which are missing from the result data. For very thin surfaces, if the normals were rounded too much with the neighboring surfaces, the thin

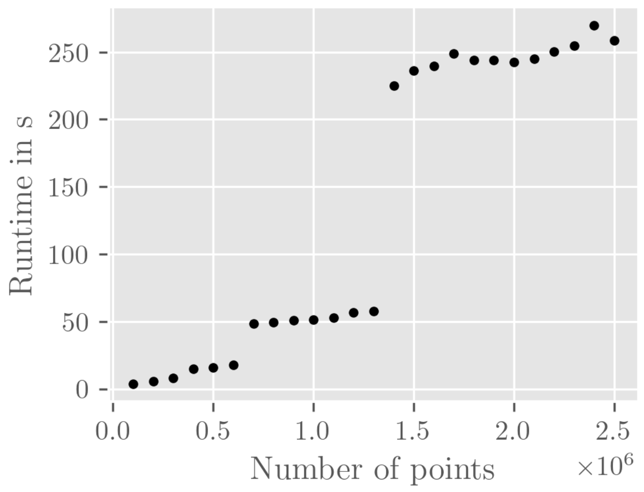


Figure 5.1: Runtimes of HDBSCAN, generated on HAW Campus dataset by adding more and more points to the clustering beginning at its geometric center.

surface was not recovered. For less dense areas, the scan pattern of the LiDAR can be seen in the results. Especially thin rows of points tend to be not connected with the next row.

On the rounded building in the middle and the historic building, it can be seen that the algorithm is able to recover non-planar surfaces. The pole like structures of streetlamps were flattened, but this was a tradeoff in the algorithm design.

The ballpoint pivoting algorithm produces sensible outlines. The outlines produced for the surfaces are not very smooth. Outlines in low density areas sometimes look a little frayed. Buildings with same sized regular patterns in their facade tend to produce many small surfaces instead of one large one. Some areas with very intricate geometry tend to be smoothed over by the spline fitting, see the facade over the pillars of the historic building and compare with fig. 4.8.

5.1.1 Accuracy

The accuary is evaluated in two steps. Firstly, how accurate is the fit of the surfaces to the ground truth data, and secondly, how much of the ground truth data is recovered.

Fig. 5.3 shows the accuracy of the fitted surfaces. Generally speaking, the sufaces adhere very well to the ground truth. The fit is better in areas with higher scan density. Lower

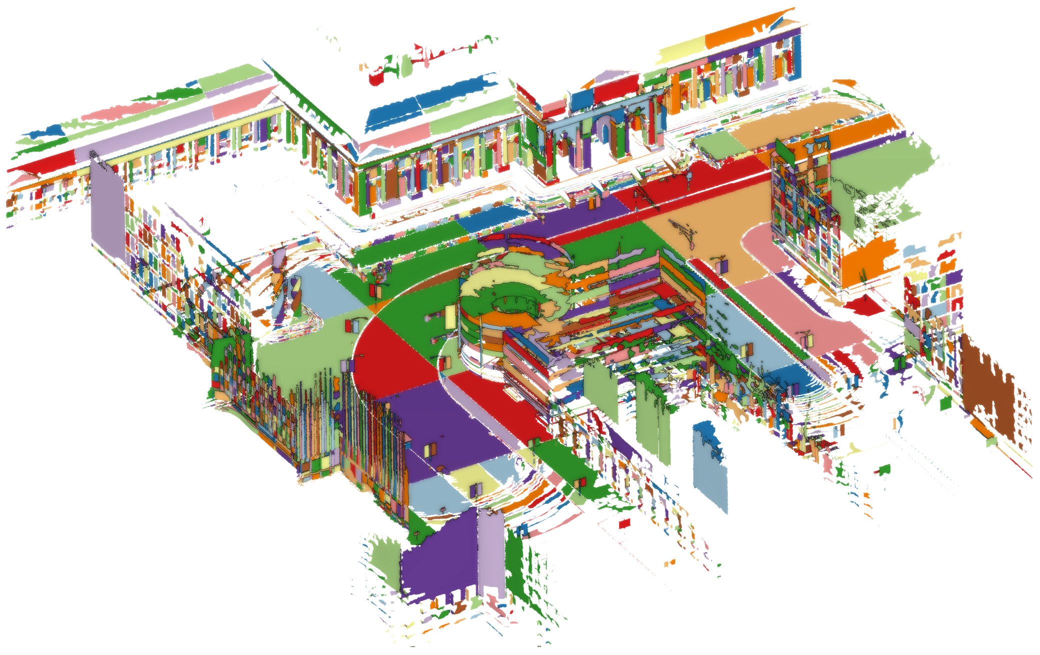


Figure 5.2: Recovered surfaces. Every patch in a different color.

density areas are recovered less faithfully. There are not many surfaces present that were not there in the original data. This can happen if points get clustered together that do not belong together. For curved surfaces, there are sometimes artifacts from the ground truth visible. Since the ground truth was defined in a mesh, the surfaces are not truly C_1 continuous. The edges of mesh triangles can sometimes be seen in the data, for example the ramp of the parking garage. High detailed areas are generally recovered less faithfully, which is caused by the oversmoothing effect as seen on the roof of the historic building.

To measure if a surface is at least partially recovered from the ground truth, every point of the ground truth is considered recovered, if it lies within a distance of 0.25 of the computed surfaces after $d_{gt \to r}$ computation. This distance was chosen to be equal to the radius used in the ground truth generation step. The total area recovered is 74.5% of the ground truth data, as seen in fig. 5.4. While high density sampled areas are mostly all present, the lower density areas are generally not recovered. While the over smoothed surfaces mostly lie within the 0.25 distance boundary, some areas are so highly simplified that they are above this threshold.

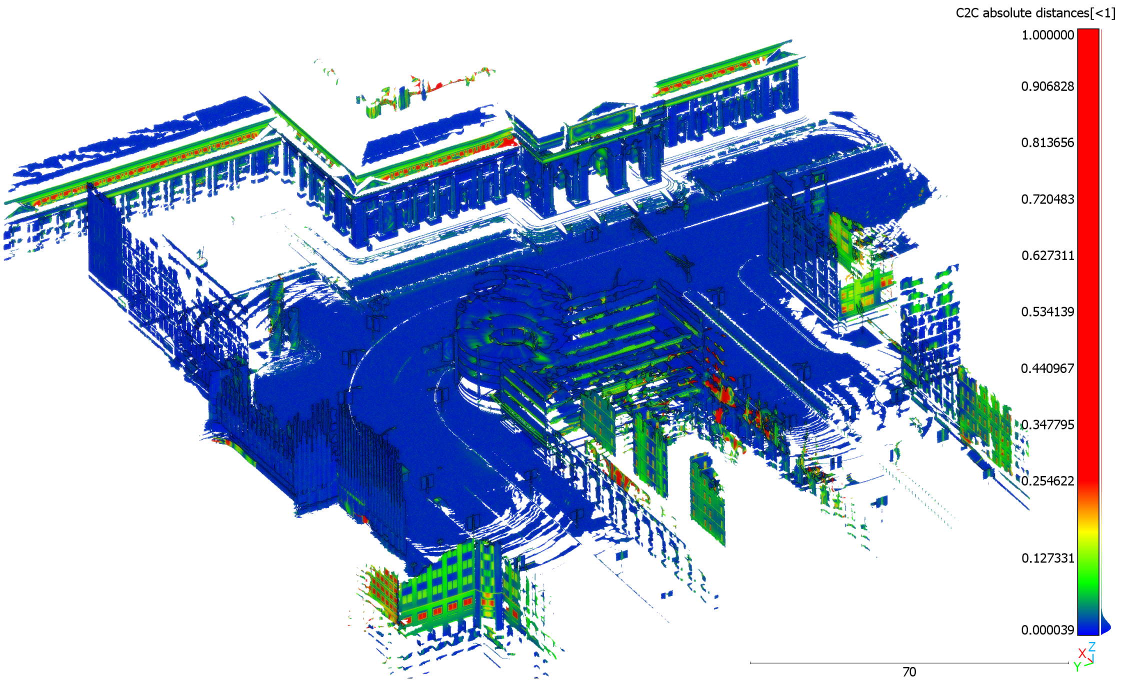


Figure 5.3: Accuracy of recovered surfaces.$d_{r \to gt}$: Distance between the recovered surfaces and the ground truth

