Chair of Computational Modeling and Simulation
TUM School of Engineering and Design
Technical University of Munich

Reconstruction of truncated instance point clouds with the help of generative models

Scientific work to obtain the degree

Master of Science (M.Sc.)

at the TUM School of Engineering and Design
of the Technical University of Munich.

Supervised by Prof. André Borrmann
M.Sc. Fiona Collins
Dr. Stavros Nousias
Chair of Computational Modeling and Simulation

M.Sc. Zhiqi Hu
University of Cambridge

Submitted by Hannes Kirn [redacted]
[redacted]

Submitted on 30. January 2024

Acknowledgment

While working for NavVis GmbH for over one year as a Working Student in the Project Management Team, my interest in working with Point Clouds (PCs) and improving them grew considerably. Additionally, it provided me with significant insights on the problematic aspects of PCs. Moreover, my specialization on Building Information Modeling (BIM) in my master studies allowed me a better understanding on how to use the BIM software tools employed in this Master Thesis and on the impact of PCs in the Scan-to-BIM pipeline.

On the other hand, I would like to thank all my NavVis colleagues, who provided the information used in this Thesis and were always willing to answer my questions. Especially I want to mention here Philipp Foerst who always referred me to the right colleagues for asking specific questions. Additionally, Tim Habigt and Christian Werner, who gave me helpful recommendations in e.g., post-processing of my noisy partial point clouds and various topics.

Furthermore, I thank my supervisors Fiona Collins, Zhiqi Hu (University of Cambridge) and Stavros Nousias for always supporting me with their expertise both on the data generation part as well as on applying Machine Learning (ML) algorithms. Additionally, I'm sincerely grateful for making the one month extension of this master thesis possible so that I was able to focus on a more state-of-the-art dataset creation by using the NavVis VLX3 setup in Heidelberg Light Detection and Ranging (LiDAR) Operations Simulator (HELIOS++). Additionally, I would like to thank Wei "Norman" Guo, Application Engineer from HESAI Technology for providing me most of their LiDAR sensor specifications to implement them as accurate as possible in HELIOS++.

Moreover, I would also like to thank my girlfriend, María, for always being there to share my thoughts on all topics, support me and listen to my struggles.

Finally, I would like to express my sincere gratitude to my parents for enabling me to study what I have wished for, and for their love and support whenever I was struggling to achieve the goals I set for myself.

Abstract

Currently, Point Clouds (PCs) are widely used especially in the Architecture, Engineering and Construction (AEC) industry for various use cases, such as planning equipment relocation in a factory; layout, and assembly line planning; global manufacturing operations; 5S and Gemba Walks; best practice sharing; visual interfaces, and in general Scan-to-BIM. Therefore, an accurate PC is very useful for getting the best information out of it and will be more helpful in processing the use cases more efficiently.

This Master Thesis aims to improve PCs especially in Mechanical, Electrical and Plumbing (MEP) environments with the use of Machine Learning (ML) algorithms such as Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN) and Point Completion Networks (PCNs). For this procedure, synthetic data is firstly generated as training datasets before completing the real truncated PC clusters. An evaluation is performed afterwards for the applied methods. For the procedure without pre-voxelization, the assignments of different materials were modified for each instance to later be able to segment each PC instances in the output of the simulations. Afterwards, a PCN was applied to complete those components.

For applying the Voxel-DCGAN method, the voxelization step on the PC is needed. This can be done by the given code from Min (2004);(Nooruddin & Turk, 2003). The given Voxel-DCGAN method which contains a 3D shape generative model can be used for completion.

Zusammenfassung

Heutzutage werden Punktwolken vor allem in der Architektur-, Ingenieur- und Bauwesen (AEC) für verschiedene Anwendungsfälle eingesetzt. Dazu gehören z.B. die Planung der Verlagerung von Anlagen in Fabriken, die Planung von Layouts und Montagelinien, globale Fertigungsabläufe, 5S- und Gemba-Rundgänge, der Austausch bewährter Verfahren, visuelle Schnittstellen oder allgemein Scan-to-BIM. Daher ist eine genaue Punktwolke sehr nützlich, um die besten Informationen aus ihr herauszuholen und den Anwendungsfall effizienter zu bearbeiten.

Diese Masterarbeit zielt auf die Verbesserung von Punktwolken ab, insbesondere in den Bereichen Mechanik, Elektrik und Sanitär mit Hilfe von maschinellen Lernalgorithmen wie Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN) und Point Completion Networks (PCNs). Für dieses Verfahren werden zunächst synthetische Daten als Trainingsdatensätze generiert, bevor die realen, abgeschnittenen Punktwolken-Cluster vervollständigt werden. Anschließend erfolgt eine Auswertung der angewandten Methoden. Bei dem Verfahren ohne Vorvoxelierung wurde für jede Instanz eine andere Materialzuweisung verwendet, um jede Punktwolken-Instanz in der Ausgabe der Simulationen zu segmentieren. Anschließend wird ein PCN zur Vervollständigung dieser Komponenten eingesetzt.

Für die Anwendung der Voxel-DCGAN Methode ist der Schritt der Voxelisierung der Punktwolke erforderlich. Dies kann mit dem gegebenen Code von MIN (2004);(NOORUDDIN & TURK, 2003) durchgeführt werden. Die angegebene Voxel-DCGAN Methode, die ein generatives 3D-Formmodell enthält, kann zur Vervollständigung verwendet werden.

Contents

List of Figures

List of Algorithms

Acronyms

3D-IWGAN	3D Improved Wasserstein Generative Adversarial Network
AEC	Architecture, Engineering and Construction
BIM	Building Information Modeling
CAD	Computer-Aided Design
CC	CloudCompare
CD	Chamfer Distance
CNN	Convolutional Network
DCGAN	Deep Convolutional Generative Adversarial Network
E57	LiDAR Point Cloud Data Format
EMD	Earth Mover's Distance
GAN	Generative Adversarial Network
GCE	Group Contextual Encoding
GeoTIFF	Geographic Tagged Image File Format
GUID	Globally Unique Identifier
HELIOS++	Heidelberg LiDAR Operations Simulator
IFC	Industry Foundation Classes
LiDAR	Light Detection and Ranging
MEP	Mechanical, Electrical and Plumbing
ML	Machine Learning
MLS	Mobile Laser Scanning
MTL	Material Library File
OBJ	Wavefront Object Format
PC	Point Cloud
PCN	Point Completion Network
PLY	Polygon File Format
PY	Python File
ReLU	Rectified Linear Unit
TUM	Technical University of Munich
TXT	Text File
VLS	Virtual Laser Scanning

Continued on next page

(Continued)

VOX	Voxel Format
Voxel-DCGAN	Voxel-Deep Convolutional Generative Adversarial Networks
XML	Extensible Markup Language
XYZ	XYZ Point Cloud Format

Chapter 1

Introduction

1.1 Building Information Modeling (BIM)

Building Information Modeling (BIM) is a digital representation of the physical and functional characteristics of a building or infrastructure, encompassing geometry, spatial relationships, geographic information, and various attributes of the components within the structure (see Figure 1.1).

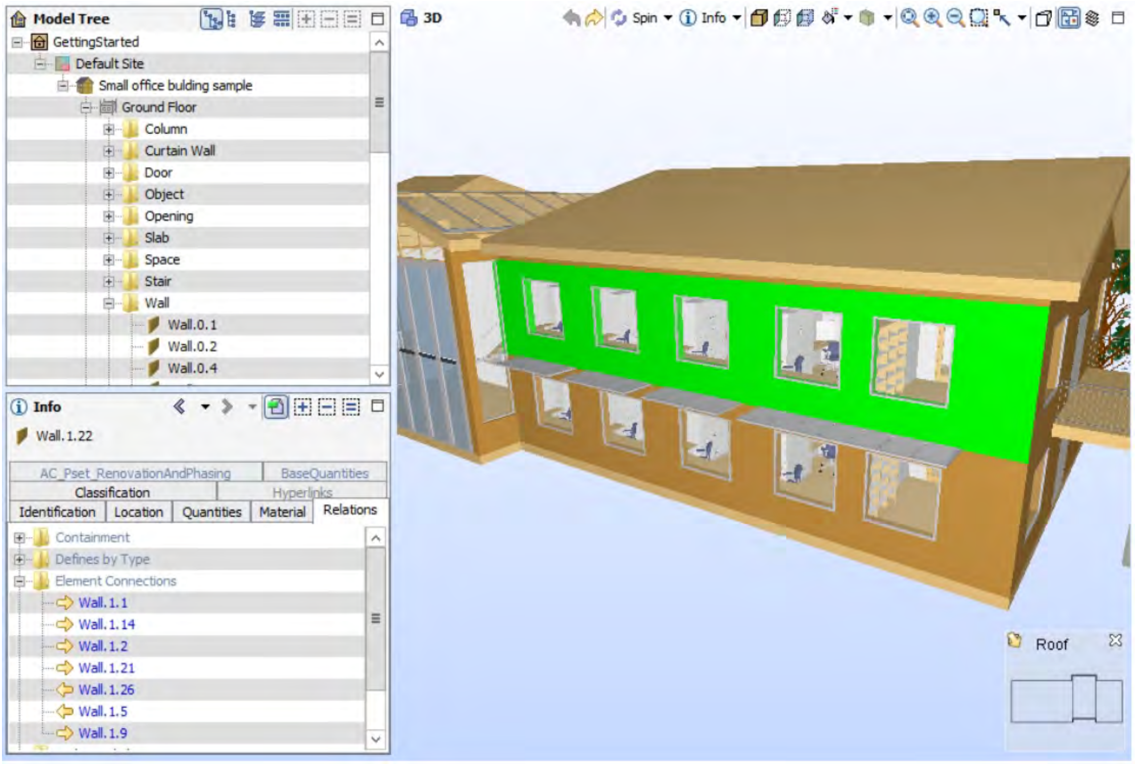


Figure 1.1: A BIM model comprises both the 3D geometry of each building element as well as a rich set of semantic information provided by attributes and relationships (BORRMANN et al., 2018)

Furthermore, as shown in Figure 1.2, BIM serves as a collaborative and interoperable platform that integrates multidisciplinary data throughout the entire lifecycle of a construction project, from conceptualization and design to construction, operation, and maintenance. It facilitates the efficient exchange and management of information among stakeholders, enabling enhanced decision-making, improved coordination, and increased overall project performance. Additionally, BIM promotes a holistic approach to building processes by providing a comprehensive and dynamic database that supports the creation, analysis, and visualization of 3D models, fostering greater accuracy, transparency, and sustainability in the construction industry (BORRMANN et al., 2018).

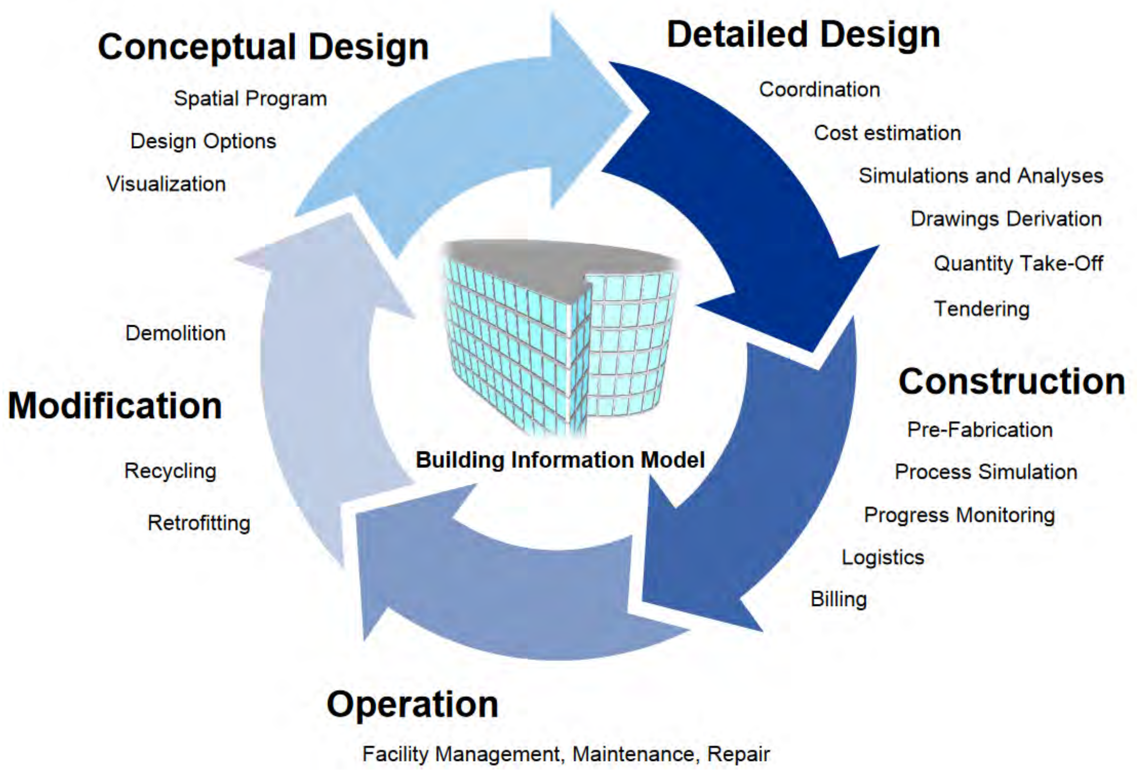


Figure 1.2: The fundamentals of BIM relies on the continuous use and low-loss handover of digital information across the entire lifecycle of a built facility (BORRMANN et al., 2018)

1.2 Point Cloud (PC)

A Point Cloud (PC) is a collection of data points in space, typically obtained through laser scanning or photogrammetry techniques, representing the geometric and possibly radiometric characteristics of physical objects within the scanned environment (SEAN HIGGINS, 2021). PCs are extensively used in various fields, including computer graphics, computer vision, remote sensing, geomatics, and robotics. The data points in a PC collectively form a digital model of the surface or structure they represent, see an example of a PC in Figure 1.3.

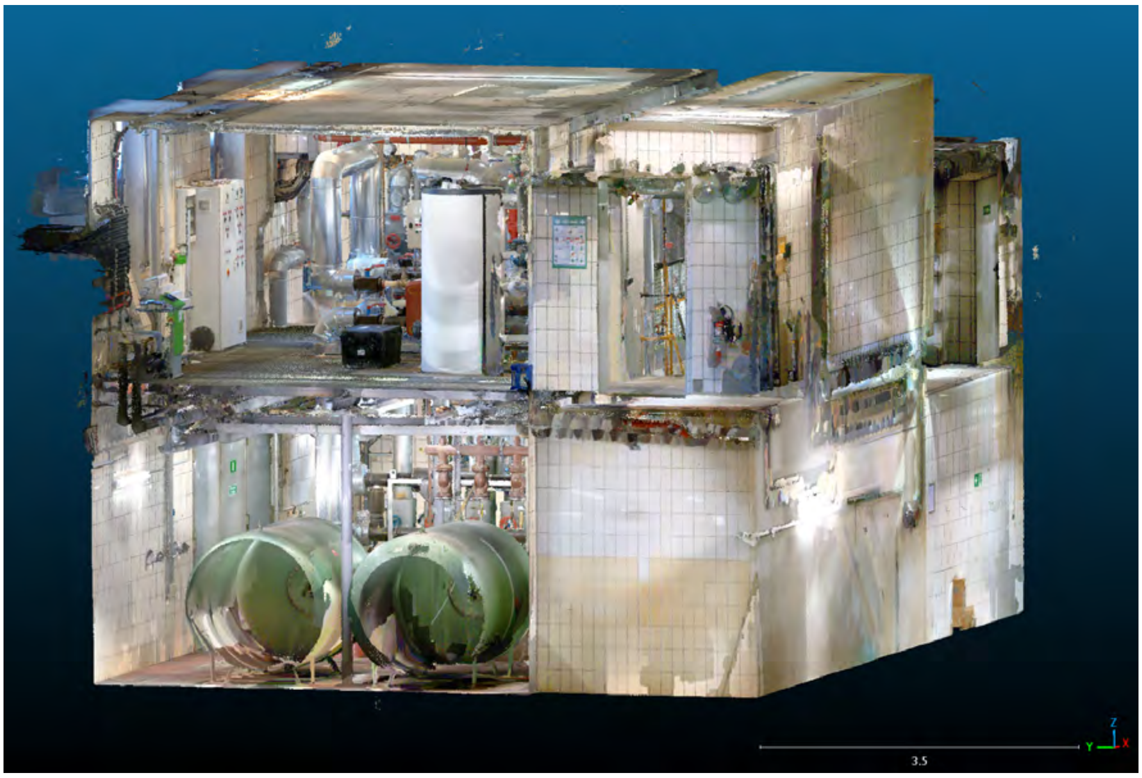


Figure 1.3: Example of a PC captured with the NavVis VLX3 device visualized in Cloud-Compare (CC)

In the context of BIM, the integration of PC data involves extracting geometric information and attributes from the PC to construct a detailed and accurate digital model of the existing structure. This model encompasses the spatial relationships, dimensions, and visual characteristics of the scanned elements. By incorporating PC data into the BIM framework, stakeholders can gain a more precise and comprehensive understanding of the as-built conditions of a facility, enabling improved decision-making, precise analysis, and effective collaboration throughout the various stages of a construction project or facility management.

1.3 Generative Models

Generative models are a class of Machine Learning (ML) models designed to generate new data samples that resemble a given dataset. These models learn the underlying patterns and structures of the training data and can subsequently generate new, realistic instances that share similar characteristics. Generative models are particularly prominent in fields such as Artificial Intelligence, Computer Vision, Natural Language Processing, and Generative Art. There are various types of generative models, each with its own approach to capturing and reproducing the distribution of the training data (see Figure 1.4). Inputs and outputs to these models can include text, images, sounds, animation, 3D models, or other types of data.

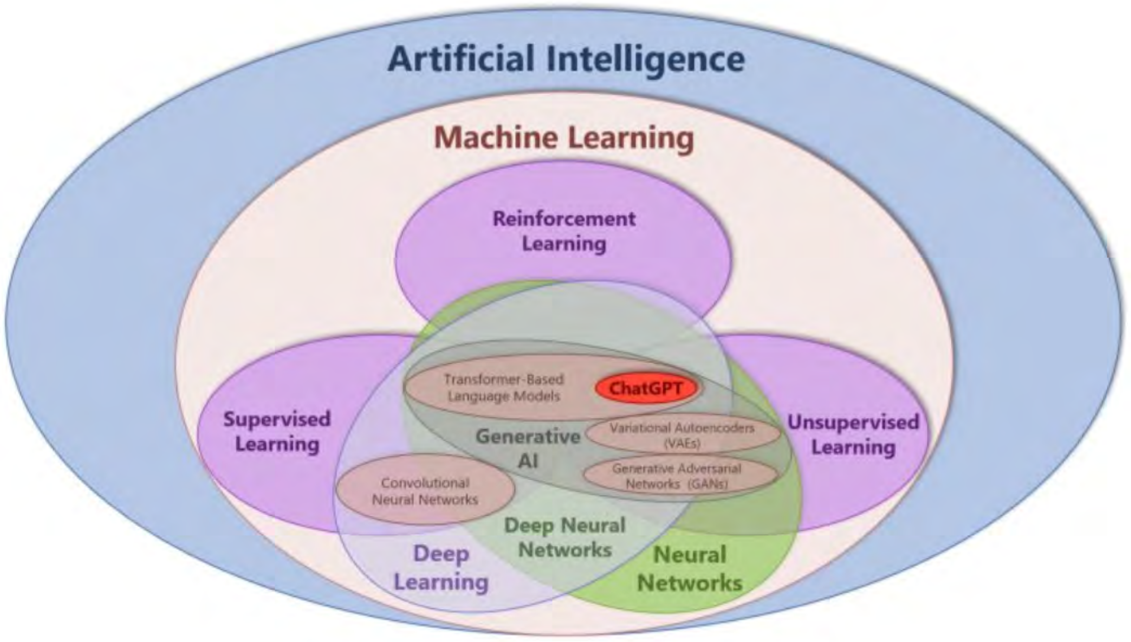


Figure 1.4: Venn Diagram by Jeff Winter; url:https://gf.linkedin.com/posts/karenkilroy_thank-you-jeff-winter-for-this-fascinating-activity-7050432311791669248-_nkT

In KARPATHY et al. (2016) the intuition of the approach, to firstly collect a big amount of data and then training it for generating similar data with a generative model, follows the given famous quote from Richard Feynman:

> *"What I cannot create, I do not understand."*
> - Richard Feynman

Applied on generative models it means, that when creating similar data out of the initial data, the model is continuously improving to understand its initial data. The reason for this is that neural networks, used as generative models, have a number of parameters significantly smaller than the amount of data they are trained on, so the models are forced to discover and efficiently internalize the essence of the data to generate it. Therefore, for long term-applications they hold the potential to automatically learn the natural features of a dataset, whether categories or dimensions or entirely something else.