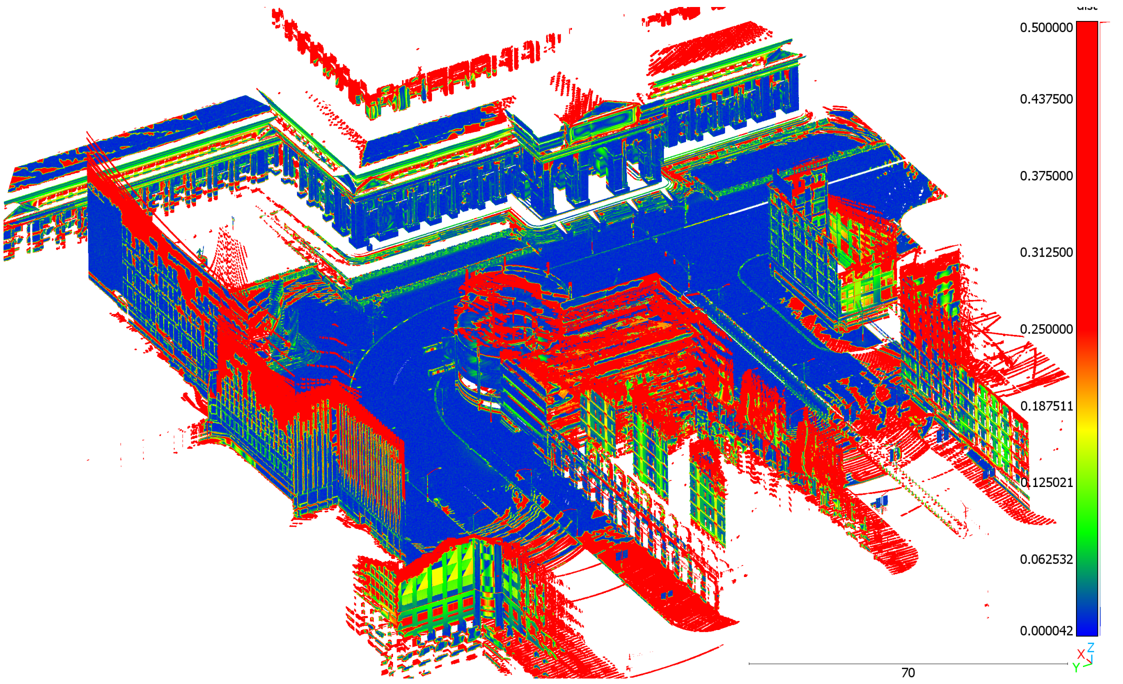


Figure 5.4: Recovered surface area. $d_{gt \to r}$: 74.5% of the ground truth area has been recovered.

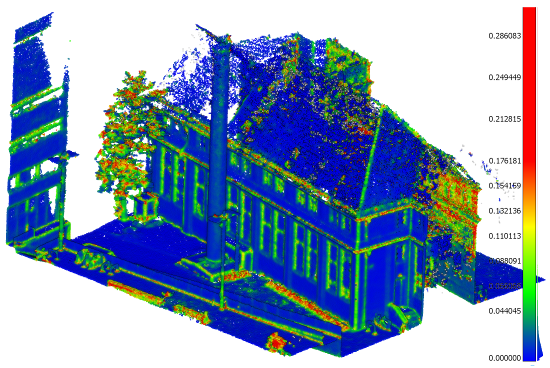
5.2 Real World Dataset

The parameters chosen by trial and error in the same way as for the synthetic dataset are, $r = 0.3$, $t_{L_3} = 0.20$, $m_c = 200$, $m_s = 5$, $ng = 4$, $t_\Delta = 0.05$, $d_s = 0.3$, $S = 1$, $H = \{5.0, 6.0\}$. While the point cloud has a size of 609, 7 MB, the spline parameters, outlines and PCA components of the same point cloud can be stored in 1, 7 MB.

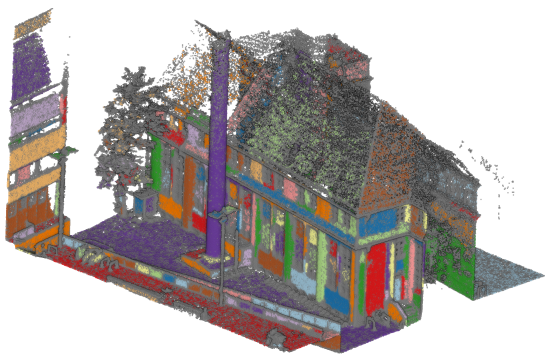
To better visualize the algorithm steps, the chunk x4y3 was choosen. Compare fig. 5.6 and fig. 5.5.

The L_3 values are higher in the areas that are not surface like. The tree on the left exhibits such high L_3 values. Corners and edges are also clearly visible. The clustering step produces relatively many noise points. Especially the roof has not many points in the clusters. Yet, a lot of the roof area has at least one point in the clusters, showing HDBSCANs capability to cluster data with different densities. No clusters can be seen that wrap around corners. The facade area has a lot of small clusters. Expanding the clusters using the fitted spline reduces the number of noise points greatly. The produced surfaces after expansion sometimes still have gaps between them. Sometimes, the expansion step overlaps the clusters. There are also clusters of non-surface like structures, like the branches of the tree.

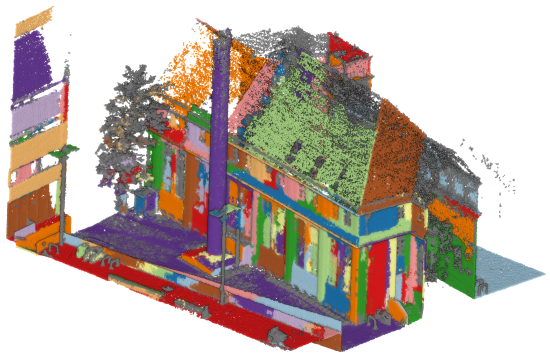
The resulting surfaces demonstrate a high degree of fidelity to the initial point cloud data. Dense and less dense clusters have been transformed into spline surfaces. The surfaces reach close to the edges and corners. Some surfaces overlap and edges of the surfaces are frayed. Furthermore, even very small surfaces are also partially recovered. The pillar is missing from the final result. This is due to an oversight in the algorithm design. The pillar is round, meaning it has normals point in every direction of the unit circle. This implies, that it cannot be transformed by PCA in any way to satisfy the spline fitting criteria of the spline fitting step. In itself, this is still uncritical, as it would simply be flattened by the spline fitting and result in a bad fit. This is why the expansion step seems to work, the pillar is simply a rounded structure close to a half circle in spline representation. The real problem arises in combination with the outline detection. The outline detection produces outlines on the bottom and top of the pillar, bridging the half circle fit of the pillar cluster. With outline and surface geometry not really fitting together, no sensible result can be computed.



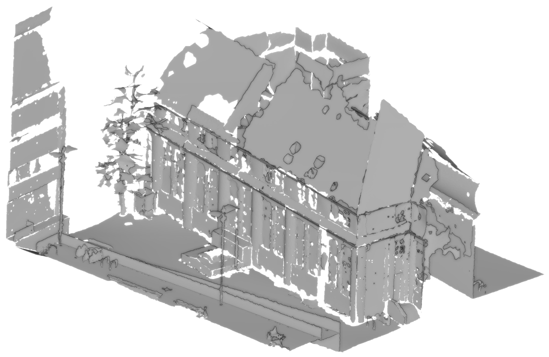
(a) L_3 value



(b) Clustering step



(c) Clusters after expanding the fitted splines



(d) Resulting surfaces

Figure 5.5: Algorithm steps on the chunk x4y3 of the HAW dataset.

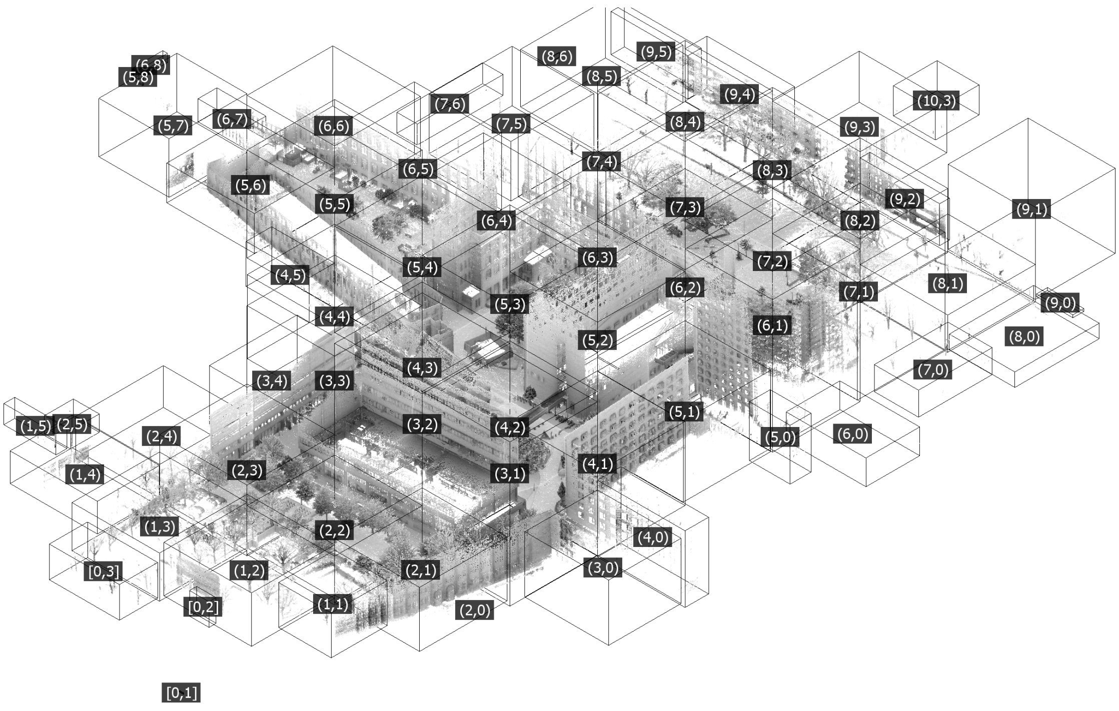


Figure 5.6: The HAW Dataset split into multiple cells

5.3 Distance Metric

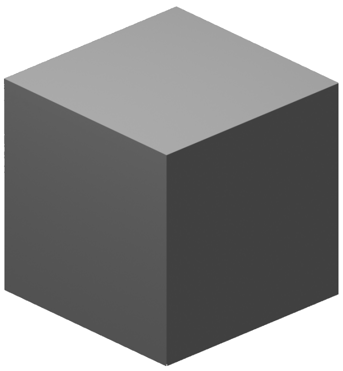
While it is shown in [5] that the oriented point distance can be used to segment points clusters with aligned normals, it is not shown how the metric influences the HDBSCAN algorithm compared to a euclidean metric. To verify that the oriented point distance has the desired effect of separating clusters with differently oriented normals, a minimum working example is constructed. The constructed example consists simply of a half cube (three sides of a cube, see fig. 5.7). This mesh is then sampled relatively sparsely (120 points) into a point cloud. The sparse sampling eases the evaluation of the results by keeping the graphs representing the inner HDBSCAN works uncluttered and clear. For the points in this point cloud, perfect normals are assumed. A three-dimensional and a projected two-dimensional minimal spanning tree for the first two HDBSCAN steps of calculating the mutual reachability distance and constructing the minimal spanning tree using the euclidean distance, which is equivalent to $ng = 0$ in the oriented point distance, is given in fig. 5.7. The graphs show, that the d_{mreach} distances are roughly equal, with points lying on the boundary edges of the half cube having slightly larger d_{mreach} values with their neighbors. This is to be expected, since density estimate of

HDBSCAN assumes a circular neighborhood, which for a point, for example, directly lying on an edge, means it would be half empty, increasing the distance to its m_s kNN. Since the core distance d_{core} is directly linked to that, this increases their d_{mreach} value to their neighbors by increasing their core distances. While points with the same normals are grouped together, because they are close to each other, no clear separation is seen between the groups. With no separation in the minimal spanning tree, there are no clusters to be found. This can be observed in fig. 5.9. The condensed plot for $ng = 0$ shows one big stable cluster, meaning no clustering of the data with HDSBCAN is possible.

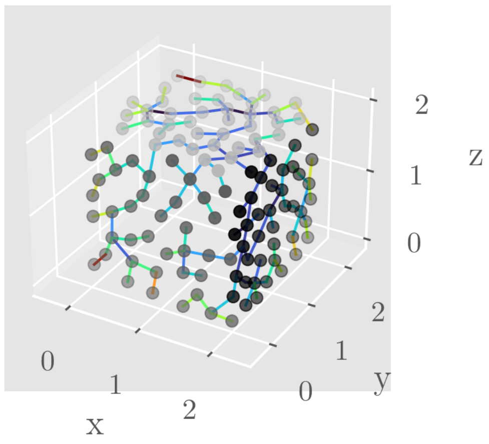
Increasing ng to 1 produces are more clear separation in the minimun spanning tree (Fig. 5.8) and larger d_{mreach} values between points originating from different sides of the half cube. The segmentation is even completely correct for this small example with noise free normals. The completely correct segmentation shows itself in the condensed graph of this clustering. Fig. 5.9 with $ng = 1$ shows three clusters that are more stable than the previous one stemming from origin. The origin cluster also looses no points, meaning the three clusters originating from it must contain all points and no points are marked as noise.

This noise free clustering can not be seen in the real experimental data. In the experimental data, HDBSCAN does produce significant noise points extending clusters to the edges of the clustered continuous surfaces represented by the point cloud points. This behavior is consistent even with low values of m_s. While some of this noise could be explained by the ways HDBSCAN estimates the density of points, the oriented point distance might also play a role. To check if the oriented point distance has an influence on this behavior, the half cube is sampled mode densely (1200 points) into a point cloud P_{hc}. The edges of the half cube are formed by 4 points P_0, P_1, P_2, P_3. Let the set of these lines be $L = \{(P_0, P_1), (P_1, P_2), (P_1, P3)\}$. Now, for every point p in P_{hc} the euclidean distance to the nearest line in L is calculated. The normals in t are calculated using PCA to get the normal smoothing effect present in the real experimental data. Both calculations are presented in 5.10.

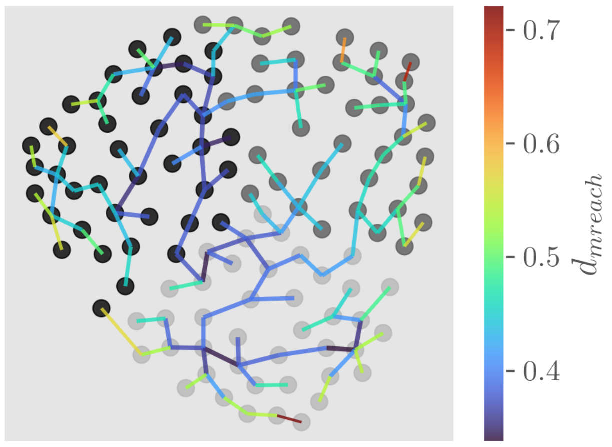
Now, the core distance for every point is calculated with $m_s = 10$. The m_s value is an educated guess based on the previous experiences and should not over or under pronounce any effects on the core distance. When increasing ng, the core distances rise more steeply the closer the point is to L. The core distances of points lying further away then r to L do not change at all. When coming into r range of L, the core distances start to rise smoothly, then the rise gets more linear, see fig. 5.11. The slow takeoff can be explained



(a) Half a cube. Only the three sides visible are present. Sidelength = 2.



(b) The sampled cube, with MST of the mutual reachability distance as generated for HDBSCAN.



(c) The MST of (b), TSNE projected onto a plane.

Figure 5.7: Half a Cube, with 120 random samples taken. Sample points in the same color as cube sides. MST generated with $ng = 0$, $m_s = m_c = 5$. The points are colored in the same color as the side of the half cube they were sampled from.

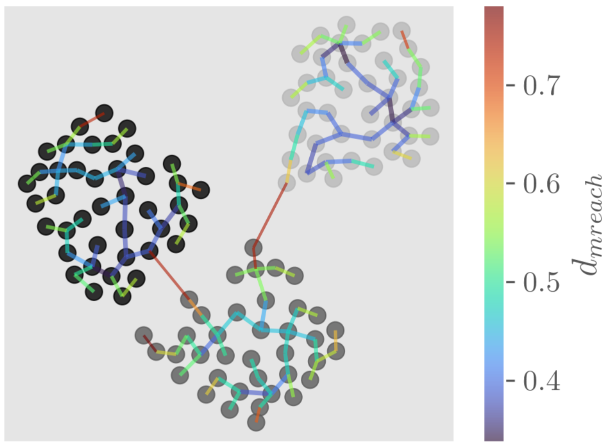


Figure 5.8: Effect of increasing ng to 1 on the MST generated by HDBSCAN using the same data as 5.7.

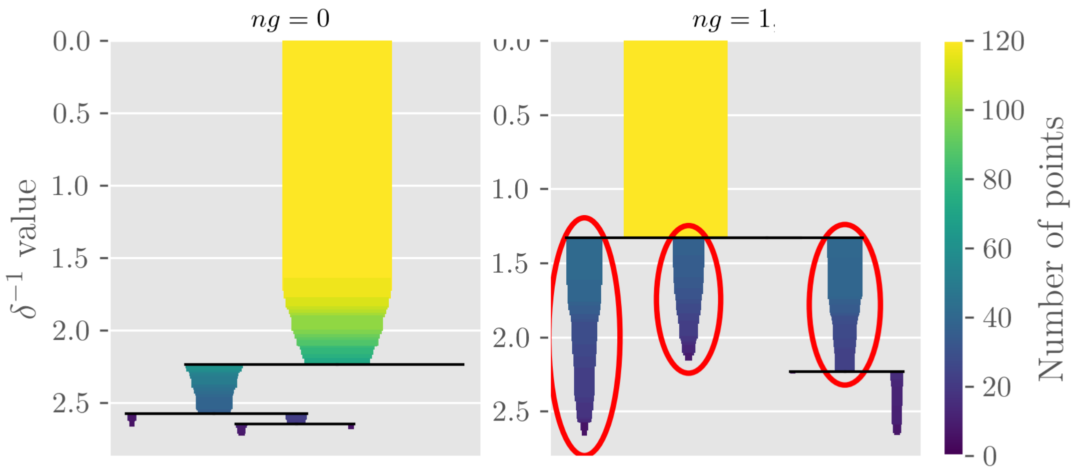


Figure 5.9: Effect of ng on the condensed tree of HDBSCAN and the cluster selection. $ng = 1$ shows the correct cluster selection.