One recent generative model is the Deep Convolutional Generative Adversarial Network (DCGAN) of RADFORD et al. (2016). It is made up of standard convolutional neural network components, such as deconvolutional layers (reverse of convolutional layers), fully connected layers, etc. The network has millions of parameters that can be optimized, and the goal is to find a setting of these parameters that makes samples generated from random codes look like the training data. It is basically a back and forth learning process between a generator network and a second discriminative network that tries to classify samples as either coming from the true distribution or the model distribution. Every time the discriminator notices a difference between the two distributions the generator slightly adjusts its parameters to make it go away, until at the end (in theory) the generator exactly reproduces the true data distribution and the discriminator random guesses, unable to find a difference.

1.4 Unsupervised Learning vs. Supervised Learning

Julianna Delua (2021) establishes that Supervised and Unsupervised Learning are two fundamental approaches in the field of Artificial Intelligence and Machine Learning (ML). In Figure 1.4, their relation in the Artificial Intelligence context is displayed as well. Supervised Learning relies on labeled datasets to train algorithms to predict outcomes accurately, while Unsupervised Learning analyzes unlabeled datasets to discover hidden patterns autonomously.

Supervised Learning encompasses classification and regression tasks. Classification involves categorizing data into specific groups, such as spam filtering or image recognition, while regression predicts numerical values based on input variables, like sales forecasts. On the other hand, Unsupervised Learning focuses on clustering similar data points, finding associations between variables, and reducing the dimensionality of complex datasets.

The primary difference between the two lies in the use of labeled data. Supervised Learning requires human intervention to label input and output data, making it more accurate but also more labor-intensive. In contrast, Unsupervised Learning operates independently, identifying patterns without predefined labels, though human validation may still be necessary. A simple example of both learning techniques is shown below in Figure 1.5.

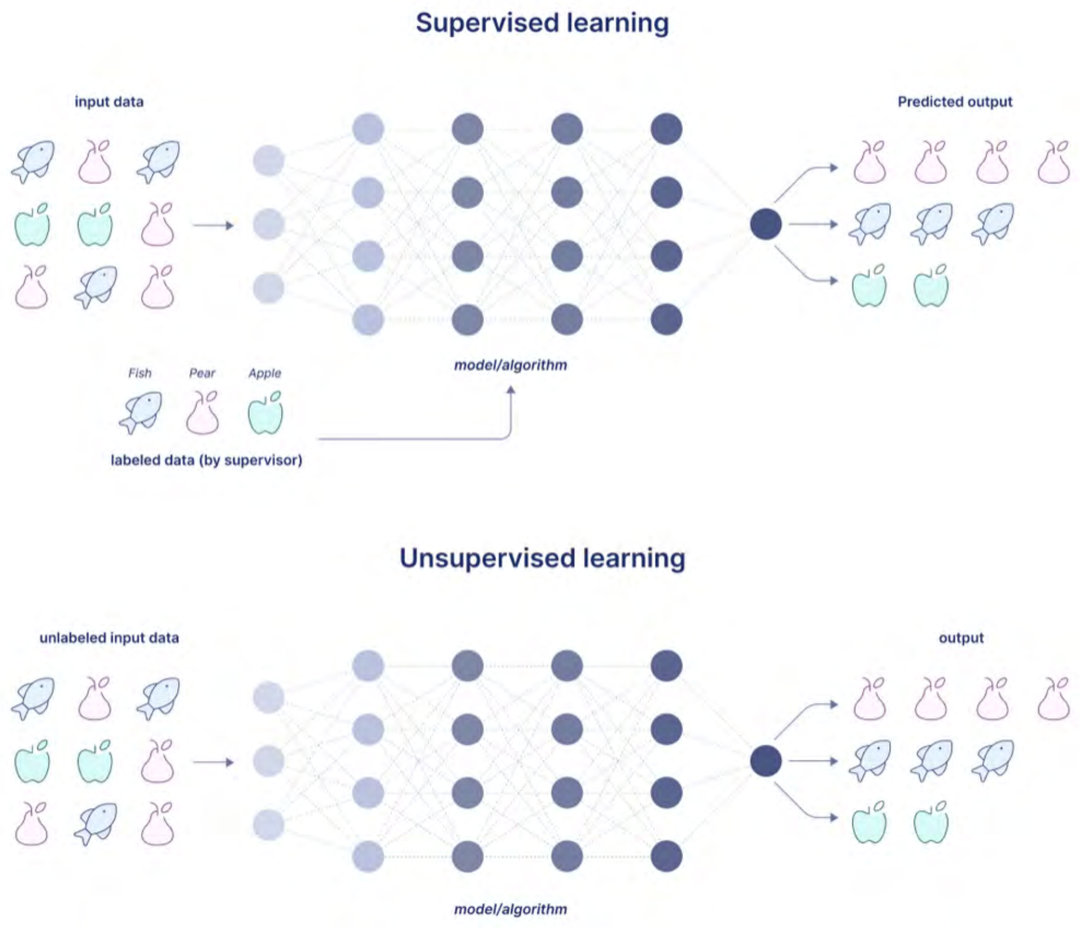


Figure 1.5: Simple example of Unsupervised Learning and Supervised Learning (KASSIANI NIKOLOPOULOU, 2023)

Supervised Learning is suitable for tasks like spam detection and weather forecasting, where outcomes are known and well-defined. Unsupervised Learning excels in anomaly detection and recommendation systems, where insights are derived from large datasets.

While Supervised Learning is simpler and more straightforward, Unsupervised Learning is computationally complex and requires robust tools to handle unstructured data. However, Semi-Supervised Learning offers a compromise by utilizing both labeled and unlabeled data, making it ideal for scenarios where feature extraction is challenging and data volume is high, such as medical imaging analysis (JULIANNA DELUA, 2021).

1.5 Motivation

Point Cloud (PC) completion for Mechanical, Electrical and Plumbing (MEP) components is helpful for various reasons. The main downstream method that it facilitates is the improved model reconstruction. Which in turn is more efficient for companies and helps them saving costs, as an accurate representation, including geometric information, for every MEP is given. Thus improving the process, as an expert would not have to manually complete such geometric information, which is needed for the modeling process. This would result in a more effective way of applying clash detection and coordination in complex environments/ MEP systems as well as in the planning, maintaining and execution of MEP installations.

Although, model reconstruction is described as a manual process in the previous paragraph, it can also be done automatically (see next paragraph). E.g., by using Hough Transform to fit cylindrical elements to the Point Cloud (PC). Still, such automatic methods can fail in case of drastic occlusions. PC completion can help for this step.

WANG et al. (2021) explain how automatic modeling is done for MEP components given by terrestrial laser scanning data (PC). Currently, modeling is often done by using a PC as a reference which is more time consuming and takes a lot of labor effort. For this automatic purpose, geometric information is extracted to refine geometric information which is then usable for modeling. The extracted information and connection relationships are imported into Dynamo to automatically generate the parametric Building Information Modeling (BIM) model. However, this automatic modeling process would not be in the scope of this Master Thesis, but Figure 1.6 shows the result of this mentioned procedure in WANG et al. (2021).

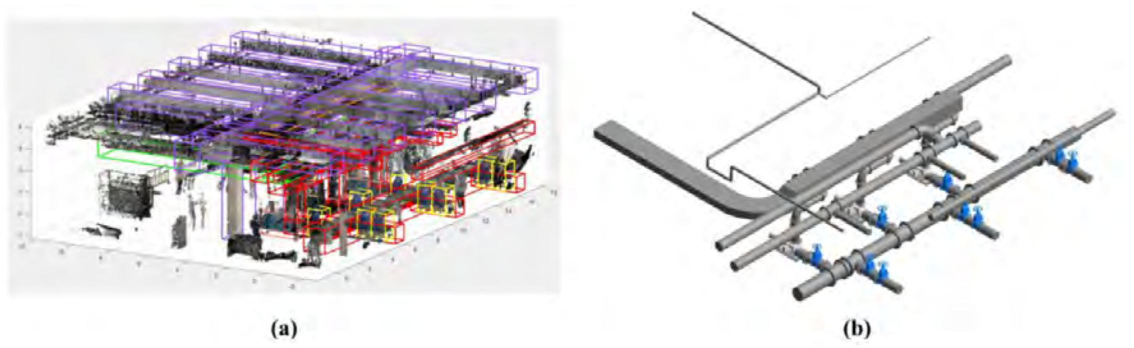


Figure 1.6: (a) Detected types of MEP components (b) Generated models via Dynamo in Revit (WANG et al., 2021)

Due to incomplete or occluded PCs the following bullet points lead to various costs-intensive challenges for companies:

- Rework or Redoing.
- Delays in Project Timelines.
- Increased Iterations and Design.
- Operational Inefficiencies.
- Safety Risks and Accidents.
- Loss of Competitive Advantage.

Considering the aforementioned, PC completion should be done directly or automized while scanning as it is happening already e.g., for autonomous driving where cars are completed over far distances (YUAN et al., 2019) to avoid the previously mentioned cost implications.

1.6 Outline

The Thesis document is divided into the following chapters:

- In Chapter 2, the theoretical background and the state-of-the-art necessary to understand the entire research work is detailed.
- In Chapter 3, the process of setting up artificial and real-world scan data generation for this thesis is explained. The application of the two Point Cloud (PC) completion methods Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN) and Point Completion Network (PCN) on self-created datasets from Chapter 3.2 are described. At the end of this chapter, examples of reconstructed completed PC instances are provided.
- In Chapter 4, the research questions are reviewed and the results of both methods are compared regarding their usability in the Scan-to-BIM workflow. Additionally, both methods are tested on a pre-segmented real-world PC.
- Finally, in Chapter 5, a summary of the research work is presented, and concluding statements are provided. Furthermore, the future scope of extension of this work is also discussed.

Chapter 2

Theoretical Background

This chapter delves into the theoretical foundations and concepts to foster a better comprehension of this thesis's terminologies, techniques, and procedures. PointNet and Point Completion Networks (PCNs) address different challenges in Point Cloud (PC) processing and serve different purposes within the broader field of 3D data analysis and manipulation.

2.1 Voxel-Deep Convolutional Generative Adversarial Nets (Voxel-DCGAN)

In this Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN) repository (Source Code) a 3D shape generative model is developed leveraging Deep Convolutional Generative Adversarial Networks (DCGANs) (RADFORD et al., 2016) enhanced with advanced techniques from improved-Generative Adversarial Network (GAN) (Source Code) (SALIMANS et al., 2016). In recent years, the widespread adoption of Supervised Learning using Convolutional Networks (CNNs) has significantly impacted computer vision applications. However, Unsupervised Learning with CNNs has not received commensurate attention.

RADFORD et al. (2016) aim to address this disparity by introducing a class of CNNs termed DCGANs. These networks are distinguished by specific architectural constraints and demonstrate considerable promise for Unsupervised Learning tasks. Through extensive training on diverse image datasets, their study provides compelling evidence of the DCGANs' ability to learn hierarchical representations, ranging from instance parts to entire scenes, within both the generator and discriminator components.

Furthermore, RADFORD et al. (2016) explore the transferability of these learned features to novel tasks, showcasing their potential as versatile image representations. This investigation serves to enhance our understanding of Unsupervised Learning methods in the realm of computer vision, providing valuable insights to the broader field of Machine Learning (ML) research. Figure 2.1 illustrates the DCGAN generator architecture used for Large-scale Scene Understanding (LSUN) scene modeling, emphasizing the absence of fully connected or pooling layers.

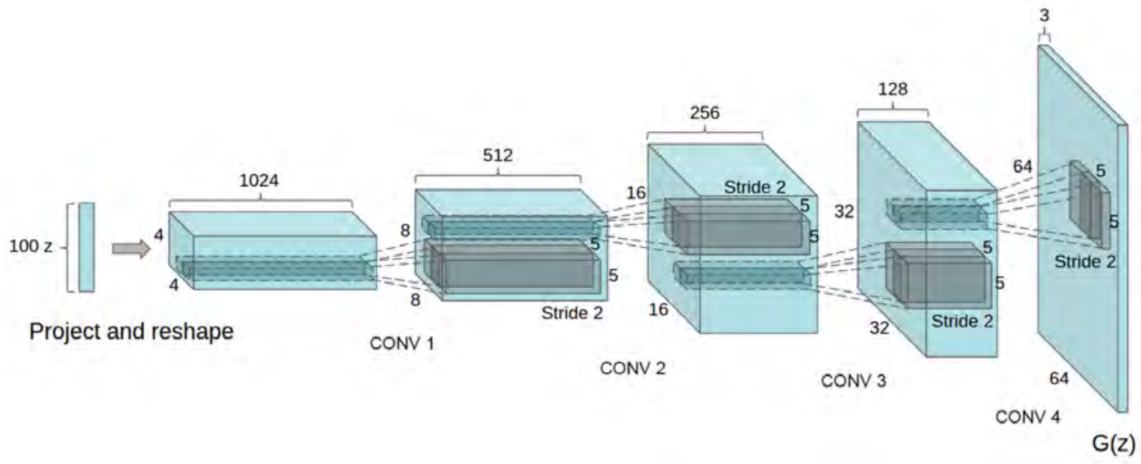


Figure 2.1: **DCGAN generator used for LSUN scene modeling.** A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions then convert this high level representation into a 64 × 64 pixel image. (RADFORD et al., 2016)

The study applies Deep Convolutional Generative Adversarial Networks (DCGANs) to the Large-scale Scene Understanding (LSUN) bedrooms dataset, demonstrating scalability and generalization performance. Additionally, a deduplication technique is employed using autoencoders and semantic hashing to prevent memorization of training examples. This process effectively removes near-duplicate images from the training set. The study also evaluates DCGANs on a dataset of human faces scraped from the web, demonstrating competitive performance in classifying faces without data augmentation. The paper concludes by highlighting the stability and effectiveness of DCGANs in learning representations for supervised and generative tasks, suggesting future directions for extending the framework to other domains like video and audio.

By applying Figure 2.1 on 3D Voxel data, the following similar full architecture of the 3D Improved Wasserstein Generative Adversarial Network (3D-IWGAN) in Figure 2.2 can be taken used for the in the beginning mentioned Voxel-DCGAN method. The noise vector size e.g., is set to 500 instead of the displayed 100 in Figure 2.2.

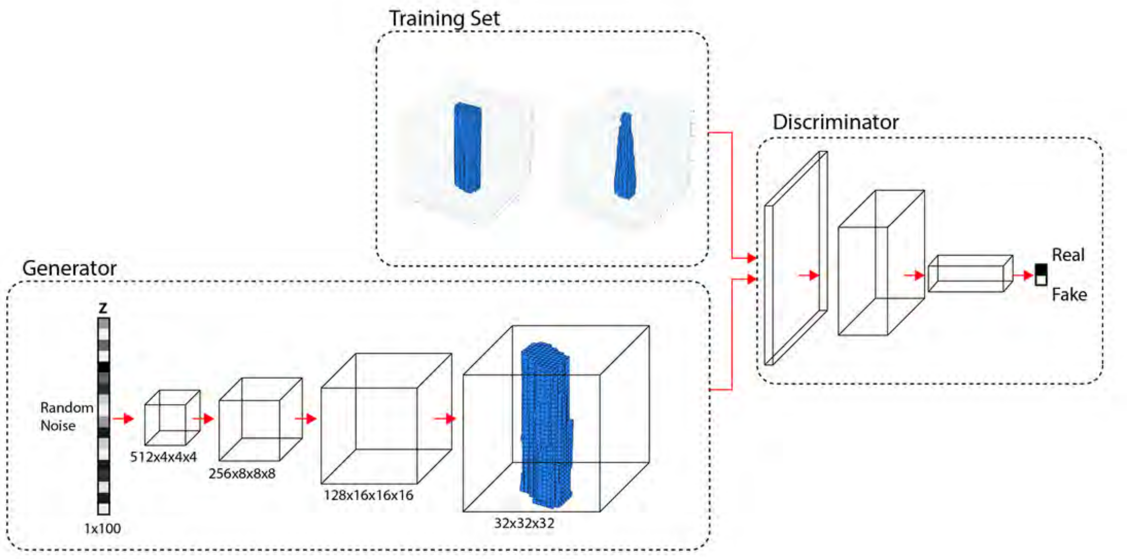


Figure 2.2: The diagram shows the generator and discriminator networks that make-up the 3D-IWGAN. (INTELLIGENCE & LAB, n.d.)

SALIMANS et al. (2016) discuss advancements in training Generative Adversarial Networks (GANs) with a focus on improving stability and generating high-quality sample which were used as well for developing the Voxel-DCGAN. It introduces techniques such as feature matching, minibatch discrimination, historical averaging, label smoothing, and virtual batch normalization to encourage convergence in GAN training. SALIMANS et al. (2016) emphasizes the application of these techniques to Semi-Supervised Learning, achieving state-of-the-art results on various datasets like MNIST, CIFAR-10, SVHN, and ImageNet.

Instead of solely optimizing likelihood or avoiding reliance on labels, SALIMANS et al. (2016) aim to produce visually realistic images and improve Semi-Supervised Learning performance. Notably, they introduce feature matching to stabilize GANs by aligning generated data statistics with real data. Additionally, they propose minibatch discrimination to prevent mode collapse and improve diversity in generated samples.

Experimental results demonstrate the effectiveness of these techniques, particularly in semi-supervised learning tasks, where their approach outperforms previous methods. They also present evaluation metrics such as the Inception score to assess sample quality, which correlates well with human judgment.

In conclusion, SALIMANS et al. (2016) highlight practical advancements in stabilizing GAN training and improving sample generation quality, with promising applications in Semi-Supervised Learning. Further research is suggested to deepen theoretical understanding and expand the scope of these techniques.

2.2 "BIM-to-Scan" for Scan-to-BIM

NOICHL et al. (2021) propose a method utilizing a laser scan simulation tool to generate realistic synthetic Ground Truth Point Cloud (PC) data for Scan-to-BIM processes. This approach aims to address the scarcity of publicly available Ground Truth datasets, crucial for training artificial intelligence models in semantic segmentation of PC data. By employing the open-source laser scanning simulation tool HELIOS++ (WINIWARTER et al., 2022) from the domain of Geo-sciences, they simulate laser scans based on 3D models, providing fully labeled data for training deep learning networks. Comparison with real laser scan data reveals that the synthetic PCs exhibit realistic characteristics but may overestimate precision. In spite of remaining manual effort for model preparation, the approach offers scalability and flexibility, showing promise for advancing Scan-to-BIM processes. Further testing is needed to evaluate the performance of artificial intelligence models trained on synthetic data when applied to real-world datasets. Figure 2.3 illustrates the domains of Scan-to-BIM and Scan-vs-BIM by using the Digital Twin framework.

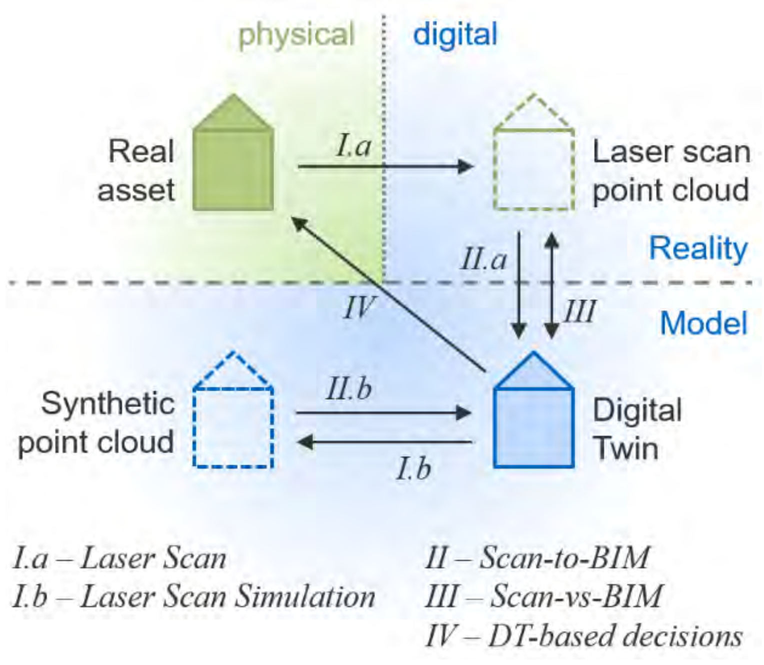


Figure 2.3: Digital Twin framework to illustrate the domains of Scan-to-BIM and Scan-vs-BIM (NOICHL et al., 2021)

In Figure 2.3, I.b stands for the simulation tool HELIOS++. The "Digital Twin" can be a 3D model of e.g., a building in the OBJ format. The "Real asset" is the building in the real environment and the "Laser scan point cloud" represents a scan taken by e.g., the NavVis VLX3, a Mobile Laser Scanning (MLS) device.

2.3 Point Cloud Segmentation and Model Retrieval via PointNet

Qi et al. (2017) explain Point Cloud (PC) segmentation via PointNet. The three different classification or segmentation options are shown in Figure 2.4. Voxelization or rendering is not needed, PointNet learns both global and local point features and is an excellent approach for 3D recognition tasks.

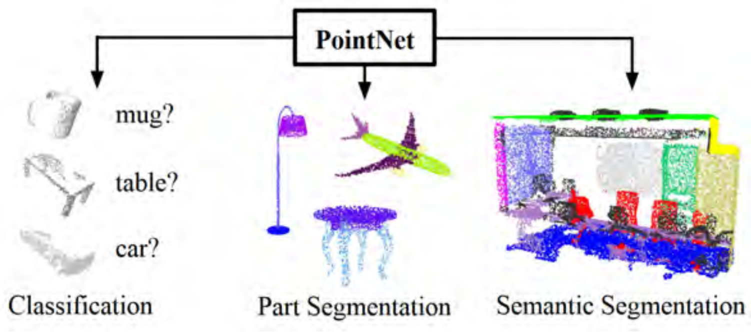


Figure 2.4: PointNet applications (Qi et al., 2017)

The main part of this approach is the single symmetric function, denominated Max Pooling, which allows the network to learn a set of optimization functions and criteria that filter interesting or informative points of the PC and encode the reason for their selection. The final fully connected layers of the network summarize these learnt optimal values into the global descriptor, for the entire shape, as shown in Figure 2.4 (shape classification) or uses them to predict a label per point (shape segmentation). Figure 2.5 describes the whole PointNet architecture, including the segmentation network as an extension of the classification network and the Max Pooling.

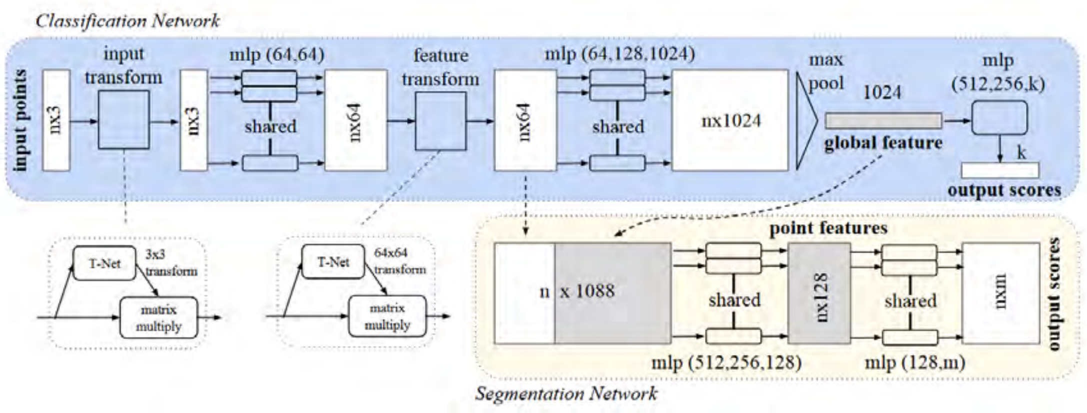


Figure 2.5: PointNet architecture (Qi et al., 2017)

The bullet points underneath further explain Figure 2.5.

- The classification network takes n points as input (the 3 in "nx3" means 3 three coordinates of a point), applies input and feature transformations. Then summarizes point features by Max Pooling.

- The output is classification scores for k classes.
- The segmentation network concatenates global and local features, along with outputs per point scores.
- “mlp” = multi-layer perceptron.
- Numbers in brackets refer to layer sizes.
- Batchnorm is used for all layers with Rectified Linear Unit (ReLU).
- Dropout layers are used for the last mlp in the classification net.

Additionally, PointNet was used for model retrieval. Figure 2.6 shows, the top-5 retrieved Computer-Aided Design (CAD) models from ModelNet test split in the categories chair, plant, nightstand, and bathtub (top to bottom). The figures in a red box are the retrieved models in wrong categories. For each given query shape from ModelNet test split, its global signature (output of the layer before the score prediction layer) given by the classification PointNet, was computed. In addition, similar shapes in the training split were retrieved by the nearest neighbor search.

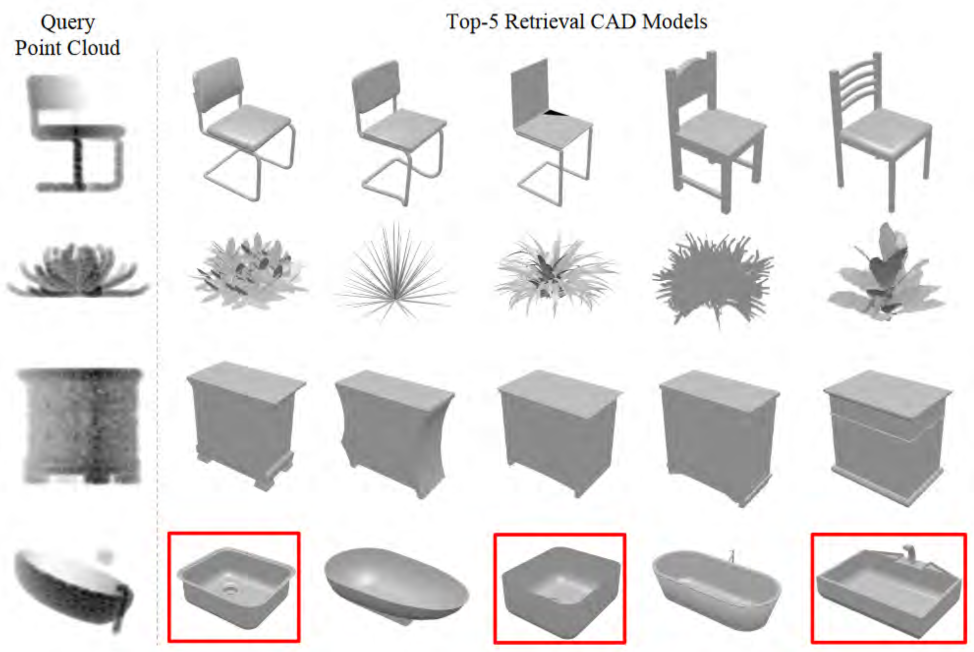


Figure 2.6: Model retrieval from PC with using PointNet (Qi et al., 2017)

2.4 Group Contextual Encoding via PointNet++

Xu et al. (2020) (Source Code) present a novel approach to enhancing 3D Point Cloud (PC) scene understanding tasks by leveraging global contextual information efficiently. It extend a contextual encoding layer originally designed for 2D tasks to 3D PC scenarios. This layer learns a set of code words in the feature space of the 3D PC to characterize global semantic context and subsequently learns a global contextual descriptor to reweight feature maps accordingly. However, they find that data sparsity becomes a significant issue in 3D PC scenarios, leading to saturation in performance when the number of code words increases. To address this, they propose a Group Contextual Encoding (GCE) method, dividing channels into groups and performing encoding on group-divided feature vectors to facilitate learning global context in grouped subspaces for 3D PCs.

The effectiveness and generalizability of the proposed method are evaluated on three widely-studied 3D PC tasks. Experimental results demonstrate significant improvements over existing methods, including outperforming VoteNet - a state-of-the-art method for 3D instance detection in PCs - by 3 mean average precision (mAP) on the SUN-RGBD - a benchmark dataset commonly used for evaluating algorithms related to scene understanding, particularly in indoor environments - benchmark and by 6.57 mAP on ScanNet - another widely used benchmark dataset for 3D scene understanding, focusing on indoor environments. Additionally, compared to the baseline of PointNet++, the proposed method achieves an accuracy of 86%, surpassing the baseline by 1.5%.

The paper contributes by extending the contextual encoding layer to 3D PC scenarios, proposing the GCE method for effective learning of global context, and demonstrating superior performance on multiple 3D benchmarks. The code for the proposed method has been released to facilitate result reproduction.

The method is integrated into PointNet++ as GCE-PointNet++, with improvements observed across various experiments. Ablation studies investigate the effects of code word number, channel number, and grouping methods, leading to the selection of optimal parameters for subsequent experiments.

Main results show that the proposed method surpasses previous state-of-the-art methods on benchmarks such as SUN-RGBD and ScanNet, achieving mAP values of 60.7 and 60.8, respectively, demonstrating its efficacy in enhancing 3D instance detection in PCs.

2.5 Point Cloud Completion via Point Completion Network (PCN)

In Yuan et al. (2019) the shape completion network Point Completion Network (PCN) is described. In comparison to PointNet++, which which focuses on Point Cloud (PC) classification, segmentation and feature extraction, the PCN is specifically used for the PC completion process. The difference of the PCN to other shape completion methods lies in that it operates on raw PCs without any structural assumption (e.g., symmetry) or annotation (e.g., semantic class) about the underlying shape. This prevents high memory costs and the loss of geometric information caused by voxelization, while allowing the network to generate more fine-grained completions.

The PCN, as a learning-based approach completes shapes with a parameterized model (a deep neural network) that directly maps the partial input to a complete shape, which offers fast inference and better generalization. Consider a scenario where X represents a collection of 3D points situated on the visible surfaces of an instance, derived from either a singular scan or a series of scans via a 3D sensor. Meanwhile, Y constitutes a comprehensive array of 3D points uniformly selected from both the observed and unobserved surfaces of said instance. The objective revolves around the shape completion predicament, which entails forecasting Y based on the knowledge of X. It is worth noting that in this context, X might not necessarily constitute a subset of Y, and there exists no explicit correspondence between the points in X and those in Y, given their independent sampling from the inherent surfaces of the object.

To address this challenge, Supervised Learning techniques are employed. By harnessing a sizable synthetic dataset where instances of X and Y can be readily obtained, a neural network is trained to directly anticipate Y from X. This network exhibits versatility across various instance categories and operates without presumptions regarding the structural attributes of the underlying instances, such as symmetry or planarity.

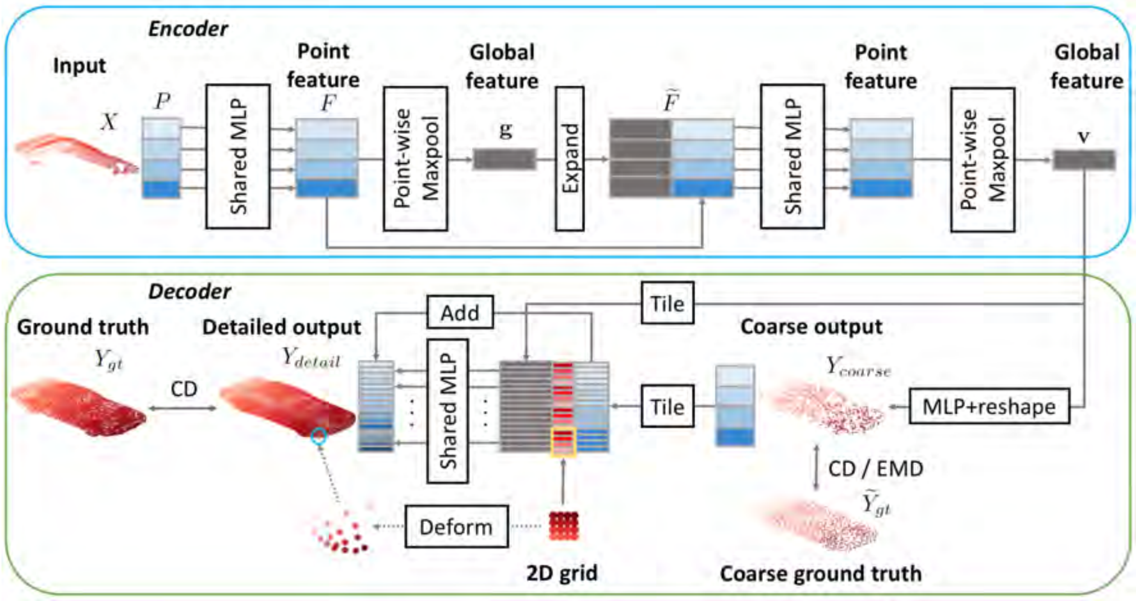


Figure 2.7: **PCN Architecture.** Here, the encoder condenses the input point cloud X into a feature vector denoted as **v**. Subsequently, the decoder utilizes **v** to initially generate a coarse rendition of the output Y, denoted as Y_{coarse}, followed by a more detailed output denoted as Y_{detail}. Each colored rectangle signifies a matrix row, with matching colors indicating identical content.

As depicted in Figure 2.7, the Point Completion Network (PCN) functions as an encoder-decoder network. The encoder receives the input Point Cloud (PC) X and produces a k-dimensional feature vector. Subsequently, the decoder utilizes this feature vector to generate both a coarse output PC, denoted as Y_{coarse}, and a detailed output PC, termed Y_{detail}. The loss function L is computed by comparing the Ground Truth PC, Y_{gt}, with the decoder's outputs, facilitating the training of the entire network via backpropagation.

It is important to note that, unlike an autoencoder, the network is not explicitly mandated to replicate the input points in its output. Instead, it learns a mapping from the domain of partial observations to that of complete shapes. Following this, YUAN et al. (2019) delve into the specific design aspects of the encoder, decoder, and the employed loss function.

Experiments on synthetic ShapeNet data - a large-scale dataset and taxonomy for 3D shape understanding in computer vision and graphics research - demonstrate the superiority of the proposed method over strong baselines, with significant improvements in completion results. Additionally, the model's robustness to occlusion and noise is tested, showing consistent performance even with highly occluded and noisy inputs.

Furthermore, the method is evaluated on real-world LiDAR scans from the KITTI dataset, achieving consistent completions despite the sparsity of the input PCs. The completed PCs are shown to improve the results of PC registration algorithms, enhancing rotational and translational accuracy.

Despite its effectiveness, the proposed model exhibits some failure modes, particularly with instances consisting of disconnected parts or containing thin structures such as wires. However, overall, the approach offers a promising solution for shape completion tasks, demonstrating strong performance across various scenarios and datasets.

The partial PCs in YUAN et al. (2019) were generated by back-projecting 2.5D depth images into 3D instead, by using subsets of the complete PC to bring the input distribution closer to real-world sensor data. The segmented completed cars in Figure 2.8 generated by the PCN show the effectiveness of the method when working with incomplete PC instances (Source Code).

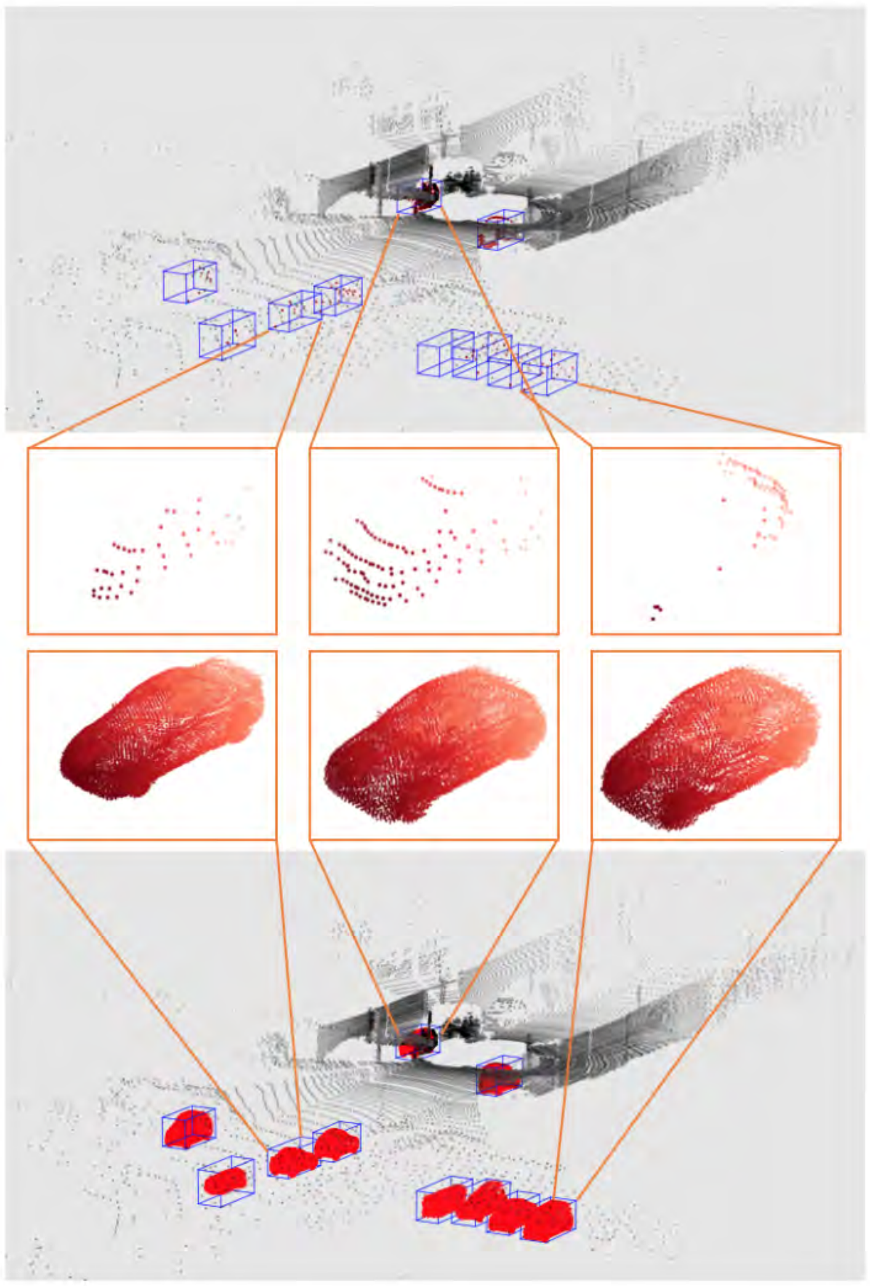


Figure 2.8: Results of applying PCN on raw LiDAR Scan (YUAN et al., 2019)

2.6 Comparison of different Point Cloud Completion Methods

In FEI et al. (2022), different Point Cloud (PC) completion methods regarding various approaches (point-based, view-based, convolution-based, graph-based, generative model-based, and transformer-based) are compared. As it was already mentioned the segmentation of such sparse PC (in e.g., distance) is crucial for the 3D detection and PC completion step afterwards. In the past, the application of 2D methods was transferred to 3D tasks, but the use of the voxelization or 3D convolution lead to high computational effort. Nowadays, PointNet and PointNet++ are quite successful in avoiding this with directly processing 3D coordinates (FEI et al., 2022).