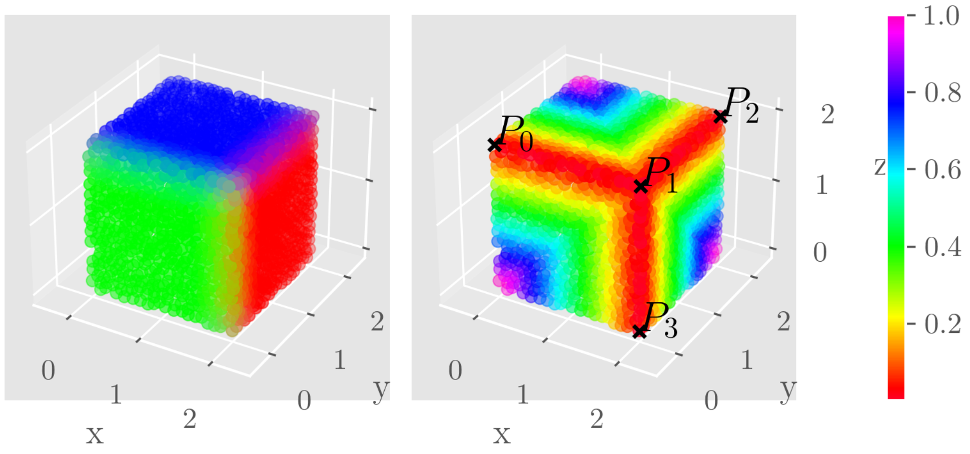


Figure 5.10: Cube from 5.7 sampled with 1200 points. On the right, normals calculated using PCA with $r = 0.3$ and their x, y and z values mapped to the r, g and b color channel. On the right, the euclidean distance of each point to the nearest edge of the half cube.

by the normal smoothing effect. With point further away form L having no erroneous smoothing in their normals, their d_{mcore} value is not affected by ng, since all the normals are the same, so they add no distance in the euclidean space used by the oriented normal distance extimation Δ_e. When the PCA normal estimation was at minimum $2r$ away from L, the smoothing starts. This pushes apart the point in euclidean space, with higher ng values amplifying the effect. Large r values smooth out more of the normals, making the difference between neighboring point normals smaller, but also spreading the effect further into the surface. The effect gets much stronger around the r distance value, since the influence of points not coplanar to the point the normal is calculated for gets more and more dominant. The linear increase can be explained with the Δ_e distance, since it is a linear approximation. The spread is explained by inconsistencies in the sampling density, because random sampling does not produce a perfectly equal density of points. The L_3 filtering can hide parts of this effect.

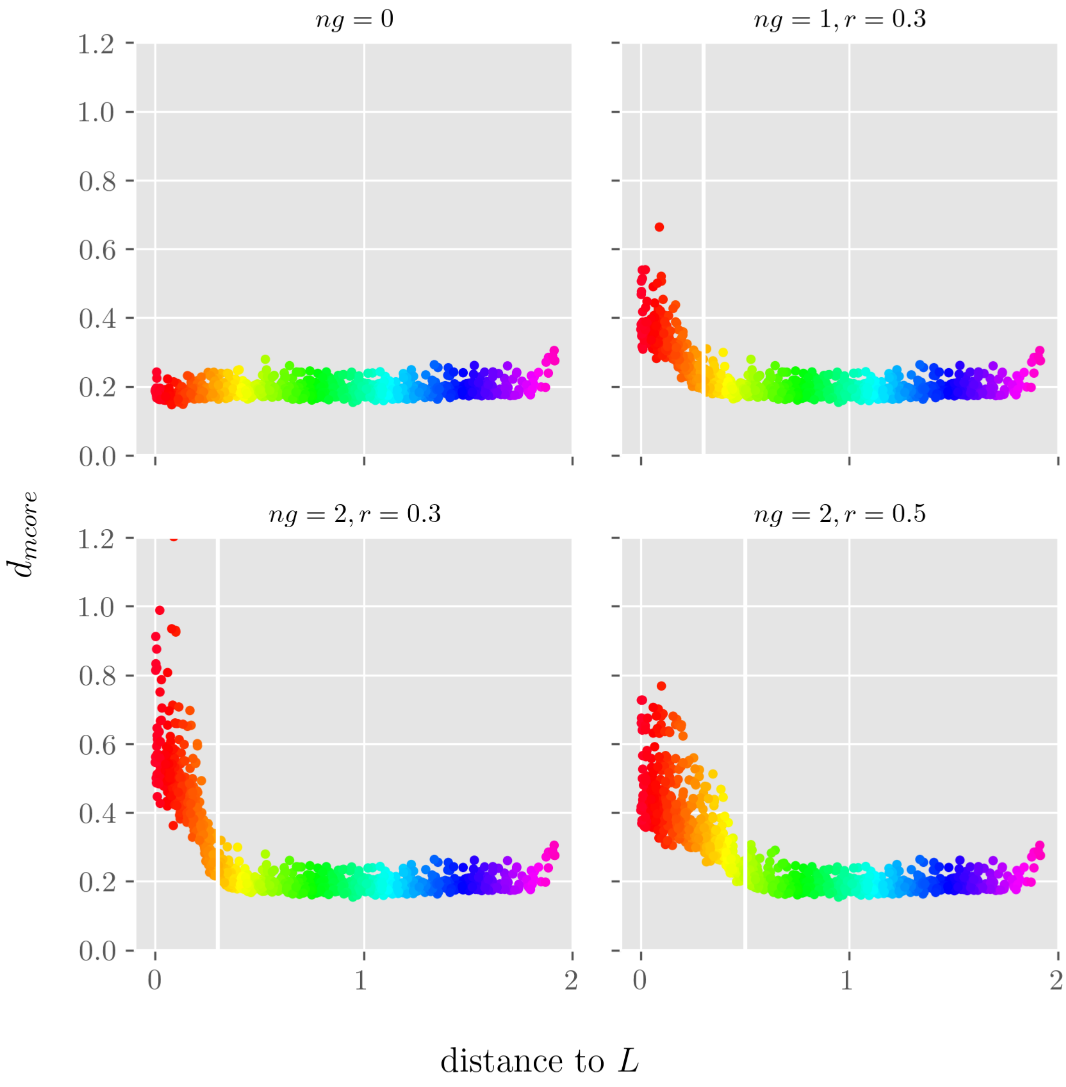
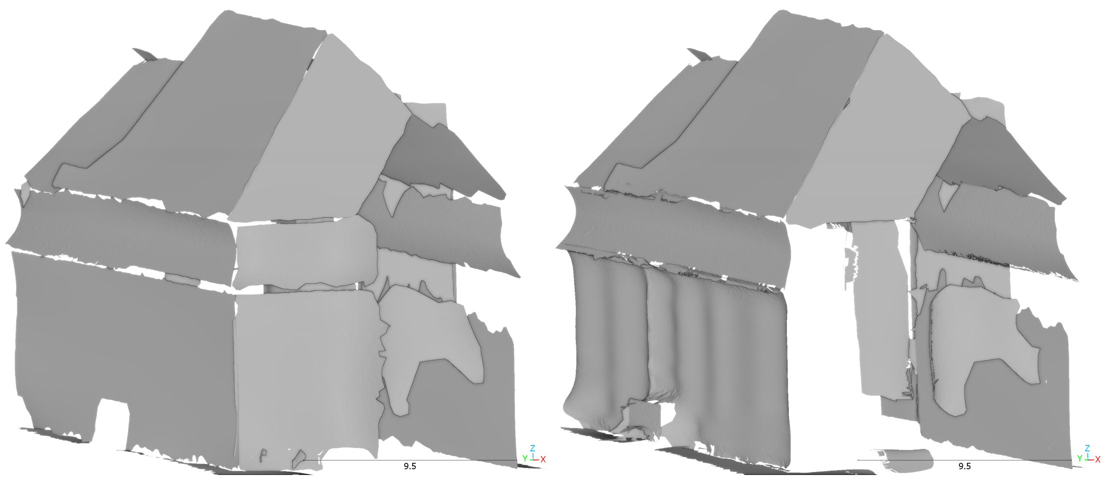


Figure 5.11: Same colorscheme giving the point distances to L as in fig. 5.10. The effect of Δ_e onto d_{mcore} with different ng and PCA normal calculation r values.