Chamfer Distance (CD) and Earth Mover's Distance (EMD) are the most frequently used performance criteria for 3D Point Cloud (PC) completion (See Equation 2.1 and Equation 2.2). In general, CD finds the minimum distance between two sets of points, and EMD evaluates the reconstruction quality of the PC. S_1 is the output PC and S_2 the Ground Truth PC.

$$CD(\mathbf{S_1}, \mathbf{S_2}) = \frac{1}{|\mathbf{S_1}|} \sum_{x \in \mathbf{S_1}} \min_{y \in \mathbf{S_2}} \|x - y\|_2 + \frac{1}{|\mathbf{S_2}|} \sum_{y \in \mathbf{S_2}} \min_{x \in \mathbf{S_1}} \|y - x\|_2 \tag{2.1}$$

$$EMD(\mathbf{S_1}, \mathbf{S_2}) = \min_{\phi : \mathbf{S_1} \rightarrow \mathbf{S_2}} \frac{1}{|\mathbf{S_1}|} \sum_{x \in \mathbf{S_1}} \|x - \phi(x)\|_2 \tag{2.2}$$

The aforementioned paper mentions other performance criteria, but considering that those are less frequently used for 3D PC completion, they are not going to be further explained. On the other hand, regarding the methods compared Fei et al. (2022) conclude that the Generative Adversarial Networks (GANs) and graph-based methods lead to excellent results in the PC completion task. Thus, the focus should be on using the combination of both in the future.

Additionally, the F1-Score is an evaluation parameter as well. The F1-Score, a fundamental metric for classification models, balances precision and recall, representing their harmonic mean. The F-Score can be modified into F0.5, F1, and F2 based on the measure of weightage given to precision over recall depending on the use case. It is calculated using the formula (Nikolaj Buhl, 2023):

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{2.3}$$

Where:

- Precision is the ratio of true positive predictions to the total number of positive predictions, calculated as:

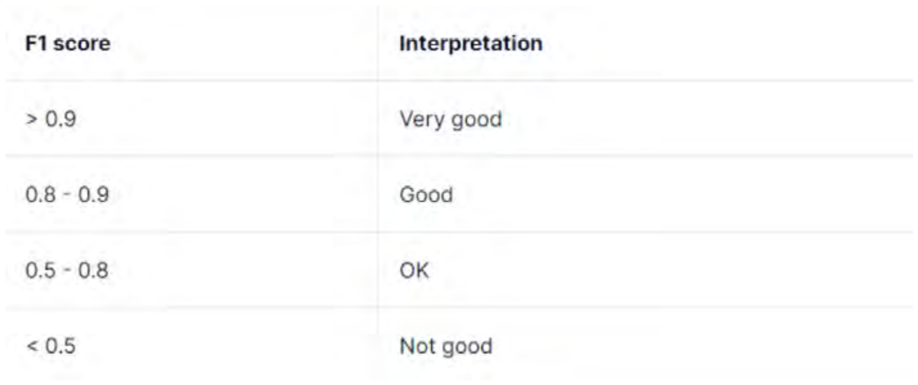
$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{2.4}$$

- Recall is the ratio of true positive predictions to the total number of actual positive instances, calculated as:

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{2.5}$$

This formula ensures a balanced assessment of a model's performance, particularly beneficial in scenarios with varied precision and recall values. The F1-Score ranges between 0 and 1, with higher values indicating better model performance.

A high F1-Score signifies a well-balanced model that achieves both high precision and recall. It implies that the model correctly identifies positive instances while minimizing false positives and false negatives. Conversely, a low F1-Score suggests a trade-off between precision and recall or poor performance in one or both aspects. As a general rule of thumb, the F1 score value can be interpreted as follows:



F1 score	Interpretation
> 0.9	Very good
0.8 - 0.9	Good
0.5 - 0.8	OK
< 0.5	Not good

Figure 2.9: Interpretation of different F-Scores (NIKOLAJ BUHL, 2023)

The threshold for the F-score refers to the cutoff value used to determine whether a predicted value should be classified as positive or negative. By adjusting the threshold, it is possible to control the trade-off between precision and recall. A higher threshold tends to increase precision (fewer false positives) but may decrease recall (more false negatives), while a lower threshold typically increases recall (fewer false negatives) but may decrease precision (more false positives). Choosing an appropriate threshold depends on the specific requirements and priorities of the task at hand.

However, the F1-Score has limitations, notably with imbalanced datasets where one class dominates. Variants like the F2 and F-beta scores offer alternatives to address these challenges. The F2-Score prioritizes recall, while the F-beta score allows flexibility in balancing precision and recall based on task requirements.

2.7 Objectives

The main focus of this Thesis is to achieve a basic framework for completing Mechanical, Electrical and Plumbing (MEP) components captured by a laser scanner in the real world. The Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN) and the Point Completion Network (PCN) will be applied on a self-generated dataset by Heidelberg Light Detection and Ranging (LiDAR) Operations Simulator (HELIOS++) (WINIWARTER et al., 2022), a general-purpose software package for simulation of terrestrial, mobile and airborne laser scanning survey. The core objectives of this research are:

- To create a virtual simulation framework in HELIOS++ using Building Information Modeling (BIM) models for the data generation part. This requires creating an accurate HELIOS++ compliant description and a step-by-step guideline to receive the expected results.

- To perform virtual laser scanning simulations with HELIOS++, include the respective BIM models, make laser scanning settings as realistic as possible and create way-points. The simulations output would focus on segmented realistic partial instance Point Clouds (PCs) which can be used for further Machine Learning (ML) purposes.

- To prepare the segmented partial instance PCs for applying the Voxel-DCGAN and the PCN successfully. For this, PC post-processing and data format converting are performed.

- When the dataset is set up correctly, the Voxel-DCGAN and PCN codes will get adapted for receiving good results regarding my dataset. Both methods should be able to complete the different components.

- At the end, conclusions shall be drawn from the simulations and the PC completion methods done in this research work. Furthermore, some possible extensions for this research work shall be discussed.

Chapter 3

Methodology and Implementation

3.1 Method Overview

On the next page in Figure 3.1, a graphical high-level overview of my workflow is provided. First of all, the initial data was merged and edited in SimpleBIM (Jiri Hietanen, Sakari Lehtinen, 2019) for creating a good foundation for further steps.

Afterwards, the resulted Industry Foundation Classes (IFC) files were converted into Wavefront Object Format (OBJ) files which were used for two different steps. On one hand, the **Ground Truth creation** and on the other hand as a scene input in Heidelberg Light Detection and Ranging (LiDAR) Operations Simulator (HELIOS++).

For the simulations in HELIOS++ additional Extensible Markup Language (XML) files were configured to set up the simulations as realistic as possible for the Mobile Laser Scanning (MLS) device NavVis VLX3 with its Light Detection and Ranging (LiDAR) specifications and positions. As a next step, two different types of partial Point Clouds (PCs) were created by changing the `accuracy_m` parameter in the Scanner XML respectively. Idealistic (noise-free) occluded partial PCs by avoiding all errors of a LiDAR scan by applying the `accuracy_m` parameter to 0.0 and noisy occluded partial PCs by setting the `accuracy_m` parameter to 0.01.

The Ground Truths were generated by importing the OBJ files of the storeys in Cloud-Compare (CC) and sampling the automatically Globally Unique Identifier (GUID) based segmented mesh instances. As a final step, the geometries were reconstructed via poisson reconstruction, sampled with the same amount of points for each instances and convert into Polygon File Formats (PLYs) files.

Both types of files (OBJ and PLY Ground Truth files) were then used as an input in the **Training process** for the Voxel-DCGAN and the PCN. Additionally, for the PCN, the mentioned simulated idealistic (noise-free) occluded partial PCs were used as an input for the PCN training.

In the **Inference process** of both methods, completed instances were generated. Important to note is that the results are highly influenced by non-matching point cloud instances due to different orientations and additional for the PCN, the offset between partial PCs and their Ground Truths PCs.

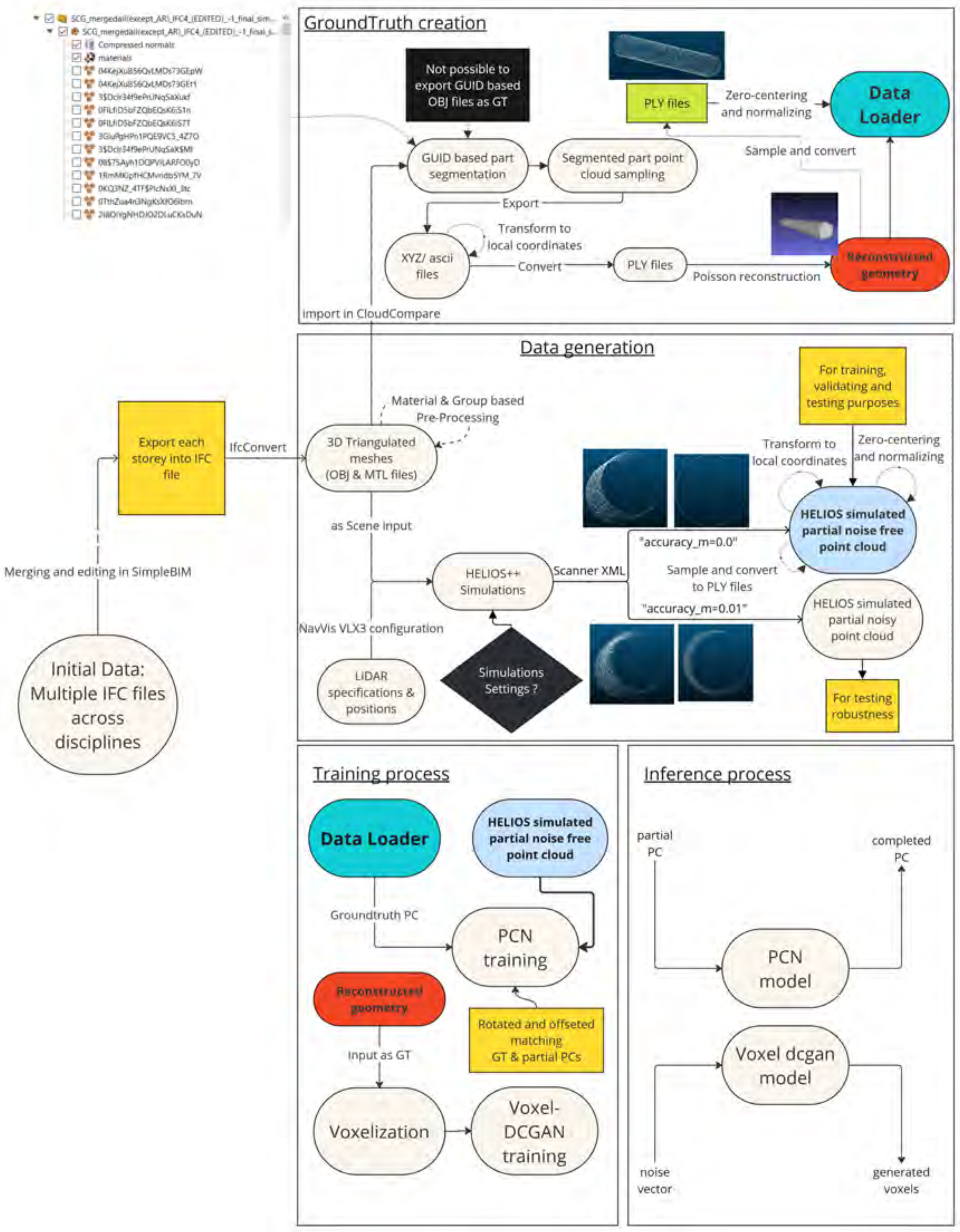

SCG_mergedall(except_AR)_IFC4_(EDITED)_-1_final_sim...
SCG_mergedall(except_AR)_IFC4_(EDITED)_-1_final_s...
Compressed normals
materials
04KejXuB56QvLMDs73GEpW
04KejXuB56QvLMDs73GEr1
3$Dclr34f9ePrUNqSaXukf
0FlLfiD5bFZQbEQsK6iS1n
0FlLfiD5bFZQbEQsK6iS7T
3GluRgHPn1PQE9VC5_4Z7O
3$Dclr34f9ePrUNqSaX$MI
0B$75Ayh1DQPVILARFO0yD
1RmMKpfHCMvndbSYM_7V
0KQ3NZ_4TF$PlcNxXl_Itc
0TthZua4n3NgKsXfO6Ibm
2I8OYgNHDJO2DLuCKsDuN
GroundTruth creation
Not possible to export GUID based OBJ files as GT
GUID based part segmentation
Segmented part point cloud sampling
PLY files
Zero-centering and normalizing
Data Loader
Sample and convert
Export
Transform to local coordinates
XYZ/ ascii files
Convert
PLY files
Poisson reconstruction
Reconstructed geometry
import in CloudCompare
Data generation
For training, validating and testing purposes
Material & Group based Pre-Processing
Export each storey into IFC file
IfcConvert
3D Triangulated meshes (OBJ & MTL files)
Transform to local coordinates
Zero-centering and normalizing
HELIOS simulated partial noise free point cloud
as Scene input
"accuracy_m=0.0"
Merging and editing in SimpleBIM
HELIOS++ Simulations
Scanner XML
Sample and convert to PLY files
"accuracy_m=0.01"
HELIOS simulated partial noisy point cloud
NavVis VLX3 configuration
LiDAR specifications & positions
Simulations Settings ?
For testing robustness
Initial Data: Multiple IFC files across disciplines
Training process
Data Loader
HELIOS simulated partial noise free point cloud
Groundtruth PC
PCN training
Reconstructed geometry
Rotated and offseted matching GT & partial PCs
Input as GT
Voxelization
Voxel-DCGAN training
Inference process
partial PC
completed PC
PCN model
Voxel dcgan model
noise vector
generated voxels

Figure 3.1: Graphical high-level method overview

3.2 Data Generation

To be able to apply the Point Cloud (PC) completion methods training data is firstly needed. The two procedures explored in this Thesis for generating such data are explained in the following sections. The first one uses the artificial possibility with using the Virtual Laser Scanning (VLS) software HELIOS++ (WINIWARTER et al., 2022). The second one includes real scan data provided by NavVis GmbH; a company founded 2013 in Munich as a Start-up out of the Technical University of Munich (TUM) which specializes on constructing Mobile Laser Scanning (MLS) devices and software for using the scanned data.

3.2.1 Virtual Laser Scanning (VLS) - Heidelberg Light Detection and Ranging (LiDAR) Operations Simulator (HELIOS++)

Industry Foundation Classes (IFC) Merging and Converting

As a first step the provided IFC files across disciplines (see below) were merged for building storeys in the SimpleBIM software (JIRI HIETANEN, SAKARI LEHTINEN, 2019):

- Submodel_different_discipline_example1_IFC4.ifc
- Submodel_architecture_example.ifc (not merged)
- Submodel_different_discipline_example1.ifc
- Submodel_different_discipline_example2.ifc
- Submodel_different_discipline_example3.ifc
- Submodel_different_discipline_example2_IFC4.ifc
- Submodel_different_discipline_example4.ifc
- Submodel_different_discipline_example5.ifc
- Submodel_different_discipline_example6.ifc
- Submodel_different_discipline_example7.ifc
- Submodel_different_discipline_example8.ifc
- Submodel_different_discipline_example9.ifc
- Submodel_different_discipline_example10.ifc

To be able to merge Submodel_different_discipline_example1_IFC4.ifc and Submodel_-different_discipline_example2_IFC4.ifc with all the other files above related, both files must be transferred from an IFC2x3 format to an IFC4 format, as it is the only possibility when using SimpleBIM. The transferring of the files from one format to the other was done with the help of Revit software (AUTODESK, 2000). The IFC files were imported and then exported with the chosen IFC4 Design Transfer View as the output format. The Submodel_different_discipline_example1_IFC4.ifc file was finally placed at the correct location with the help of the `Placement Editor` in SimpleBIM.

The provided IFC models for each storey come from different disciplines, such as Architecture, Engineering, etc., and contain various definitions of each storey per discipline. In consideration, all storeys were created and merged by the help of the `Containment Editor`, `Select all visible instances` and the `area selection function` in SimpleBIM. The Submodel_different_discipline_example2_IFC4.ifc file was divided into each storey by running the tool `Add Min and Max Elevation and Center Point`. All the elements of this file were selected, dragged and dropped into the property window and sorted regarding the respective storey height. Furthermore, the `Location Editor` was used for instances that are multiple building storeys high. First, the instances were cutted using the `Location Prisms` (see Figure fig. 3.2). Then they were created out of the given Spaces for each storey in the provided IFC files, and afterwards, they were added to the correct storeys. Nonetheless, it was not possible to move some of the multiple building storeys high cut instances, so for the export they were excluded for each storey. Finally, each storey was exported into an IFC file again.

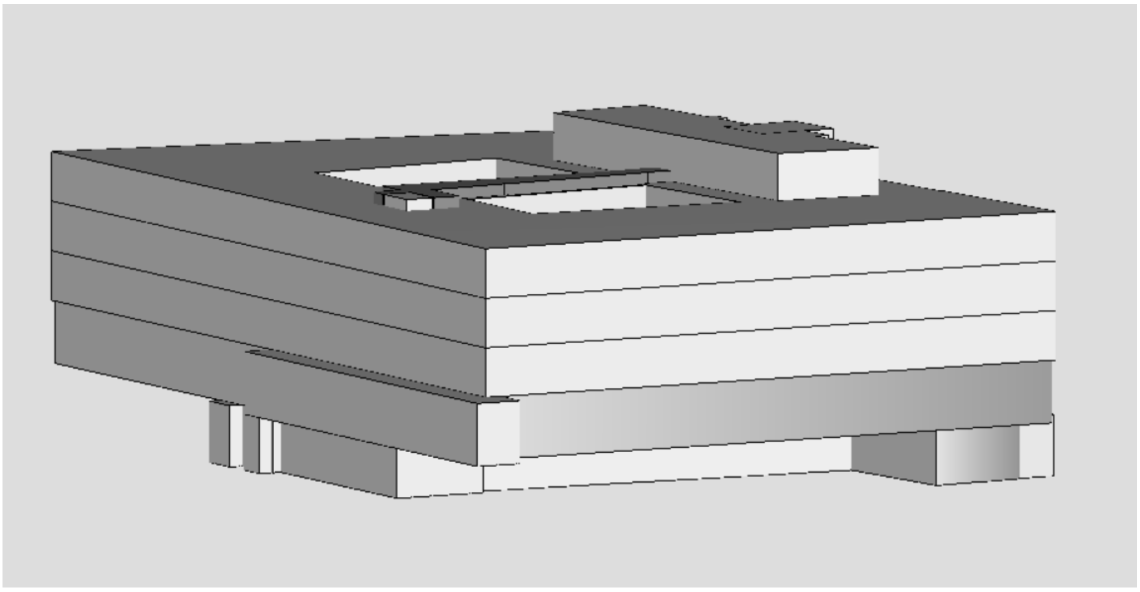


Figure 3.2: Created Location Prisms in SimpleBIM (JIRI HIETANEN, SAKARI LEHTINEN, 2019)

Afterwards the command-line application IfcConvert (IFCOPENSHELL CONTRIBUTORS, 2022) was applied to the respective Industry Foundation Classes (IFC) files to get Wavefront Object Format (OBJ) files which are needed for future simulations in HELIOS++ (WINIWARTER et al., 2022). This, as it is only supporting Geographic Tagged Image File Format (GeoTIFF), OBJ, XYZ Point Cloud Format (XYZ) and Voxel Format (VOX) files as scene inputs (see Figure 3.3). The geometry option `-j 8` and the serialization option `-use-element-guids` of the IfcConvert options were used. The first one was used for faster processing and the second one for future ML purposes. The element GUID - also called GlobalID (IfcGloballyUniqueID) - is an attribute of the common supertype Ifc.Root of all IFC entities (BUILDINGSMART CONTRIBUTORS, 2019a).

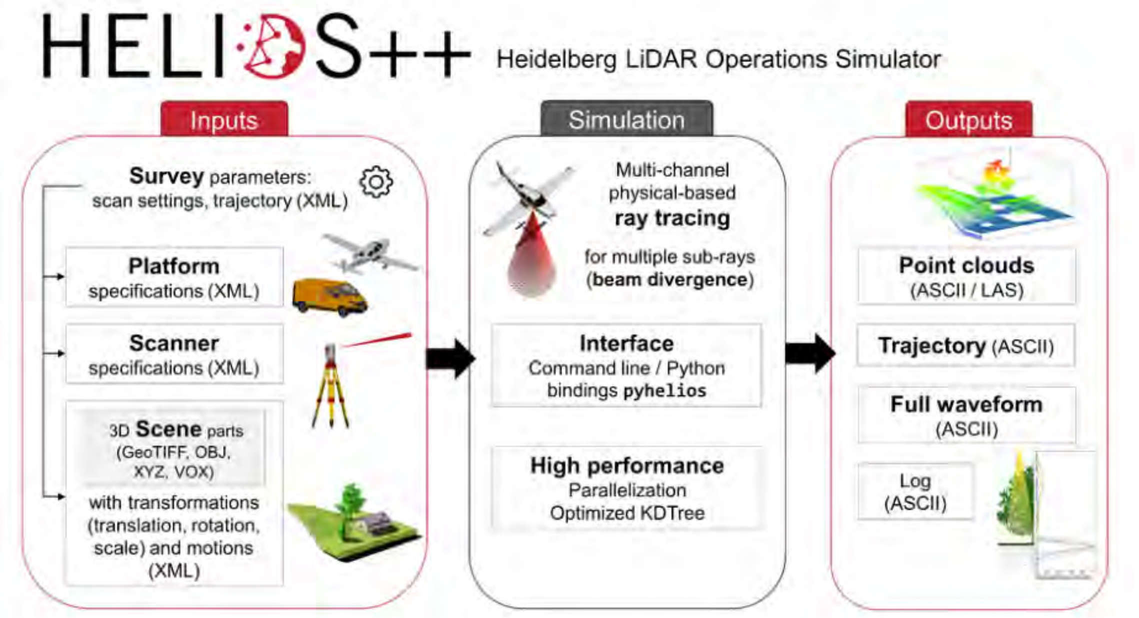

HELIOS++
Heidelberg LiDAR Operations Simulator
Inputs
Survey parameters: scan settings, trajectory (XML)
Platform specifications (XML)
Scanner specifications (XML)
3D Scene parts (GeoTIFF, OBJ, XYZ, VOX)
with transformations (translation, rotation, scale) and motions (XML)
Simulation
Multi-channel physical-based ray tracing
for multiple sub-rays (beam divergence)
Interface Command line / Python bindings pyhelios
High performance Parallelization Optimized KDTree
Outputs
Point clouds (ASCII / LAS)
Trajectory (ASCII)
Full waveform (ASCII)
Log (ASCII)

Figure 3.3: HELIOS++ software parts and simulation workflow (WINIWARTER et al., 2022)

Material Referencing

The resulting OBJ file, which comes together with a Material Library File (MTL), was manipulated for correctly referencing regarding individual purposes. In line 2 of Algorithm 3.1 the structure of the code which needs to be added is provided. For referencing the MTL, the keyword "mtllib" and the corresponding MTL name have to be written in the OBJ file in the second line. Additionally, the existing not manipulated code lines "usemtl surface-style..." which reference multiple times to the same material have to be replaced by a newly created and non-used material in the OBJ file. This was done by firstly executing Algorithm A.3 (added new material) and then Algorithm A.5 (replaced identical lines with references to new material).

Algorithm 3.1: Example of correct OBJ file beginning - added MTL referencing code line

```
1 # File generated by IfcOpenShell v0.7.0-476ab506d
2 mtllib Submodel_different_discipline_example1.mtl
3 g product-47b0ad3c-f62e-4e91-9239-b7543e4f15b4-body
4 s 1
5 v 2697483.60863716 5349159.6815045 470.909999999997
6 ...
```

To be able to do the preferred filtering of each different component in the PC later e.g., in CloudCompare (CC), a new different material with the code lines "helios_classification..." (see line 6, line 9, line 12 in Algorithm 3.2) and "newmtl..." (see line 8, line 11 in Algorithm 3.2) has to be added in the MTL (done by Algorithm A.1). The amount of additional new material depends on the total components existing in the OBJ file, i.e., the amount of new material is determined by the number of components of the OBJ file (done by

Algorithm A.2). This should be the case since when various components refer to the same material this could lead to problems when segmenting them e.g., in CC later.

Algorithm 3.2: Example of correct Material Library File (MTL) file snippet - added "helios_classification" and "newmtl..." lines

```
...
newmtl surface-style-191565-bim-referenzpunkt
Kd 0.313725490196078 0.313725490196078 0.313725490196078
Ks 0.5 0.5 0.5
Ns 64
helios_classification 18

newmtl surface-style-89575
helios_classification 19

newmtl surface-style-35831
helios_classification 20
...
```

Additionally, a group of components with multiple materials has to be changed to one component containing just one material to allow a better automatic filtering in CC (done by Algorithm A.6). To get a better understanding about it, the example of Valves from a provided Submodel_different_discipline_example2.ifc in Figure 3.5 should help.

Figure 3.4 shows the classification of IfcValve in the general Industry Foundation Classes (IFC) scheme.

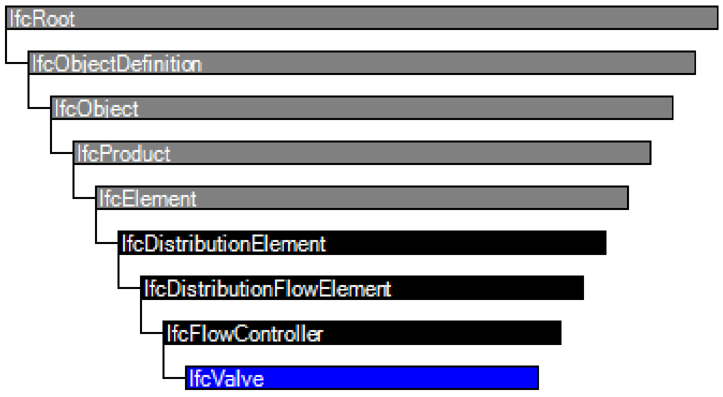
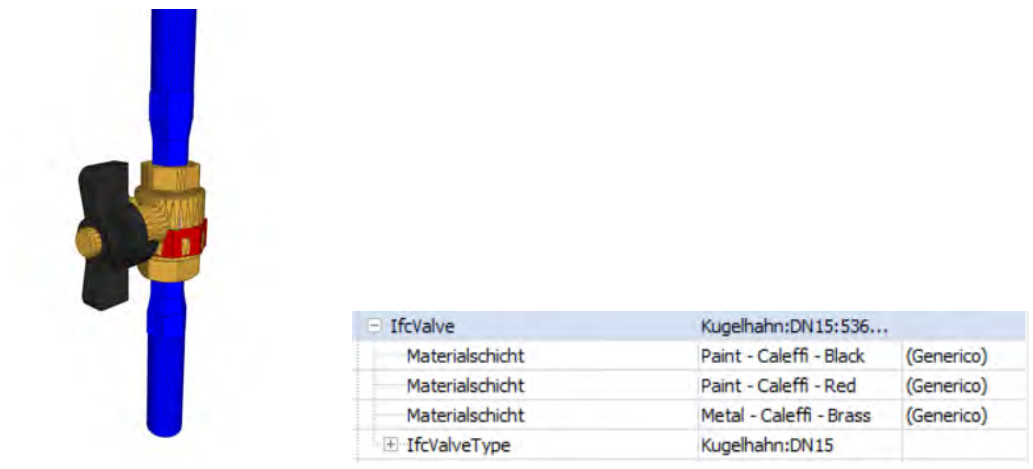


Figure 3.4: Entity inheritance of IfcValve in an IFC data model (BUILDINGSMART CONTRIBUTORS, 2019b)

The black-, red- and brass-colored surfaces of the Valve and the table next to it in Figure 3.5 represents the different materials. As material properties are being used for segmenting the components, the whole Valve must have only one material after converting it into the OBJ file and manipulating it.



IfcValve	Kugelhahn:DN15:536...	
Materialschicht	Paint - Caleffi - Black	(Generico)
Materialschicht	Paint - Caleffi - Red	(Generico)
Materialschicht	Metal - Caleffi - Brass	(Generico)
IfcValveType	Kugelhahn:DN15	

Figure 3.5: IfcValve visualization and entities in BIMvision (DATACOMP, n.d.)

For applying the correct material referencing to any OBJ and MTL file, the provided Python Files (PY) in the appendix must be executed as provided. In each script the input OBJ and MTL file have to be changed for the ones that want to be newly referenced, along with the value of the beginning of `helios_classification` of the new added material (see comment in Algorithm A.3).

1.Materialclasses_Assignments_mtl.py (Algorithm A.1)
2.Calculation_objmaterials_amount.py (Algorithm A.2)
3.Adding_material_mtl.py (Algorithm A.3)
4.Check_identical_or_multiple_used_Materials.py (Algorithm A.4)
5.Check_replace_newref.py (Algorithm A.5)
6.Check_identical_or_multiple_used_Materials.py (Algorithm A.4)
7.Reducing_materials_to_one.py (Algorithm A.6)

Setup XML files

In this section, the settings of each used Extensible Markup Language (XML) files for the Virtual Laser Scanning (VLS) regarding the preferred purposes are explained. Figure 3.6 shows the input options for each file in more detail. In general, to all XML files, state-of-the-art data information is applied for creating the realest environment conditions.

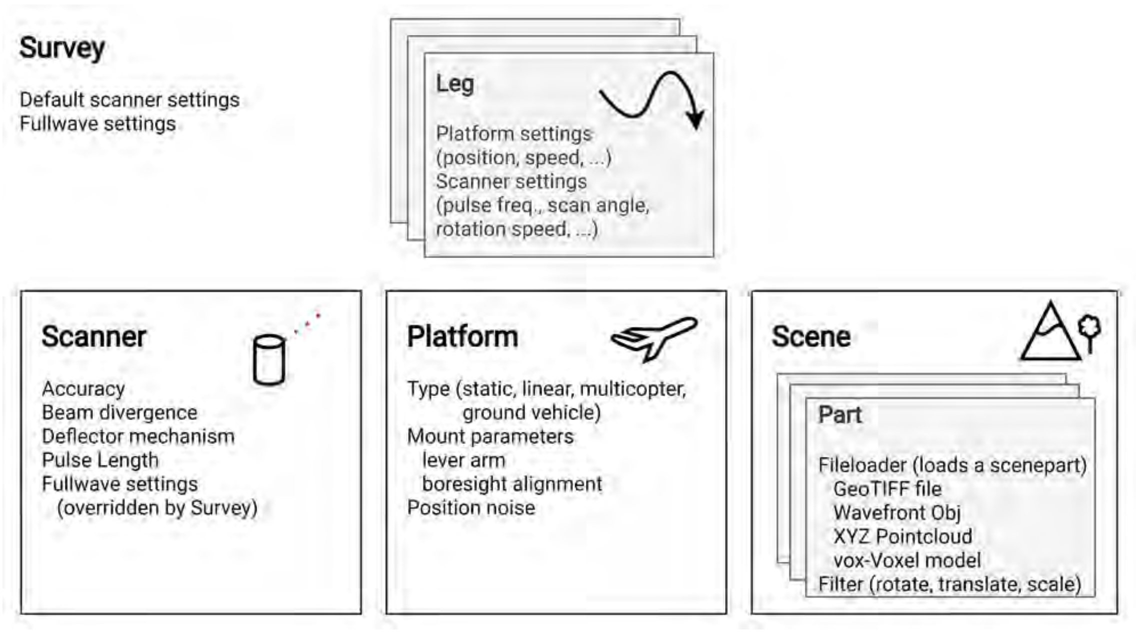


Figure 3.6: Main HELIOS++ XML files, their content, and their connection (WINIWARTER et al., 2022)

Survey XML Algorithm 3.3 describes the parameters and settings of an example Survey used for setting up a simulation. One Survey XML file was created for each storey, including many different waypoints. The waypoints were defined as realest as possible, e.g., going into each room and keeping the distances to walls or other instances high enough. In line 3 and line 4 the settings for the Scanner and the Platform are predefined for using certain waypoints or overall and can be seen as templates (see line 7 to line 10) for a distance from one waypoint to the next one. The waypoints were created by importing firstly the OBJ file of the respective storey in CC. Then each selected point was copied and pasted in a separate Text File (TXT).

Afterwards, a Python Script (see Algorithm B.1) was applied to transform the structure of the chosen waypoints into the structure needed for the Survey XML and copied into it. The `movePerSec_m` in line 3 has to be changed to a lower value for the final simulation to get a denser PC. E.g., the realistic scanning speed with the NavVis VLX 3 would be around $1\,\frac{\mathrm{m}}{\mathrm{s}}$ as this is the average walking speed. The `headRotatePerSec_deg` is set to 10 rps as the spin rate was given in 600 rpm in HESAI, 2023 on page 13 and the `scanFreq_hz` was set to 10. All these implementations can be found in line 4 of Algorithm 3.3.

The beam divergence was provided from HESAI after requesting with 0.136 deg*0.156 deg (H*V). For getting the final total `beamDivergence_rad`, both values needed to be firstly transferred into Radian (see Equation 3.1 and 3.2) and afterwards combined with the help of the Pythagoras Theorem (see Equation 3.3).

$$\Delta\theta_{\text{horizontal}} = 0.136\,\text{Grad} \cdot \left(\frac{\pi}{180}\right) \tag{3.1}$$

$$\Delta\theta_{\text{vertical}} = 0.156\,\text{Grad} \cdot \left(\frac{\pi}{180}\right) \tag{3.2}$$

$$\Delta\theta_{\text{total}} = \sqrt{\Delta\theta^2_{\text{horizontal}} + \Delta\theta^2_{\text{vertical}}} \tag{3.3}$$

As this is only for the half-angle, the values need to be doubled (see Equation 3.4). This leads to a final value of 0.007224229598 rad, which can be found in line 4 of Algorithm 3.3.

$$\Delta\theta_{\text{total,full}} = 2 \cdot \Delta\theta_{\text{total}} \tag{3.4}$$

Algorithm 3.3: Example of one Survey XML file snippet

```
1  <?xml version="1.0"?>
2  <document>
3      <platformSettings id="platform1" movePerSec_m="1" />
4      <scannerSettings id="scanner1" active="true" pulseFreq_hz="20000"
          scanFreq_hz="10" beamDivergence_rad="0.007224229598"
          headRotatePerSec_deg="3600" headRotateStart_deg="0" scanAngle_deg="
          20.15" trajectoryTimeInterval_s="0.01"/>
5      <survey name="tum_data_mls" platform="data/platforms.xml#navvis-vlx-3-
          on-person" scanner="data/scanners_als.xml#hesai_XT32M2X" scene="data
          /scenes/owndata/storeys/-1/storey_-1_scene.xml#tum_scene_-1">
6      <!-- platform: person, deflector: rotating -->
7          <leg>
8                      <platformSettings template="platform1" x="
                          2697480.750000" y="5349162.000000" z="470.910004"
                          /> <!-- with 1 m/s, we make it to the next point
                          in 3.640055 seconds -->
9                      <scannerSettings template="scanner1"
                          headRotateStop_deg="13104.197801"/> <!-- the
                          rotation will also take 3.640055 seconds -->
10             </leg>
11             <leg>
12                     <platformSettings template="platform1" x="
                          2697484.250000" y="5349163.000000" z="470.910004"
                          /> <!-- with 1 m/s, we make it to the next point
                          in 2.305838 seconds -->
13                     <scannerSettings template="scanner1"
                          headRotateStop_deg="8301.016736"/> <!-- the
                          rotation will also take 2.305838 seconds -->
14             </leg>
15     ...
16     </survey>
17 </document>
```

Scenes XML As the coordinate systems in HELIOS++ are defined as shown in Figure 3.7 below, the waypoints in Algorithm 3.3 of the trajectory line were set up referring to the coordinates of the OBJ coordinate system.

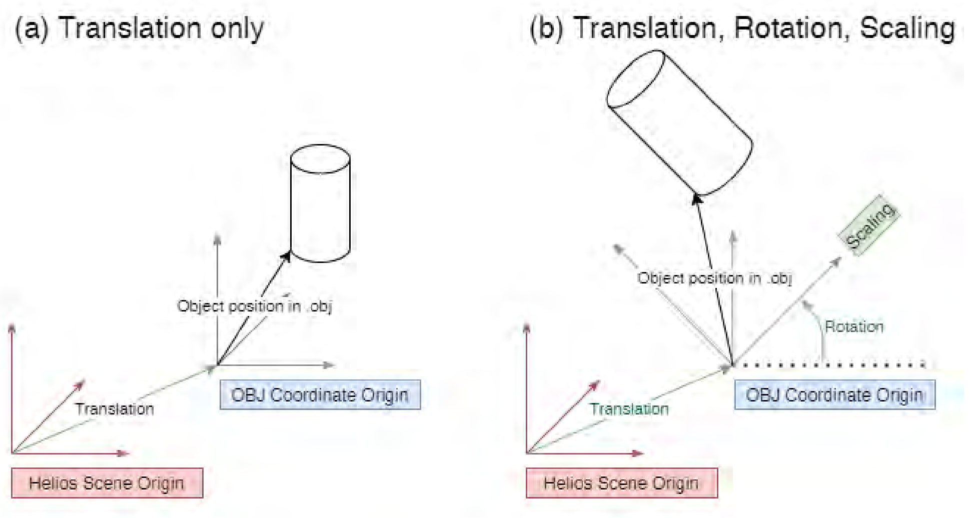


Figure 3.7: HELIOS++ scene, OBJ coordinate systems definitions and transformation filters (Winiwarter et al., 2022)

For getting the OBJ file as the environment for the simulations, the OBJ loader from HELIOS++ was applied in the scene XML (see line 6 in Algorithm 3.4). As for OBJ files the Material Library File (MTL) is automatically referenced, there was no need to further add the MTL.