5.4 Parameters



(a) $t_{\Delta} = 0.05$, $d_s = 0.3$, $S = 1$ (b) $t_{\Delta} = 0.2$, $d_s = 0.5$, $S = 0.01$

Figure 5.12: Effects of changing certain Parameters. Chunk x4y3 of the HAW dataset. Shared between both: $r = 0.4$, $t_{L_3} = 0.05$, $m_c = 1000$, $m_s = 10$, $ng = 4$

The algorithm has many parameters that can be tuned. The tuning of HDBSCAN parameters is large explained in [3]. Larger m_c values lead to larger clusters, therefore less fine segmentation of surfaces, with m_s controlling the amount of noise point filtering. ng, also explained in [3], also influences how strongly surfaces be segmented. The r parameter of the PCA step controls how smooth the point normals and the L_3 values are. Most of the time, when a cruder segmentation is wanted, increasing the m_s values together with the r value will lead to the desired result. Such a larger segmentation can be seen in fig. 5.12. When compared to fig. 5.5 the resulting surfaces are much larger. An increase in r is often needed since smoothed out normals tend to point in the same direction even in the presents of smaller variations in the surface. This makes them less spread out in larger connected surfaces supporting the clustering step.

The effects of the spline smoothing value S can be seen in fig. 5.12. While the wall appears almost flat with a low S, the bulges in the wall are preserved with a lower smoothing value. The missing walls were multiple surfaces clustered together erroneously. With a high smoothing value, the edge can be smoothed out. With a lower smoothing value, these clusters can no longer be fitted with a spline, making them disappear from the result.

Also seen in fig. 5.12 are the effects of increasing t_Δ and d_s. Increasing t_Δ with d_s can lead to more recovery of points from the noise points produced in clustering and L_3 filtering. However, increasing t_Δ too much worsens the overlapping issue, clearly seen by the overlapping of the walls. Increasing d_s on its own lets smaller surface deviation be flattened instead of forming holes, but it can lead the creation of surfaces not present in the data by adding points to the spline that do not belong to the surface represented by it.

5.5 Runtimes

The runtime analysis was performed on the HAW dataset with the same parameters as the dataset evaluation. The experiments have been performed on a systen with a 20 core Intel ©Core™ i7-12800H processor and 64 GB of RAM and no GPU acceleration.

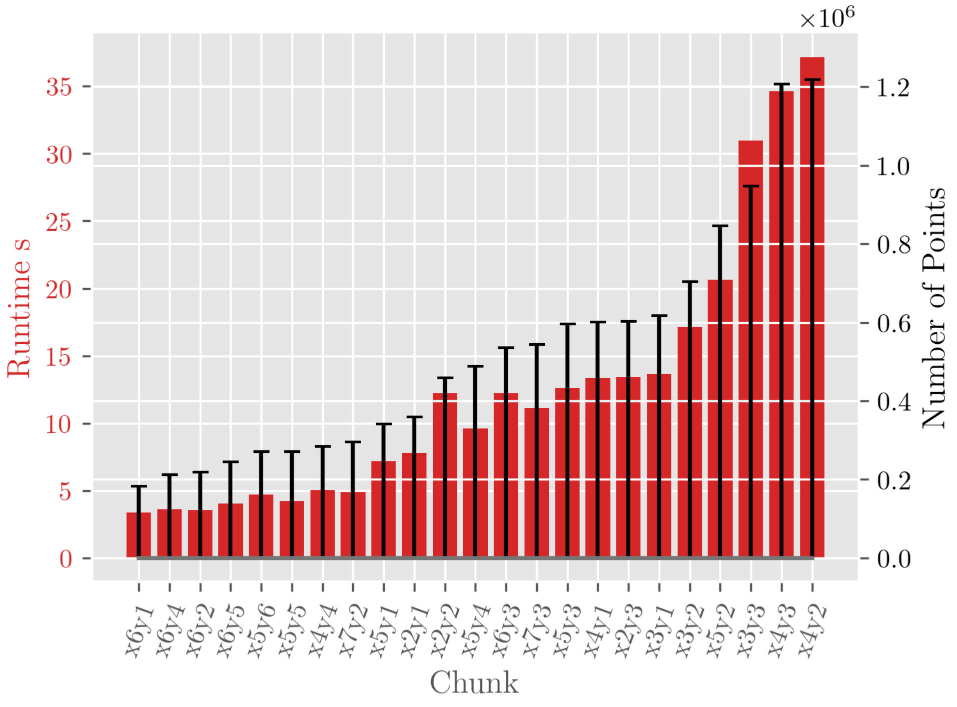


Figure 5.13: HDBSCAN runtimes. Only chunks with a runtime above 5 s for readability. Chunks as in fig. 5.6.

The runtimes of every step of the algorithm have been captured in detail, except for the PCA and L_3 filtering. The PCA step took around $1,3\,\text{min}$, with larger values for r increasing the runtimes, with the L_3 filtering being nearly instant.

The runtime for the clustering step was captured per chunk, see fig. 5.13. As it can be seen, that the denser and central chunks, compared with fig. 5.6, have longer runtimes, which is to be expected given the complexity of HDBSCAN. Also, visible is the distribution of point cloud sizes within the chunks, with many chunks not containing many points. Because the used HDBSCAN library already performs its calculation in parallel and the chunks are clustered one after another, resulting in a combined runtime of $302,45\,\text{s}$, or roughly $5\,\text{min}$.

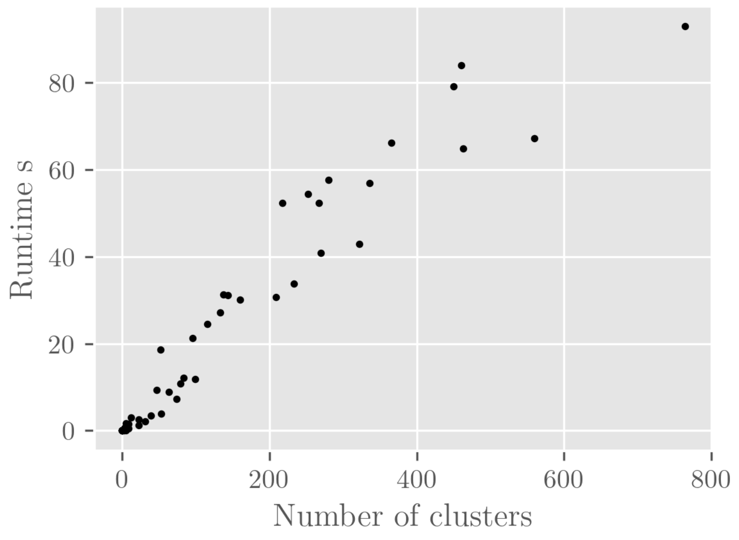


Figure 5.14: Spline fitting runtime, per chunk.

Spline fitting was also measured per chunk, since clusters in one chunk are always fitted with a spline, independent of the other chunks and within a single thread. The spline fitting and surface expansion were measured together. Fig. 5.14 shows a roughly linear time complexity, which is expected from the hashmap used in the expansion step and the spline evaluation step. The graph also shows, that the runtime is largely dependent on the number of clusters within a chunk. The overall runtime was $60,01$ seconds, with chunk wise parallel execution on 20 cores.

The ball pivoting step for the outline calculation has been measured per surface patch, because these calculations are indipendent from another and were executed using a thread-

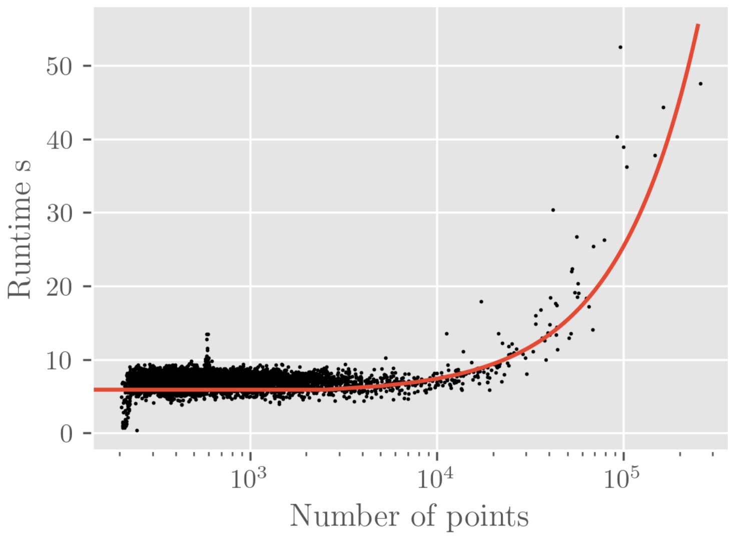


Figure 5.15: Outline calculation with ball pivoting. Per surface patch, not per chunk, with a linear function fitted to the data (red).

pool. Fig. 5.15 shows a linear time complexity rising with the size of the surface patch. The measurements are dominated by the ball pivoting step, since it deals with many more points than the outline construction on average. Overall calculation time was 2536, 4 seconds, or 42 min.