Algorithm 3.4: Example of one Scene XML file snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <document>
3      <scene id="tum_scene_-1" name="TUM_-1_Scene">
4      ...
5          <part>
6              <filter type="objloader">
7                  <param type="string" key="filepath" value="data/sceneparts/
                       owndata/storeys/-1/SCG_mergedall(except_AR)_IFC4_(EDITED)
                       _-1_final_sim.obj" />
8              </filter>
9              <filter type="rotate">
10                 <param key="rotation" type="rotation">
11                     <rot angle_deg="0" axis="z"/>
12                 </param>
13             </filter>
14             <filter type="scale">
15                 <param type="double" key="scale" value="1" />
16             </filter>
17             <filter type="translate">
```

```
                <param type="integer" key="onGround" value="-1" />
                <param type="vec3" key="offset" value="0;0;0" />
            </filter>
        </part>
    </scene>
</document>
```

Platforms XML In Algorithm 3.5 the used platforms settings are provided. In line 1 the type of the scanning path is defined as a `linearpath`, i.e., the path from each waypoint to the next waypoint will be a straight line.

Algorithm 3.5: Example of one Platforms XML file snippet

```
1 <platform id="navvis-vlx-3-on-person" name="NavVis VLX 3 on person" type="
     linearpath">
2                 <!-- <scannerMount x="0" y="0" z="0">
3                         <rot axis="x" angle_deg="0" />
4                 </scannerMount> -->
5         ...
6 </platform>
```

The NavVis VLX 3 scanner is worn by a person (see Figure 3.8), so there are no further adjustments regarding the platform. Configurations for the two LiDAR scanners will be applied in the following paragraph 3.2.1.

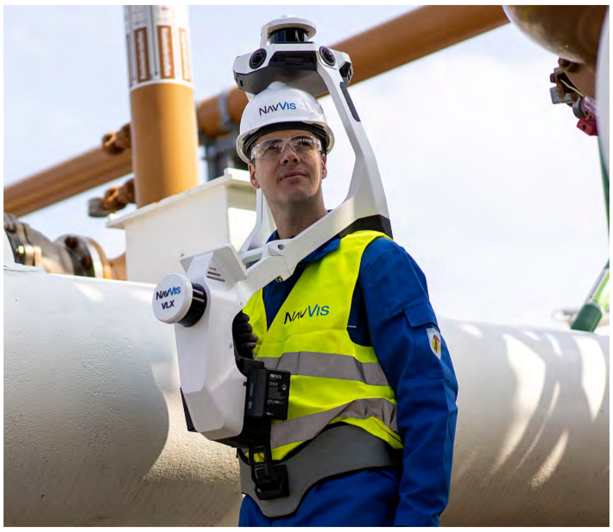


Figure 3.8: NavVis VLX 3 worn by a person (GoGeomatics Newsmakers, 2023)

Scanners XML For configuring the scanner settings as realistic as possible, the NavVis VLX 3 scanning device as a Multi-Scanner and Multi-Channel is taken (see Figure 3.8). It consists of two XT32M2X 32-Channel Medium-Range Mechanical LiDAR scanner from HESAI (Hesai, 2023). One on top of the device (scanning horizontally) and one in front of the person's body (scanning vertically). To set up the different locations of the two scanners correctly, it is important to understand how the coordinate system in HELIOS++ is defined by default (see Figure 3.9).

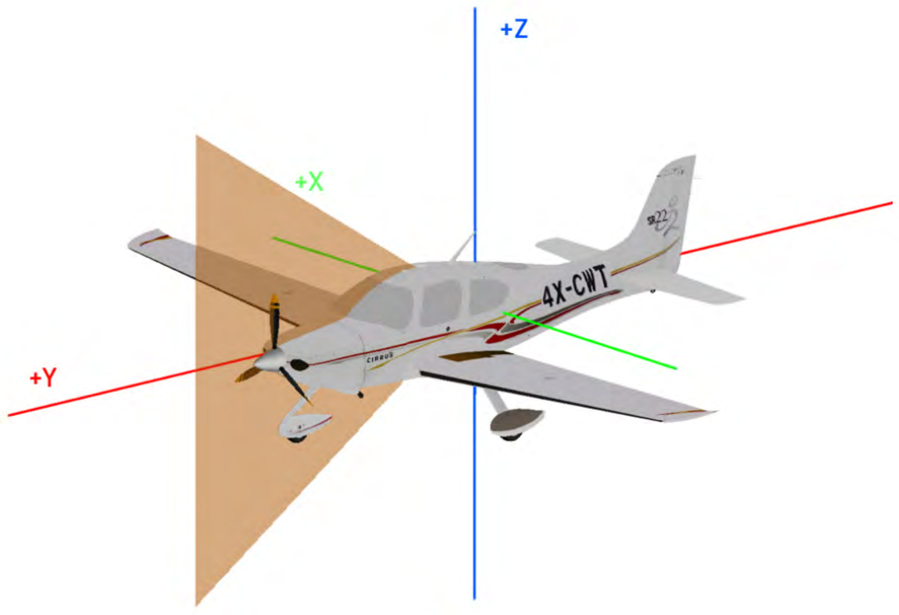


Figure 3.9: Default orientation and coordinate system of the laser beam in HELIOS++ (WINIWARTER et al., 2022)

As the adjustment of the differentiation of the laser emitter and laser receiver in HELIOS++ is not yet possible, both were placed with the offset (yellow dot in Figure 3.10) in the y+-axis and not in the x-axis for both scanners. The heights of the `beamorigin` were determined at $1.9\,\mathrm{m}$ for the horizontal scanner on top and $1.3\,\mathrm{m}$ for the one in front. The offset of $0.0305\,\mathrm{m}$ in the direction of y+-axis was added for the scanner in front of the body as well. Due to the rotated location of the vertical scanner in front it leads to a total z-coordinate of $1.3305\,\mathrm{m}$.

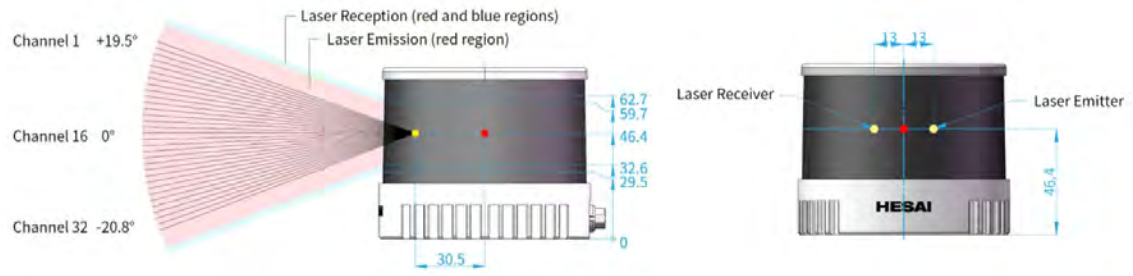
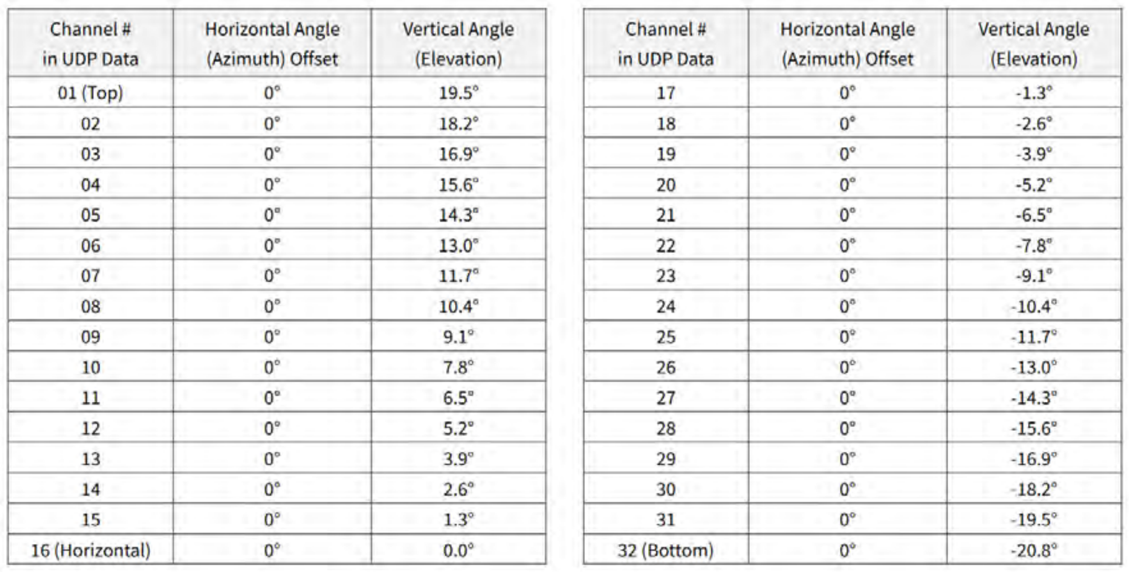


Figure 3.10: HESAI XT32M2X 32-Channel Medium-Range Mechanical LiDAR scanner (HESAI, 2023)

The angles of each channel are distributed in Figure 3.11. All the channels were transferred to the Scanners XML.

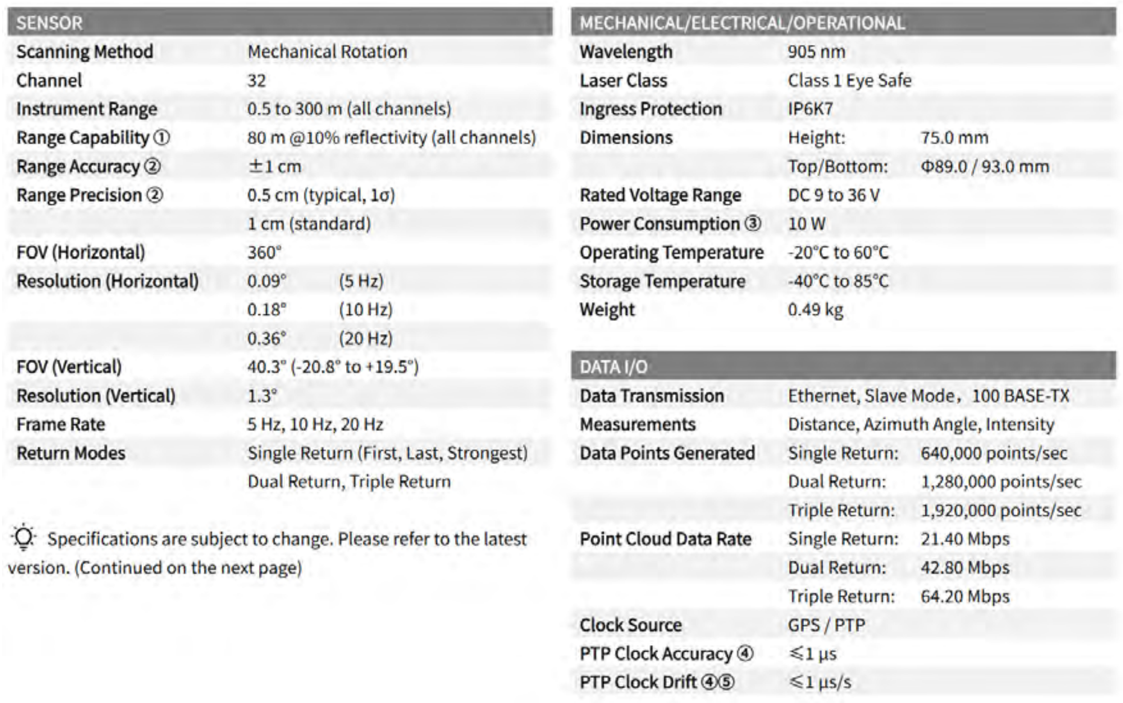


Channel # in UDP Data	Horizontal Angle (Azimuth) Offset	Vertical Angle (Elevation)
01 (Top)	0°	19.5°
02	0°	18.2°
03	0°	16.9°
04	0°	15.6°
05	0°	14.3°
06	0°	13.0°
07	0°	11.7°
08	0°	10.4°
09	0°	9.1°
10	0°	7.8°
11	0°	6.5°
12	0°	5.2°
13	0°	3.9°
14	0°	2.6°
15	0°	1.3°
16 (Horizontal)	0°	0.0°

Channel # in UDP Data	Horizontal Angle (Azimuth) Offset	Vertical Angle (Elevation)
17	0°	-1.3°
18	0°	-2.6°
19	0°	-3.9°
20	0°	-5.2°
21	0°	-6.5°
22	0°	-7.8°
23	0°	-9.1°
24	0°	-10.4°
25	0°	-11.7°
26	0°	-13.0°
27	0°	-14.3°
28	0°	-15.6°
29	0°	-16.9°
30	0°	-18.2°
31	0°	-19.5°
32 (Bottom)	0°	-20.8°

Figure 3.11: Detailed Channel Distribution of HESAI XT32M2X 32-Channel Medium-Range Mechanical LiDAR scanner (Hesai, 2023)

For the vertical scanner the positive and negative signs are changed due to the rotated position. More concrete specifications were taken from the table (see Figure 3.12) and transferred as well. The `pulseFreqs_hz` is set to 20.000 points per second (Single Return).



SENSOR		
Scanning Method	Mechanical Rotation	
Channel	32	
Instrument Range	0.5 to 300 m (all channels)	
Range Capability ①	80 m @10% reflectivity (all channels)	
Range Accuracy ②	±1 cm	
Range Precision ②	0.5 cm (typical, 1σ)	
	1 cm (standard)	
FOV (Horizontal)	360°	
Resolution (Horizontal)	0.09°	(5 Hz)
	0.18°	(10 Hz)
	0.36°	(20 Hz)
FOV (Vertical)	40.3° (-20.8° to +19.5°)	
Resolution (Vertical)	1.3°	
Frame Rate	5 Hz, 10 Hz, 20 Hz	
Return Modes	Single Return (First, Last, Strongest)	
	Dual Return, Triple Return	

Specifications are subject to change. Please refer to the latest version. (Continued on the next page)

MECHANICAL/ELECTRICAL/OPERATIONAL		
Wavelength	905 nm	
Laser Class	Class 1 Eye Safe	
Ingress Protection	IP6K7	
Dimensions	Height:	75.0 mm
	Top/Bottom:	Φ89.0 / 93.0 mm
Rated Voltage Range	DC 9 to 36 V	
Power Consumption ③	10 W	
Operating Temperature	-20°C to 60°C	
Storage Temperature	-40°C to 85°C	
Weight	0.49 kg	

DATA I/O		
Data Transmission	Ethernet, Slave Mode, 100 BASE-TX	
Measurements	Distance, Azimuth Angle, Intensity	
Data Points Generated	Single Return:	640,000 points/sec
	Dual Return:	1,280,000 points/sec
	Triple Return:	1,920,000 points/sec
Point Cloud Data Rate	Single Return:	21.40 Mbps
	Dual Return:	42.80 Mbps
	Triple Return:	64.20 Mbps
Clock Source	GPS / PTP	
PTP Clock Accuracy ④	≤1 μs	
PTP Clock Drift ④⑤	≤1 μs/s	

Figure 3.12: HESAI XT32M2X 32-Channel Medium-Range Mechanical LiDAR scanner specifications (Hesai, 2023)

The specifications and a snippet of a few channels implemented can be found in Algorithm 3.6. it is important to mention here line 3 in Algorithm 3.6, as because the `accuracy_m` has to get set to 0.0 instead of the actual specification of the LiDAR sensors of 0.01 to avoid noisy instance PC which should be further used for the training purposes in the ML methods.

Algorithm 3.6: Scanners XML file snippet

```
<!-- ##### BEGIN Hesai XT32M2X ##### -->
<scanner id         = "hesai_XT32M2X"
           accuracy_m     = "0.0"
           name     = "Hesai XT32M2X 32-Channel Medium-Range Mechanical
              Lidar"
           optics     = "rotating"
           pulseLength_ns     = "30"
           rangeMin_m     = "0.05"
           rangeMax_m = "300"
           pulseFreqs_Hz     = "20000"
           scanAngleMax_deg     = "1"
           scanAngleEffectiveMax_deg = "1"
           scanFreqMin_Hz     = "0"
           scanFreqMax_Hz     = "0"
           wavelength_nm     = "905"
           maxNOR      = "1"
                      headRotatePerSecMax_deg  = "7200">   <!-- added
                         values from NavVis' colleagues information and
                         Hesai Lidar Sensor data sheet -->

       <FWFSettings beamSampleQuality="1"/> <!-- set to one for fast
           simulations -->
       <channels>
       <!-- BEGIN channels on scanner on top -->
              <channel id="0">
                     <beamOrigin x="0" y="0.0305" z="1.9">
                            <rot axis="y" angle_deg="90" />
                            <rot axis="x" angle_deg="19.5" />
                     </beamOrigin>
                     <headRotateAxis x="0" y="0" z="1"/>
              </channel>
       <channel id="1">
                     <beamOrigin x="0" y="0.0305" z="1.9">
                            <rot axis="y" angle_deg="90" />
                            <rot axis="x" angle_deg="18.2" />
                     </beamOrigin>
                     <headRotateAxis x="0" y="0" z="1"/>
              </channel>
    ...
    </channels>
```

```
</scanner>
```

Below a test simulation of the NavVis VLX configuration can be seen in a small box in HELIOS++ after one full rotation of both LiDAR sensors (See Figure 3.13).

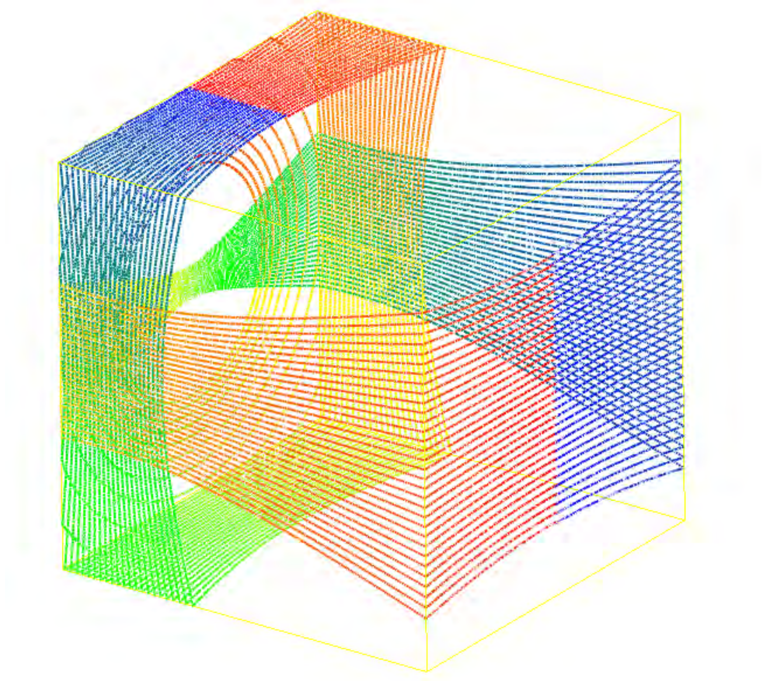
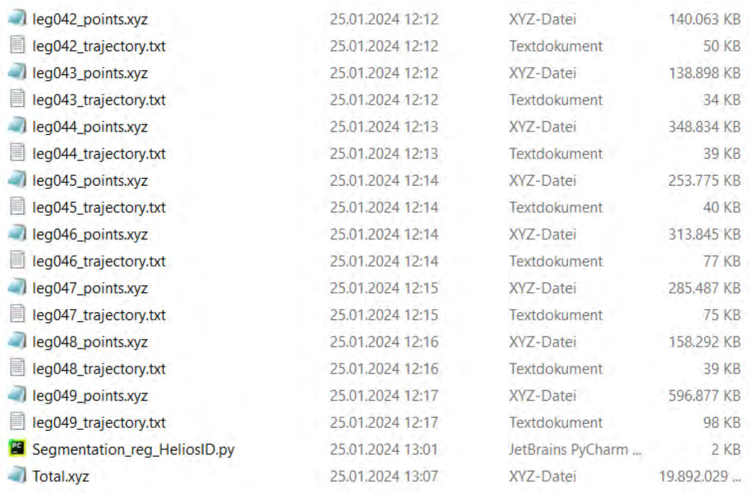


Figure 3.13: Example virtual Scan of the NavVis VLX in a box via HELIOS++

Point Cloud Instances Creation

After setting up all the previously mentioned steps and running the simulation successfully, HELIOS++ will output separated XYZ files for each leg of the simulation. The following python scripts were executed to get the segmented and meshed incomplete PC instance as PLY files (Paul Bourke, 2024) in the local coordinate system:

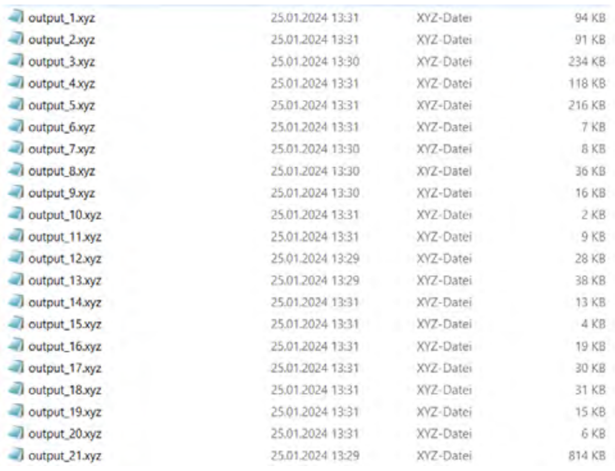
1.Example_merging_xyz_files.py (Algorithm C.1)



leg042_points.xyz	25.01.2024 12:12	XYZ-Datei	140.063 KB
leg042_trajectory.txt	25.01.2024 12:12	Textdokument	50 KB
leg043_points.xyz	25.01.2024 12:12	XYZ-Datei	138.898 KB
leg043_trajectory.txt	25.01.2024 12:12	Textdokument	34 KB
leg044_points.xyz	25.01.2024 12:13	XYZ-Datei	348.834 KB
leg044_trajectory.txt	25.01.2024 12:13	Textdokument	39 KB
leg045_points.xyz	25.01.2024 12:14	XYZ-Datei	253.775 KB
leg045_trajectory.txt	25.01.2024 12:14	Textdokument	40 KB
leg046_points.xyz	25.01.2024 12:14	XYZ-Datei	313.845 KB
leg046_trajectory.txt	25.01.2024 12:14	Textdokument	77 KB
leg047_points.xyz	25.01.2024 12:15	XYZ-Datei	285.487 KB
leg047_trajectory.txt	25.01.2024 12:15	Textdokument	75 KB
leg048_points.xyz	25.01.2024 12:16	XYZ-Datei	158.292 KB
leg048_trajectory.txt	25.01.2024 12:16	Textdokument	39 KB
leg049_points.xyz	25.01.2024 12:17	XYZ-Datei	596.877 KB
leg049_trajectory.txt	25.01.2024 12:17	Textdokument	98 KB
Segmentation_reg_HeliosID.py	25.01.2024 13:01	JetBrains PyCharm ...	2 KB
Total.xyz	25.01.2024 13:07	XYZ-Datei	19.892.029 ...

Figure 3.14: Different XYZ files for each leg with merged and created "Total.xyz" file

2.Segmentation_reg_HeliosID.py (Algorithm C.2)



output_1.xyz	25.01.2024 13:31	XYZ-Datei	94 KB
output_2.xyz	25.01.2024 13:31	XYZ-Datei	91 KB
output_3.xyz	25.01.2024 13:30	XYZ-Datei	234 KB
output_4.xyz	25.01.2024 13:31	XYZ-Datei	118 KB
output_5.xyz	25.01.2024 13:31	XYZ-Datei	216 KB
output_6.xyz	25.01.2024 13:31	XYZ-Datei	7 KB
output_7.xyz	25.01.2024 13:30	XYZ-Datei	8 KB
output_8.xyz	25.01.2024 13:30	XYZ-Datei	36 KB
output_9.xyz	25.01.2024 13:30	XYZ-Datei	16 KB
output_10.xyz	25.01.2024 13:31	XYZ-Datei	2 KB
output_11.xyz	25.01.2024 13:31	XYZ-Datei	9 KB
output_12.xyz	25.01.2024 13:29	XYZ-Datei	28 KB
output_13.xyz	25.01.2024 13:29	XYZ-Datei	38 KB
output_14.xyz	25.01.2024 13:31	XYZ-Datei	13 KB
output_15.xyz	25.01.2024 13:31	XYZ-Datei	4 KB
output_16.xyz	25.01.2024 13:31	XYZ-Datei	19 KB
output_17.xyz	25.01.2024 13:31	XYZ-Datei	30 KB
output_18.xyz	25.01.2024 13:31	XYZ-Datei	31 KB
output_19.xyz	25.01.2024 13:31	XYZ-Datei	15 KB
output_20.xyz	25.01.2024 13:31	XYZ-Datei	6 KB
output_21.xyz	25.01.2024 13:29	XYZ-Datei	814 KB

Figure 3.15: Different XYZ files segmented based on their HeliosID (HeliosID is included and every filename

3.Origin_to_Local_COS.py (Algorithm C.6)

This step was done to reduce the size of the coordinates from each file to its local coordinates again. Without this step, it would have not been possible to open the files in e.g., the software MeshLab. The values used for globally shifting back the instances to its local coordinates are the values suggested from CloudCompare (CC) when the OBJ files were initially imported.

4.Convert_to_PLY.py

For this step Open3D (Zhou et al., 2018) was used (Algorithm C.7).
After this step was done, the resulted PLY files contained only their respective X, Y and Z coordinates. The created PLY files with their content were ready to use for the Point Completion Network (PCN).

5.Surface_reconstruction_ball_pivoting_obj.py

For this step Bernardini et al. (1999) was used (Algorithm C.3)

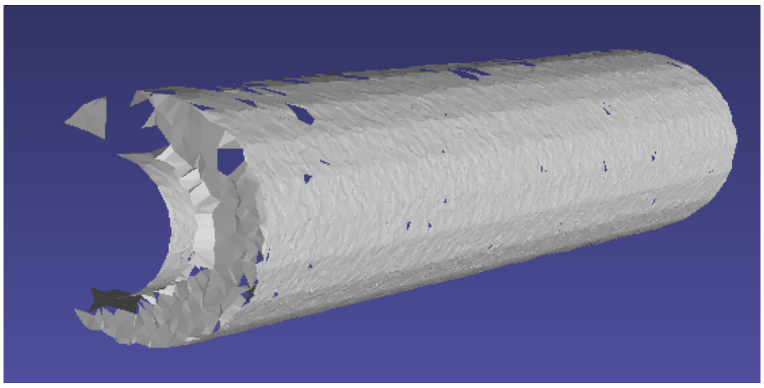


Figure 3.16: Example OBJ file of a partial DuctSilencer instance after ball pivoting surface reconstruction

Finally, mapping tables for each storey were created by a PY Script (see Algorithm C.4) to allow, if necessary, to add all previous information to each meshed instance again (see Algorithm C.5). The information was not added, as it is not needed for the further application of the ML methods. Nonetheless, it is important to point out that it can be added in the first line of all OBJ instances by using the possibility of an OBJ file to have multiple group names (see sections "Grouping" and "Syntax" in Paul Bourke, 2023). A code snippet of a new mapped OBJ file can be seen in Algorithm 3.7 and the respective new added code in line 1.

Algorithm 3.7: Example of a meshed and new mapped OBJ file beginning of an incomplete PC instance - added grouping code line

```
1 g 13657.0 02n_cFnRX8bfT2cgeHGzGH surface-style-13597 711277 IfcPipeSegment
    Rohrtypen:Geberit PE-HD (DN Größe):4237303 Rohrtypen:Geberit PE-HD (DN
    Größe)
2 v 2697484.9413 5349163.4302 474.4699 6845059.9565 0.0000 1 1 225 0 13657
    414863.000350000
3 v 2697484.9129 5349163.4279 474.4749 35120985.7638 0.0000 1 1 233 0 13657
    414863.000362500
4 v 2697484.9155 5349163.427 474.4718 35006388.7353 0.0000 1 1 231 0 13657
    414863.000359375
```