6 Discussion

The results of the algorithm are satisfactory. They fit the desired properties of accuracy and size needed for the geometric layer of an HD Map. The approach is much more general and makes fewer assumptions about the data than the methods presented in 3.3. The best surface reconstructions are given in areas with low curvature, large surfaces and high point density. This is to be expected. Here the accuracy to the ground truth data can be in the sub centimeter range. This indicates, that the spline fitting is able to remove noise from the data. The accuracy is reduced with lower density areas, but still lies mostly within an acceptable range of a few centimeters. Generally, if a surface is recovered, it is mostly accurate. The only surfaces not well recovered are high details areas. These areas tend to be smoothed over. The combination of PCA, which tends to smooth over the normals of such small details, and HDBSCAN clustering these areas as one, leads to the spline fitting smoothing the geometry in such a way, that it is not true to the original surface. Decreasing the PCA r radius might reduce the smoothing effect, but will also make the normals more noisy, which makes it harder for HDBSCAN to segment the cloud correctly. For HD Map applications, omitting smaller surfaces can be acceptable, as long as they do not produce holes in the data, which is not the case here. In lower density areas, the scan pattern of the LiDAR can sometimes be seen in the final result. Another concern is the generation of erroneous surfaces, that are artifacts of the algorithm and are not, at least partially, in the original data. Here, the algorithm performs well, not creating many of these surfaces, and if, then just in low density areas.

The total surface recovered is with 74.5% satisfactory. One might argue that nearly 25% missing from the data is quite much, but this is also due to the used evaluation method of assuming that around very low density points the same area could be recover as from high density points. The algorithms shows similar behavior as in the accuracy evaluation. While the recovery in high density areas is very good. The argument can be made, that recovering such areas is not as desirable. Recovering such areas would need a great amount of inter- and extrapolation. This can often only be done when assuming a certain data structure, making the recovered surfaces prone to errors introduced by these assumptions. Nonetheless, it shows that the flexibility of HDBSCAN has its limits dealing with different density clusters, at least with the oriented point distance. This might also be due to the fixed PCA radius. With a low number of points, the normals and L_3 values become very noise.

The size reduction also meets the HD Map criteria. With a reduction to a few megabytes, which is a reduction of 99.8 percent for the HAW Dataset, the reduction is significant. The main reduction can be found in large surfaces with low or no curvature. They contain many thousand points, with the largest ones up to 300.000 in this example, but can be represented using a spline and an outline with not that many points. While other methods also probably achieve similar results, not data was given in recent literature.

The chunking greatly reduces the algorithm runtimes. Since the chunking is only really needed in the clustering step, maybe a merging of adjacent clusters over the boundaries of chunks would be beneficial. The distribution of chunk sizes shows that many chunks contain not a lot of points. This is a result of MLS point clouds being less dense on the edges of the point cloud. Since larger chunks are better, to a certain extent, merging small chunks before clustering might be a solution. While the seams caused by the algorithm are not very problematic in the data, they are nevertheless recognizable. However, when looking at the algorithm accuracy evaluation, the chunk seams tend to be invisible, indicating they are not a limiting factor of overall result quality. From this follows that the chunking approach has more benefits than drawbacks.

The L_3 filtering generally works with C_1 discontinuities, but it does not perform well when it comes to filtering out non-surface like objects like vegetation. On the real world data, parts of trees are still present in the results. This could be solved by increasing the r PCA radius, but this would deteriorate small detail in surfaces even further. In the literature previously reviewed, more sophisticated filtering and classification steps exist.

Using HDBSCAN to separate a point cloud into at least C_1 continuous surfaces works well in practice. HDBSCAN in combination with the oriented point distance metric segments well and can handle the different densities present in a MLS recorded point cloud. A big advantage over many other techniques is, that the segmentation is agnostic to the underlying true surface. While many techniques can only segment planes or certain geometric shapes like sphere or cylinders, the HDBSCAN segmentation can segment all surface types as long as they have smooth normals with moderate curvatures. However, the number of noise points produced by the clustering step is quite high. In areas with a lot of detail, the algorithm sometimes suffers from oversegmentation. Noise and a lot of high curvature geometry produces a lot of island clusters in combination with the effect of the distance metric on points close to edges. Also, the clustering does not guarantee that surfaces are always segmented. Clusters that already disconnected from the main point cloud by a large gap are sometimes not further divided into surfaces. It is suspected, that the reason for this is, that the large gap causes a relatively long edge in the HDBSCAN minimum spanning tree, which results in a stable edge in the collapsed single linkage tree, creating a stable cluster that is selected. Maybe, the cluster selection criteria could be improved for this special case or doing a connected component analysis beforehand.

Using splines as a general surface representation for smooth surfaces works well. While spline fitting was not the main focus of this work, as other curved surfaces representations might have also worked, it is still shown that spline interpolation and extrapolation work well on MLS point cloud that was segmented. The spline fitting produces accurate results. Expanding the found clusters using spline extrapolation largely mitigates the effects of the oriented point distance metric and HDBSCAN noise points be recovering them fairly accurate. Problematic can be the overlapping of adjacent expanded surface patches. While this is not visible in the accuracy data, since a duplicated surface is not detrimental to the minimal euclidean distance, it is still not a good surface representation. The overlapping is caused by simply checking all points within a distance in a cluster and not growing the regions from the edges, like [24] and [25] do it. Edges and corners are only approximately recovered, since the expansion does only expend to point already present in the LiDAR point cloud. Doing it this way, the inherent undersampling problem around C_1 discontinuities is not solved. However, a very big advantage of doing it this way is the runtime. While exact approaches have very long runtimes [37], while still being limited to planes, or make assumptions about the shape of the data to ease the calculation [25], the expansion used here is general and fast.

The outlines produced by the ball pivoting are generally sensible. The approach to use a multiple of a density estimate to treat different clusters with different point densities seems to have worked for the most part, since less dense clusters also produced good outlines. Limitations arise in areas with very low sampling density. Here, the outlines tend to be more frayed. Also, clusters with a high range of densities are sometimes not treated as well. The produced outlines tend to underrepresent the surfaces, since the points connected by the ball pivoting generally do not lie directly on the edge of the true underlying surface. In areas with low density, the ball pivoting edges get more frayed.

Parameters tuning of the algorithm allows to produce a wide range of different results tuned to the application and desired fidelity of the result. Small surfaces with lots of detail can be produced, or larger surfaces with more smoothed out details. However, the amount of parameters that all influence each other can make the algorithm hard to tune to the desired result. Here the chunking is also handy, since tuning on one chunk is less computationally intense than tuning on the entire datasets.

Overall runtimes of the algorithm meet the criteria for large MLS point clouds. The chunking makes it scale to basically any size because the chunks can be treated one after another. Also, the algorithm parallelizes well and can be implemented in such a way with relative ease. The part of the algorithm that has the highest theoretical computational complexity is HDBSCAN. But since this is limited by choosing an adequate chunk size, the HDBSCAN segmentation step is not the limiting factor when it comes to runtimes. The complexity of the spline fitting has not been analyzed, but looking at the measured runtimes, it is not dominant. The spline expansion is also reasonably fast by using kd-trees and hashmaps to find fitting splines. The performance bottleneck is the outline detection step, more precisely, the ball pivoting. The large number of points and surface patches make it relatively slow. Using a two-dimensional version in spline space might make this step much faster. The algorithm could also be reimplemented in a programming language compiling to a native binary format to increase speed.

A limitation of the algorithm are surfaces that are smooth, but cannot simply mapped to a euclidean plane in a way that the spline fitting criteria are fulfilled. For example, when the surface normals form are spread apart further than 180°, or form a continuous upward spiral. Thankfully, such surfaces are relatively rare in urban environments.

6.1 Datasets

The created dataset is of high value for evaluating MLS based surface recovery. No other datasets offering a ground truth of this type was found in literature, at least for this work. The used euclidean distance based evaluation might be improved upon, it is not very fair when it comes to recovery in low density areas and around the open edges of a MLS point cloud.

The resulting surfaces in the synthetic dataset are different from the ones recovered from the HAW dataset in their quality, with the synthetic dataset generally producing the more desirable result. This indicates that the synthetic dataset suffers from a few shortcomings. The Gaussian error term might have been too low. Also, maybe outliers should have been added to the data. Realistic results were also hindered by not using a very sophisticated error model for the simulated LiDAR, more advanced ones exist [33]. Another problem might be the mesh, that the point cloud and ground truth data are based on. This mesh contains perfectly curved, within the limits of a mesh representation, and planar surfaces without added high details features such as for example bricks, which are present in real world datasets. Real world datasets tend to contain more clutter like plants and non-building object such as bikes, cars etc, making the separation of surface harder.

However, it can be seen that the algorithm still performed mostly comparable with roughly the same parameters, meaning the synthetic datasets, while it has shortcomings, is still a good basis for accuracy and surface area recovery evaluation.

7 Conclusion and Future Work

In summary, the algorithm presents a promising solution for generating high-definition map geometry from point cloud data. It demonstrates accurate results that align with HD Map requirements for accuracy and size. The approach's flexibility and reduced assumptions compared to existing methods are advantageous. High-density areas showcase accurate results, while lower-density regions still remain acceptable. The algorithm efficiently handles large point clouds through chunking and parallelization, with acceptable runtimes. The use of HDBSCAN for segmentation and spline fitting for smooth surfaces proves effective. Despite challenges with high-detail areas and low density parts of the point cloud, the algorithms overall performance is noteworthy for HD Map applications. Further improvements in filtering and outline detection could enhance its capabilities. In conclusion, the algorithm offers a strong candidate for advancing HD Mapping technologies.

While the presented algorithm exhibits promising outcomes in generating high-definition map geometry from point cloud data, several avenues for further research and development could enhance its capabilities and address certain limitations. Enhancing the filtering step to better distinguish non-surface objects like vegetation and clutter remains an important area for improvement A better outline detection would strongly benefit the results and runtimes. The HDBSCAN approach would benefit from a distance metric producing less noise in the clustering and the cluster selection could maybe be customized to avoid undersegmentation. Also, a more realistic dataset with ground truth data for evaluation is needed.

Bibliography

[1] Akiba, Takuya ; Sano, Shotaro ; Yanase, Toshihiko ; Ohta, Takeru ; Koyama, Masanori: Optuna: A Next-generation Hyperparameter Optimization Framework. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. New York, NY, USA : Association for Computing Machinery, Juli 2019 (KDD '19), S. 2623–2631. – URL https://doi.org/10.1145/3292500.3330701. – Zugriffsdatum: 2023-08-16. – ISBN 9781450362016

[2] Bernardini, F. ; Mittleman, J. ; Rushmeier, H. ; Silva, C. ; Taubin, G.: The ball-pivoting algorithm for surface reconstruction. In: *IEEE Transactions on Visualization and Computer Graphics* 5 (1999), Oktober, Nr. 4, S. 349–359. – ISSN 1941-0506

[3] Brückner, Sebastian: *Pole and Plane Detection in Punktwolken*. 2021

[4] Brückner, Sebastian: *Comparing point cloud preprocessing for clustering based surface reconstruction*. 2023

[5] Brückner, Sebastian: *Extraktion ebener Flächen aus Punktwolken*. 2023

[6] Caesar, Holger ; Bankiti, Varun ; Lang, Alex H. ; Vora, Sourabh ; Liong, Venice E. ; Xu, Qiang ; Krishnan, Anush ; Pan, Yu ; Baldan, Giancarlo ; Beijbom, Oscar: nuScenes: A multimodal dataset for autonomous driving. URL http://arxiv.org/abs/1903.11027. – Zugriffsdatum: 2023-08-09, Mai 2020. – Forschungsbericht. arXiv:1903.11027 [cs, stat] type: article

[7] Campello, Ricardo J. G. B. ; Moulavi, Davoud ; Sander, Joerg: Density-Based Clustering Based on Hierarchical Density Estimates. In: Pei, Jian (Hrsg.) ; Tseng, Vincent S. (Hrsg.) ; Cao, Longbing (Hrsg.) ; Motoda, Hiroshi (Hrsg.) ; Xu, Guandong (Hrsg.): *Advances in Knowledge Discovery and Data Mining*. Berlin, Heidelberg : Springer, 2013 (Lecture Notes in Computer Science), S. 160–172. – ISBN 9783642374562

[8] CATTINI, Stefano ; ROVATI, Luigi ; DI CECILIA, Luca ; FERRARI, Luca: Comparison of the VLP-16 LiDAR system with an absolute interferometer. In: *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, Mai 2020, S. 1–6. – ISSN: 2642-2077. – ISSN 2642-2077

[9] CURTIN, Ryan ; MARCH, William ; RAM, Parikshit ; ANDERSON, David ; GRAY, Alexander ; ISBELL, Charles: Tree-Independent Dual-Tree Algorithms, PMLR, Mai 2013, S. 1435–1443. – URL https://proceedings.mlr.press/v28/curtin13.html. – Zugriffsdatum: 2023-07-28. – ISSN 1938-7228

[10] DELAUNAY, B.: Sur la sphère vide. In: *Bulletin de l'Academie des Sciences de l'URSS. Classe des sciences mathematiques et na* 1934 (1934), Nr. 6, S. 793–800

[11] DESCHAUD, Jean-Emmanuel: KITTI-CARLA: a KITTI-like dataset generated by CARLA Simulator. In: *arXiv preprint arXiv:2109.00892* (2021)

[12] DIERCKX, Paul: *An improved algorithm for curve fitting with spline functions.* Department of Computer Science, K.U.Leuven, Leuven, Belgium, 1981-07-01

[13] DIERCKX, Paul: *Curve and Surface Fitting with Splines.* Clarendon Press, 1995. – ISBN 9780198534402

[14] DIGNE, Julie: An Analysis and Implementation of a Parallel Ball Pivoting Algorithm. In: *Image Processing On Line* 4 (2014), Juli, S. 149–168. – URL https://www.ipol.im/pub/art/2014/81. – Zugriffsdatum: 2023-08-16. – ISSN 2105-1232

[15] DING, Tianyu ; ZHU, Zhihui ; DING, Tianjiao ; YANG, Yunchen ; VIDAL, Rene ; TSAKIRIS, Manolis ; ROBINSON, Daniel: Noisy Dual Principal Component Pursuit. In: CHAUDHURI, Kamalika (Hrsg.) ; SALAKHUTDINOV, Ruslan (Hrsg.): *Proceedings of the 36th International Conference on Machine Learning* Bd. 97, PMLR, 09–15 Jun 2019, S. 1617–1625. – URL https://proceedings.mlr.press/v97/ding19b.html

[16] EDELSBRUNNER, H. ; KIRKPATRICK, D. ; SEIDEL, R.: On the shape of a set of points in the plane. In: *IEEE Transactions on Information Theory* 29 (1983), Juli, Nr. 4, S. 551–559. – ISSN 1557-9654

[17] EDMONDS, J ; JOHNSON, E: *Matching, Euler tours and the Chinese postman. Mathematical programming.* 1973

[18] Glennie, C. L. ; Kusari, A. ; Facchin, A.: CALIBRATION AND STABILITY ANALYSIS OF THE VLP-16 LASER SCANNER. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-3/W4 (2016), S. 55–60. – URL https://isprs-archives.copernicus.org/articles/XL-3-W4/55/2016/

[19] Griffiths, David ; Boehm, Jan: SynthCity: A large scale synthetic point cloud. In: *ArXiv preprint*, 2019

[20] Group, Dynamic Robot S.: *Newer College Dataset*. – URL https://ori-drs.github.io/newer-college-dataset/. – Zugriffsdatum: 2023-08-16

[21] Gschwandtner, Michael ; Kwitt, Roland ; Uhl, Andreas ; Pree, Wolfgang: BlenSor: Blender sensor simulation toolbox. In: *Advances in Visual Computing: 7th International Symposium, ISVC 2011, Las Vegas, NV, USA, September 26-28, 2011. Proceedings, Part II 7* Springer (Veranst.), 2011, S. 199–208

[22] Hayes, TJ: Algorithms for curve and surface fitting. In: *Software for numerical mathematics* (1974), S. 219–233

[23] Holz, Dirk ; Ichim, Alexandru E. ; Tombari, Federico ; Rusu, Radu B. ; Behnke, Sven: Registration with the Point Cloud Library: A Modular Framework for Aligning in 3-D. In: *IEEE Robotics & Automation Magazine* 22 (2015), Dezember, Nr. 4, S. 110–124. – ISSN 1558-223X

[24] Huang, Hui ; Wu, Shihao ; Gong, Minglun ; Cohen-Or, Daniel ; Ascher, Uri ; Zhang, Hao (.: Edge-aware point set resampling. In: *ACM Transactions on Graphics* 32 (2013), Februar, Nr. 1, S. 9:1–9:12. – URL https://doi.org/10.1145/2421636.2421645. – Zugriffsdatum: 2023-08-14. – ISSN 0730-0301

[25] Huang, Jin ; Stoter, Jantien ; Peters, Ravi ; Nan, Liangliang: City3D: Large-Scale Building Reconstruction from Airborne LiDAR Point Clouds. In: *Remote Sensing* 14 (2022), Januar, Nr. 9, S. 2254. – URL https://www.mdpi.com/2072-4292/14/9/2254. – Zugriffsdatum: 2023-08-14. – ISSN 2072-4292

[26] Hughes, John F. ; Dam, Andries van ; McGuire, Morgan ; Sklar, David F. ; Foley, James D. ; Feiner, Steven K. ; Akeley, Kurt: *Computer graphics: principles and practice (3rd ed.)*. Boston, MA, USA : Addison-Wesley Professional, July 2013. – 1264 S. – ISBN 0321399528

[27] Knupp, Patrick M.: Algebraic mesh quality metrics for unstructured initial meshes. In: *Finite Elements in Analysis and Design* 39 (2003), Januar, Nr. 3, S. 217–241. – URL https://www.sciencedirect.com/science/article/pii/S0168874X02000707. – Zugriffsdatum: 2023-07-28. – ISSN 0168-874X

[28] Lafarge, Florent ; Mallet, Clément: Creating Large-Scale City Models from 3D-Point Clouds: A Robust Approach with Hybrid Representation. In: *International Journal of Computer Vision* 99 (2012), August, Nr. 1, S. 69–85. – URL https://doi.org/10.1007/s11263-012-0517-8. – Zugriffsdatum: 2023-08-14. – ISSN 1573-1405

[29] Lee, Hongjae ; Jung, Jiyoung: Clustering-Based Plane Segmentation Neural Network for Urban Scene Modeling. In: *Sensors* 21 (2021), Januar, Nr. 24, S. 8382. – URL https://www.mdpi.com/1424-8220/21/24/8382. – Zugriffsdatum: 2023-08-13. – ISSN 1424-8220

[30] Liu, Rong ; Wang, Jinling ; Zhang, Bingqi: High Definition Map for Automated Driving: Overview and Analysis. In: *The Journal of Navigation* 73 (2020), März, Nr. 2, S. 324–341. – URL https://www.cambridge.org/core/journals/journal-of-navigation/article/abs/high-definition-map-for-automated-driving-overview-and-analysis/7FFB4F68B9C27F4312AF8DCD553205FE. – Zugriffsdatum: 2023-08-16. – ISSN 0373-4633, 1469-7785

[31] Long Nguyen, H. ; Belton, D. ; Helmholz, P.: COMPARATIVE STUDY OF AUTOMATIC PLANE FITTING REGISTRATION FOR MLS SPARSE POINT CLOUDS WITH DIFFERENT PLANE SEGMENTATION METHODS. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W4 (2017), S. 115–122. – URL https://isprs-annals.copernicus.org/articles/IV-2-W4/115/2017/

[32] Maalek, Reza ; Lichti, Derek D. ; Ruwanpura, Janaka Y.: Robust Segmentation of Planar and Linear Features of Terrestrial Laser Scanner Point Clouds Acquired from Construction Sites. In: *Sensors* 18 (2018), März, Nr. 3, S. 819. – URL https://www.mdpi.com/1424-8220/18/3/819. – Zugriffsdatum: 2023-08-14. – ISSN 1424-8220

[33] Manivasagam, Sivabalan ; Wang, Shenlong ; Wong, Kelvin ; Zeng, Wenyuan ; Sazanovich, Mikita ; Tan, Shuhan ; Yang, Bin ; Ma, Wei-Chiu ; Urtasun,

Raquel: Lidarsim: Realistic lidar simulation by leveraging the real world. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, S. 11167–11176

[34] McInnes, Leland ; Healy, John: Accelerated Hierarchical Density Based Clustering. In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, November 2017, S. 33–42. – ISSN: 2375-9259. – ISSN 2375-9259

[35] McInnes, Leland ; Healy, John ; Astels, Steve: hdbscan: Hierarchical density based clustering. In: *The Journal of Open Source Software* 2 (2017), mar, Nr. 11. – URL https://doi.org/10.21105%2Fjoss.00205

[36] Nan, Liangliang ; Sharf, Andrei ; Zhang, Hao ; Cohen-Or, Daniel ; Chen, Baoquan: SmartBoxes for interactive urban reconstruction. In: *ACM SIGGRAPH 2010 papers*. New York, NY, USA : Association for Computing Machinery, Juli 2010 (SIGGRAPH '10), S. 1–10. – URL https://doi.org/10.1145/1833349.1778830. – Zugriffsdatum: 2023-08-14. – ISBN 9781450302104

[37] Nan, Liangliang ; Wonka, Peter: Polyfit: Polygonal surface reconstruction from point clouds. In: *Proceedings of the IEEE International Conference on Computer Vision*, 2017, S. 2353–2361

[38] Plachetka, Christopher ; Sertolli, Benjamin ; Fricke, Jenny ; Klingner, Marvin ; Fingscheidt, Tim: 3DHD CityScenes: High-Definition Maps in High-Density Point Clouds. In: *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, Oktober 2022, S. 627–634

[39] Poullis, Charalambos: A Framework for Automatic Modeling from Point Cloud Data. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35 (2013), November, Nr. 11, S. 2563–2575. – ISSN 1939-3539

[40] Reinsch, Christian H.: Smoothing by spline functions. In: *Numerische Mathematik* 10 (1967), Oktober, Nr. 3, S. 177–183. – URL https://doi.org/10.1007/BF02162161. – Zugriffsdatum: 2023-08-09. – ISSN 0945-3245

[41] Saket, S. ; Pandya, Sharnil: An Overview of Partitioning Algorithms in Clustering Techniques, URL https://www.semanticscholar.org/paper/An-Overview-of-Partitioning-Algorithms-in-Saket-Pandya/f36151291b161060b95f469b13c2fd935bd08027. – Zugriffsdatum: 2023-08-10, 2016

[42] Sampath, Aparajithan ; Shan, Jie: Segmentation and Reconstruction of Polyhedral Building Roofs From Aerial Lidar Point Clouds. In: *IEEE Transactions on Geoscience and Remote Sensing* 48 (2010), März, Nr. 3, S. 1554–1567. – ISSN 1558-0644

[43] Schönherr, Nils: *Kartografierung des HAW-Campus mittels 3D-Lidar und IMU.* 2021

[44] Shewchuk, Jonathan R.: Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In: Lin, Ming C. (Hrsg.) ; Manocha, Dinesh (Hrsg.): *Applied Computational Geometry: Towards Geometric Engineering* Bd. 1148. Springer-Verlag, Mai 1996, S. 203–222. – From the First ACM Workshop on Applied Computational Geometry

[45] Smith, Lindsay I.: A tutorial on principal components analysis. (2002)

[46] Sun, Shaohui ; Salvaggio, Carl: Aerial 3D Building Detection and Modeling From Airborne LiDAR Point Clouds. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 6 (2013), Juni, Nr. 3, S. 1440–1449. – ISSN 2151-1535

[47] Tsakiris, Manolis C. ; Vidal, Rene: Dual Principal Component Pursuit, URL https://www.cv-foundation.org/openaccess/content_iccv_2015_workshops/w24/html/Tsakiris_Dual_Principal_Component_ICCV_2015_paper.html. – Zugriffsdatum: 2023-08-14, 2015, S. 10–18

[48] Wilson, Benjamin ; Qi, William ; Agarwal, Tanmay ; Lambert, John ; Singh, Jagjeet ; Khandelwal, Siddhesh ; Pan, Bowen ; Kumar, Ratnesh ; Hartnett, Andrew ; Pontes, Jhony K. ; Ramanan, Deva ; Carr, Peter ; Hays, James: Argoverse 2: Next Generation Datasets for Self-Driving Perception and Forecasting. URL http://arxiv.org/abs/2301.00493. – Zugriffsdatum: 2023-08-09, Januar 2023. – Forschungsbericht. arXiv:2301.00493 [cs] type: article

[49] Xia, Shaobo ; Chen, Dong ; Wang, Ruisheng ; Li, Jonathan ; Zhang, Xinchang: Geometric Primitives in LiDAR Point Clouds: A Review. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 13 (2020), S. 685–707. – ISSN 2151-1535

[50] Xiao, Aoran ; Huang, Jiaxing ; Guan, Dayan ; Zhan, Fangneng ; Lu, Shijian: Transfer Learning from Synthetic to Real LiDAR Point Cloud for Semantic Segmentation. URL http://arxiv.org/abs/2107.05399. – Zugriffsdatum: 2023-08-17, Dezember 2021. – Forschungsbericht. arXiv:2107.05399 [cs] type: article

[51] Xu, Yusheng ; Stilla, Uwe: Toward Building and Civil Infrastructure Reconstruction From Point Clouds: A Review on Data and Key Techniques. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 14 (2021), S. 2857–2885. – ISSN 2151-1535

[52] Zhang, Liqiang ; Li, Zhuqiang ; Li, Anjian ; Liu, Fangyu: Large-scale urban point cloud labeling and reconstruction. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 138 (2018), April, S. 86–100. – URL https://www.sciencedirect.com/science/article/pii/S0924271618300376. – Zugriffsdatum: 2023-08-14. – ISSN 0924-2716

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

[54] Zhou, Yichao ; Huang, Jingwei ; Dai, Xili ; Luo, Linjie ; Chen, Zhili ; Ma, Yi: HoliCity: A City-Scale Data Platform for Learning Holistic 3D Structures. (2020). – arXiv:2008.03286 [cs.CV]

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original