```
v 2697484.8967 5349163.4251 474.4745 45815367.3070 0.0000 1 1 236 0 13657
    414863.000367187
...
```

3.2.2 Using Real-World Data

The public available Point Cloud (PC) dataset captured by the NavVis VLX3 on this website (Source) was received after filling out and sending the request formular. Afterwards, the PC was segmented with Aurivus (Stefan Hörmann, Martin Bach, 2019) with regards to Mechanical, Electrical and Plumbing (MEP) instances. As an output an .aurivus complete segmented PC file and separated LiDAR Point Cloud Data Format (E57) PCs including e.g., all Pipes/ Pipe_Fittings were obtained to be imported to CloudCompare (CC). CC, has a function named "Label Connected Components" that allows the segmentation of all different PCs single instances. It was identified that such option is not working properly as it cannot fully separate the multiple single instances, e.g., instead of individually segmenting, it combines different instances - with different characteristics - as if they were the same. The reason for this is that this function is used to assign labels to groups of connected points based on their spatial proximity, which results in labeling clusters of points from different overlapping single instances.

Additionally, it is important to note that, in general, the .aurivus files do not provide more semantic information to all single instances (except the category "Pipes") in comparison to the IFC model used for the virtual simulation. Furthermore, there are not corresponding Ground Truths for the real-world partial PC instances. That means that the assigning of every instance to a specific instance category and the usage of this real-world data for either, the PC or Voxel-DCGAN, was not possible. Still, it is probable that the segmentation step for the single instances is more feasible to perform using the programming language PY than in CC as it was already done with the output data in human readable formats from the virtual simulations. Nonetheless, the fact that the real-world data is in non-human readable formats such as E57 or .aurivus can still present problems. Therefor, the data generation was focused on mainly using simulated data.

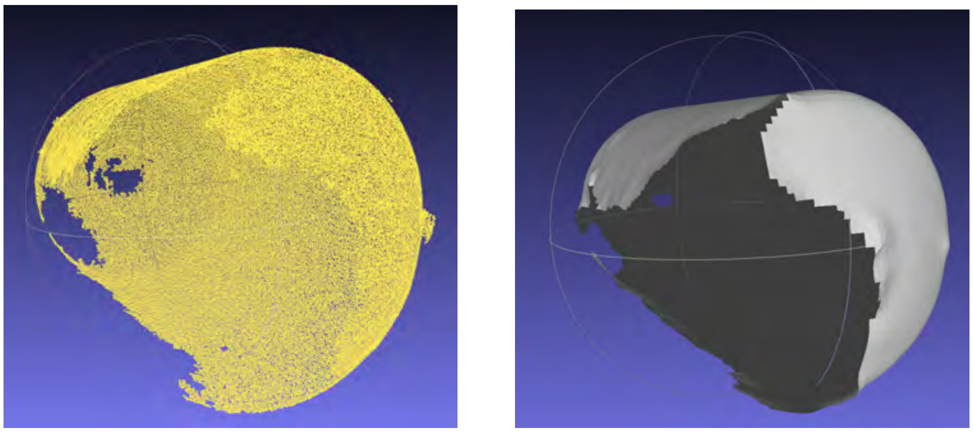

(a) (b)

Figure 3.17: **Real-World PC and Mesh captured by NavVis VLX3.** (a) Real-World PC captured by NavVis VLX3 and segmented by Aurivus. (b) Real-World meshed (Surface Reconstruction VCG) PC captured by NavVis VLX3

3.2.3 Ground Truth Creation

The Ground Truths used for the Voxel-DCGAN and the PCN were created by doing the following steps (refer to Figure 3.1 for more detailed graphical explanations regarding each step):

1.Importing the total OBJ file for one storey into CloudCompare (CC).
2.Marking all submeshes named by their GUID and sampling them with 16384 points.
3.Marking all created instances PCs from Top-to-Bottom and exporting them as XYZ/ ascii files.
4.Transforming them to local coordinates (Algorithm C.6).
5.Converting them to PLY files (Algorithm C.7).
6.Performing screened poisson surface reconstruction and saving as OBJ files (Algorithm B.4).
7.Sampling with 16384 points and converting them to PLY files (Algorithm B.3).
8.Zero-centering and normalizing the files (Algorithm B.2).
(9.Renaming the new files in relation to the new storey to e.g. "output_2nd_234")

The used Ground Truths for the Voxel-DCGAN were created after step six (6) as the OBJ format was needed for this method. On the other hand, they were created for the PCN after all steps as the PLY format is needed. Examples of one instance Ground Truth for the PCN and Voxel-DCGAN can be seen in Figure 3.18.

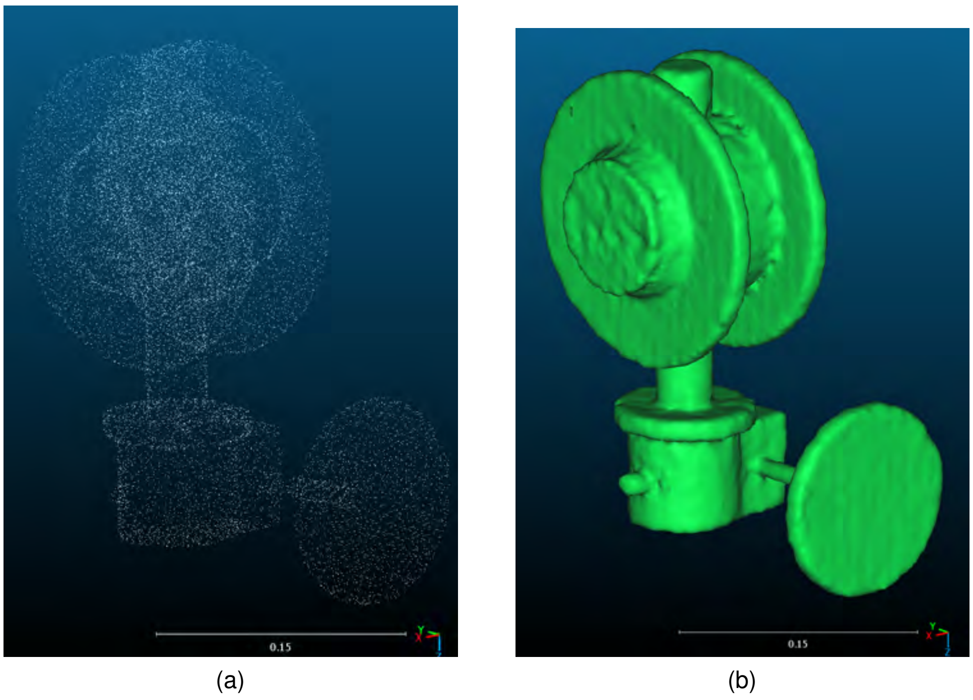


Figure 3.18: **Examples of instance Ground Truth used for the PCN and Voxel-DCGAN**. (a) Example of instance Ground Truth used for PCN. (b) Example of instance Ground Truth used for Voxel-DCGAN

3.3 Point Cloud completion Methods

This chapter contains the application of the PCN (YUAN et al., 2019) (Source Code) and the Voxel-DCGAN (Source Code) on my own dataset. Both methods are applied on certain categories (e.g., SpaceHeater and DuctSilencer) from the dataset of storey -1. The dataset for storey -1 contains in total 16345 complete PCs and 10676 partial PCs. The total amount of complete Mechanical, Electrical and Plumbing (MEP) point clouds is 15735 and the amount of partial MEP point clouds is 10112. For each category the amount of usable data differs due to the limited size of the dataset. Therefore, a few example categories were chosen (SpaceHeater and DuctSilencer) to showcase the application of both methods on my own data in a simplified way. Notebooks in Google Colab Pro+, Google Drive, Github and GitLab were used to set up the code settings and to collaborate. Additionally, PyCharm was used on my local machine for code debugging.

3.3.1 Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN)

The segmented Ground Truth OBJ files mentioned in subsubsection 3.2.3 for the categories SpaceHeater and DuctSilencer were firstly voxelized with Binvox (MIN, 2004);(NOORUDDIN & TURK, 2003). An example of one Ground Truth OBJ file and the corresponding .binvox file of the DuctSilencer category is shown in the following Figure 3.19.

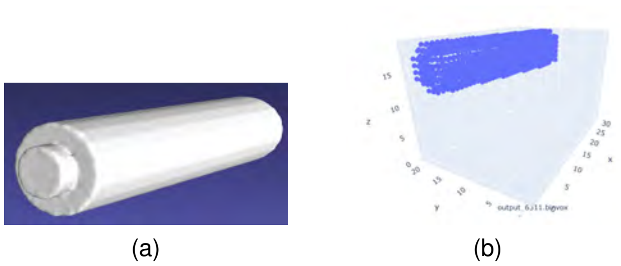


Figure 3.19: **Example of one Ground Truth OBJ file and the corresponding .binvox file of the DuctSilencer category**. (a) OBJ file and (b) .binvox file.

The output files of this method were .binvox files and no solid voxelization was used as many instances like pipes have to be hollow in the interior.

3.3.2 Point Completion Network (PCN)

The preparation of the dataset to apply the PCN correctly is important. On the one hand, the folder structure was set up as it is arranged in the initial example data and the data was segmented (i.e., containing single instance), zero-centered and normalized in the unit cube. For sorting the data to each category, Algorithm B.5 in combination with the storey mapping table were used. The MEP folders were already created, so that files related to non-MEP components were skipped and not copied. The respective .list files, including the sorting of the files were generated in an automized way. The data splitting into Training, Validating and Testing data was done for each category with a ratio of approx. 80%, 10% and 10% respectively. The amount of partial PCs, for each complete PC and individual category, differs as the amount of data in my used dataset is limited. But in general, a minimum two partial PCs for each complete PC was taken for the training. The segmentation was already explained in the previous chapter and the zero-centering and normalizing of all PLY files were done by using Algorithm C.7. For applying the PCN no voxelization is needed and its design allows it to be applied on raw/ incomplete PCs which saves memory costs, as fewer geometric information is lost to produce high-grained completed PCs.

Chapter 4

Experimental Evaluation

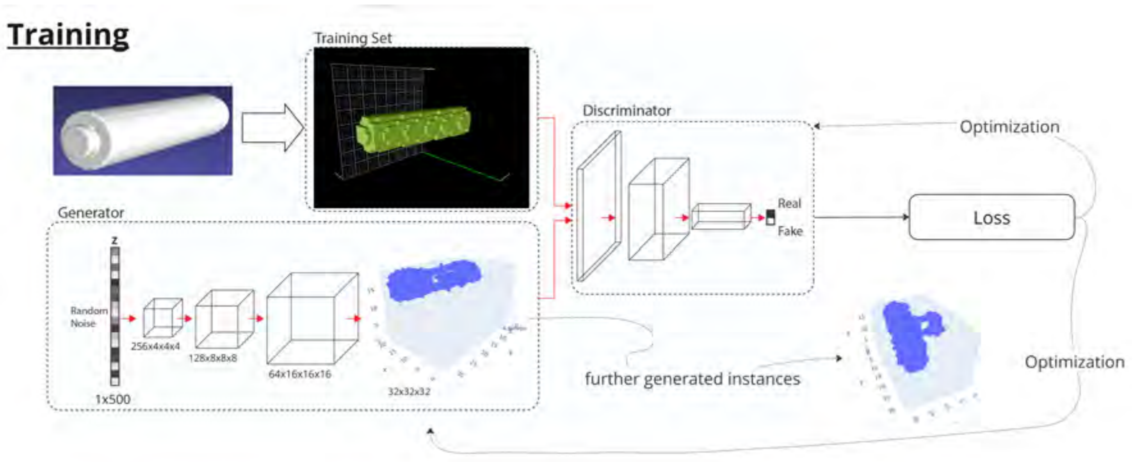
4.1 Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN) Simulation Set-up

In a Machine Learning (ML) context, the parameters listed below offer insights into my used training regimen. The learning rate of 2e-4 signifies the step size taken during the optimization process, dictating how much model parameters are adjusted with each iteration to minimize error. With 50 epochs, the model undergoes 50 complete passes over the training dataset. The batch size, reduced from 64 to 5, determines the number of samples processed before updating the model's weights, affecting training efficiency and memory usage. A noise vector size (nz) of 500 implies the dimensionality of random noise added to the input, crucial for generating diverse outputs in generative models. Additionally, a scale factor of 4 (nsf) indicates the degree of downscaling applied to the encoded voxel size, potentially influencing the granularity and detail of the model's representations.

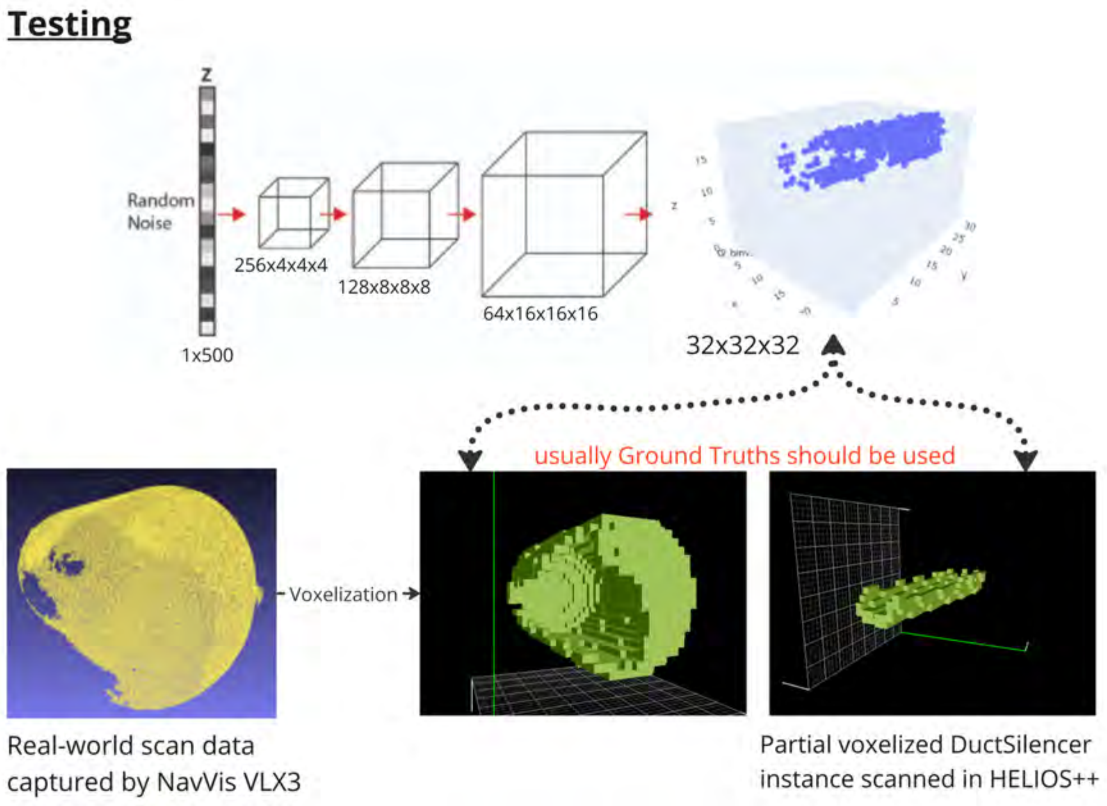
The following configuration was used:

- Learning rate = 2e-4
- Epochs = 50
- Batch size = 5 (was initially set to 64)
- nz = 500 (noise vector size)
- nsf = 4 (encoded voxel size, scale factor)

4.1.1 Training, Validating and Testing



(a)


Testing
Random Noise
256x4x4x4
128x8x8x8
64x16x16x16
1x500
32x32x32
usually Ground Truths should be used
Voxelization
Real-world scan data captured by NavVis VLX3
Partial voxelized DuctSilencer instance scanned in HELIOS++

(b)

Figure 4.1: **Point cloud completion - Voxel-DCGAN Method Overview.** (a) Training and Validating process. (b) Testing process.

To train the Voxel-DCGAN, the actual complete real 3D .binvox files (Ground Truths) of one category were taken to create most similar and indistinguishable similar instances of this category (see Figure 4.1 (a)). This implies that for applying the Voxel-DCGAN the generated partial PCs were not used. All of the complete .binvox files of the respective category were used, employing the whole entirety of the complete data that is available in my dataset.

In the repository (Source Code) used, no specific validation process is documented or used in the code. Normally, validation should be performed after each epoch, as it is done for the PCN with an unseen dataset to monitor the performance of the model on the unseen dataset. The validation step was here implemented in the "train.py" with unseen data. The implemented validating loop is similar to the training loop, but operates on the validation dataset. It computes the average loss over all batches of the validation dataset. This helps monitor how well the model is generalizing to unseen data.

As well as for the validation step, no specific testing process is documented or used in the code. Usually, the testing is done on an additional unseen dataset during training and validation. This step was here implemented in the "train.py" after the completing the training. For the testing, on one hand partial real-world PCs captured by the NavVis VLX3 and other partial PCs from the HELIOS++ simulations were used (see Figure 4.1 (b)). But, for correctly applying this testing step, usually their Ground Truths should be used. Ideally, the completion of partial real-world PC instances should be done with semantic instance-matching when the Voxel-DCGAN is sufficiently trained for every partial specific instance by the use of their Ground Truths.

4.2 Point Completion Network (PCN) Simulation Set-up

4.2.1 Hyperparameter selection, training and validation

The training was done separately for each category as the number of partial Point Clouds (PCs) for each complete PC differs. Training with regards to the loss functions Chamfer Distance (CD) and Earth Mover's Distance (EMD) was done as well. The following parameters were used:

- Learning rate (lr) = 0.001
- Epochs (epochs) = 400
- Batch size for data loader (batch_size) = 64
- Number of workers for data loader (num_workers) = 8
- Logger frequency in every epoch (log_frequency) = 1

Validating was done after each epoch. After all epochs were finished, the model gave the best L1 CD values which was further used as pre-trained model. Subsequently, the testing was done separately for all categories. Testing with regards to the loss functions CD and EMD was done as well. One tag experiment was chosen to store all the results of the testing. The model path of the best pre-trained model was taken. Testing was also done by using noisy partial PCs as input of my simulation results. The following parameters were used:

- Batch size for data loader (batch_size) = 64
- Number of workers for data loader (num_workers) = 8

4.3 Comparison of Method’s Results

In this chapter, the Point Completion Network (PCN) and the Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN) were quantitatively and qualitatively evaluated. For both methods, the main reasons causing the erroneous values in the parameters are on one hand, the limitation of amount of data used and on the other, the different initial positions of the instances.

4.3.1 Quantative Metrics

Losses & F1-Score

Point Completion Network (PCN):
Chamfer Distance (CD), Earth Mover’s Distance (EMD) losses and F1-Score:

DuctSilencer category:
Training: The best L1 CD model is in epoch 380, the minimum L1 CD is 27.59388089179. The best EMD model is in epoch 366, the minimum EMD is 26.620883494615555.

Testing on accurate Point Clouds (PCs):

L1_CD(1e-3)	L2_CD(1e-4)	EMD(1e-3)	FScore-0.01(%)
24.0351	17.0883	39.1814	22.0874

In SpaceHeater category:
Training: The best L1 CD model is in epoch 261, the minimum L1 CD is 41.3503855466842. The best EMD model is in epoch 63, the minimum EMD is 60.65165996551514.

Testing with accurate PCs:

L1_CD(1e-3)	L2_CD(1e-4)	EMD(1e-3)	FScore-0.01(%)
89.9538	365.3143	594.6836	4.2386

The L1_CD and L2_CD metrics have different values, because they serve distinct purposes in evaluating the dissimilarity between two PCs. L1_CD, also referred to as Manhattan distance, calculates the absolute differences between corresponding points’ coordinates and sums them up. This metric is resilient to outliers and disparities in scales among coordinates. It provides equal weight to discrepancies across all dimensions, making it suitable for scenarios where uniform sensitivity to differences is desired. Conversely, L2_CD, or Euclidean distance, computes the square root of the sum of squares of coordinate differences. It is more sensitive to outliers and discrepancies in scales, thus emphasizing larger differences in distances.

The smaller L1_CD, L2_CD and EMD values for the DuctSilencer category in comparison to the SpaceHeater ones probably appeared, because firstly more partial PCs were available for each orientated instance. Additionally, the F1-Score indicates a better balance between precision and recall for the DuctSilencer category. The F1-score ranges from 0 to 1, where 1 is the best possible score. It is a measure of a model's ability to provide a good trade-off between precision (the ability to avoid false positives) and recall (the ability to find all true positives). A higher F1-score indicates a better balance between precision and recall. Still, with more data those values can be further improved.

Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN):
Discriminator and Generator losses:

DuctSilencer category:
In epoch 30 the lowest discriminator loss takes 0.51365489 and the generator loss takes 7.58635521. The generator loss is mainly varying between 5-10. The discriminator loss decreased from 25 to almost 0.

SpaceHeater category:
In epoch 50 the lowest discriminator loss takes 4.23658419 and the generator loss takes 6.10359430. The generator loss is mainly varying between 5-8. The discriminator loss decreased linearly from 25 to 5. No new output files were generated for this category.

The varying of the generator loss on a more or less constant value indicates a saturated learning. That means, it is possible that the generator has reached a point where further improvement is challenging. This could happen when the model has learned to generate samples that are realistic but might not capture additional nuances or details. In such cases, the generator loss may stabilize around a certain value. The generator loss fluctuated initially as the generator learns to produce samples that can fool the discriminator. However, over time, it should decrease as the generator improves its ability to generate realistic samples. The fact that the generator loss for both categories remains too high suggests that the generator is struggling to generate realistic samples, indicating a need for adjustments in the training process or architecture. The lower values of the discriminator loss, especially for the DuctSilencer category, indicate that the discriminator is becoming more effective at distinguishing real from generated data.

Robustness

Point Completion Network (PCN):

DuctSilencer category:
Tested on noisy-partial PCs:
The confusion did not change by testing with noisy partial input PCs.

Testing on unseen data (e.g., skateboard category from initial repository data):

L1_CD(1e-3)	L2_CD(1e-4)	FScore-0.01(%)
81.3201	237.3259	5.6614

SpaceHeater category:
Testing on noisy PCs:
The confusion did not change by testing with noisy partial input PCs.

Testing on unseen data (e.g., skateboard category from initial repository data):

L1_CD(1e-3)	L2_CD(1e-4)	FScore-0.01(%)
1477.5923	89091.5868	0.0052

The robustness of both models regarding unseen categories is still low, especially for the SpaceHeater category. Better values for the DuctSilencer category were achieved, probably due to a more similar orientation of the DuctSilencer instances with the skateboard category instances. Comparing the quantitative results of both methods, better values can be achieved with a larger amount of data and same orientations for identical instances.

4.3.2 Visual Inspection

Point Completion Network (PCN):

In Figure 4.2 and Figure 4.3 results of the PCN completion applied on my datasets for the categories SpaceHeater and DuctSilencer are displayed. For the SpaceHeater category, 4 testing examples were chosen and for the DuctSilencer category, 2 testing examples were taken.

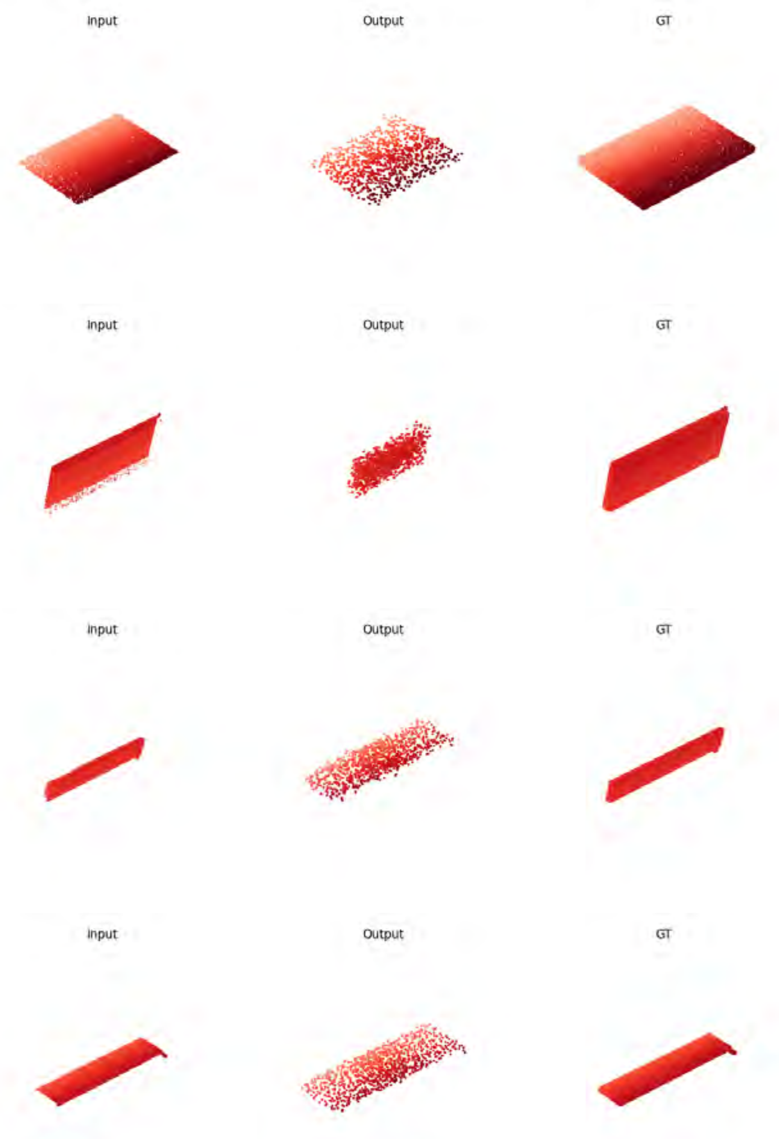


Figure 4.2: Completion results for SpaceHeater category

In row 3 of the SpaceHeater results, the wrongly orientated output can be identified. This is due to the lack of availability of training data of such an positioned instance of a SpaceHeater. The visible ratio of the input data is quite good for the SpaceHeater category, that is why the output already represents a good shape, even though it is still worse than the input.

The pre-trained model of the SpaceHeater category was also tested with the partial noisy input data and the results look fairly the same as for the partial accurate ones. This means, even for real-world scanned instances, that the PCN is robust enough to remove such outlier point due to the strong shape prior in the training. Probably a minimum of accurate points, which describes the shape, has to be given as a partial real-world scan input.

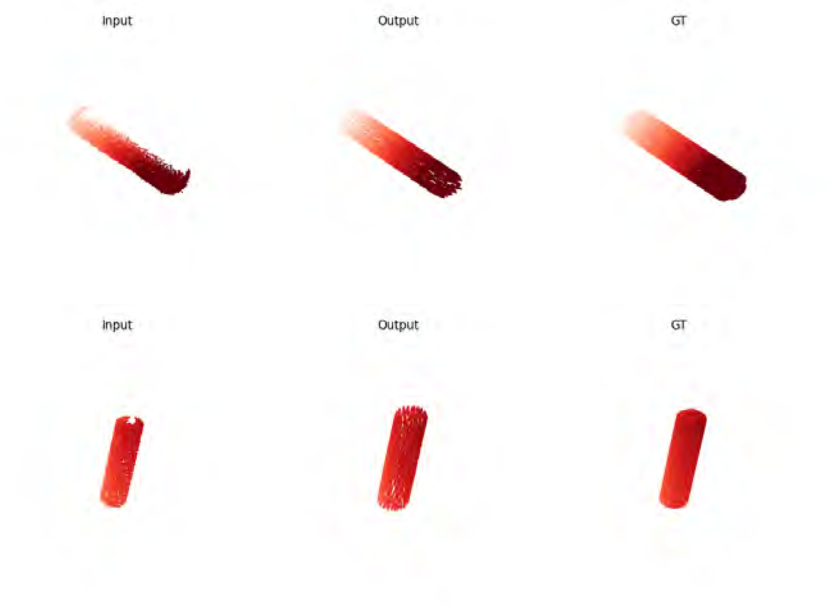


Figure 4.3: Completion results for DuctSilencer category

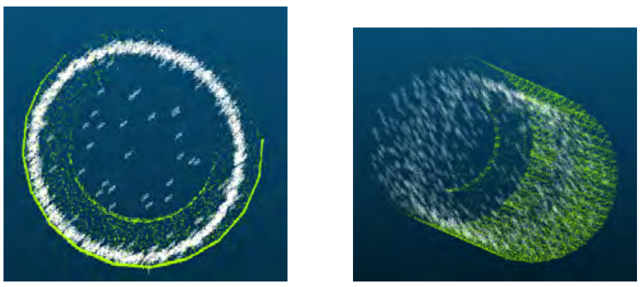


Figure 4.4: Overlaying Input (green) and Output (white) of DuctSilencer Testing instance (front and side)

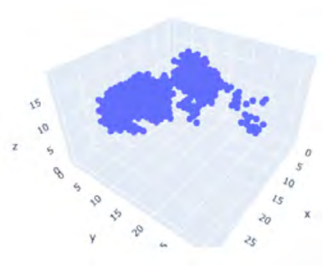
In Figure 4.4, the generated output PC (white) and the partial input PC (green) of a DuctSilencer testing instance is shown. Occluded points were successful generated, even if the dimensions differs to the Input PC due to lack of training data.

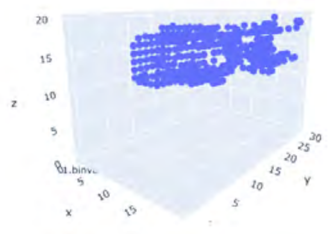
The pre-trained model of the DuctSilencer category was also tested with the partial noisy data and the results looks similar as the partial accurate ones.

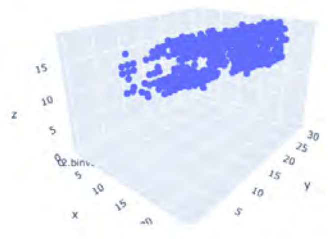
In general, the PCN could not perfectly complete the instances due to the lack of a larger dataset, but it has a huge potential to receive the same accuracy as the initial example data of the given repository (Source Code).

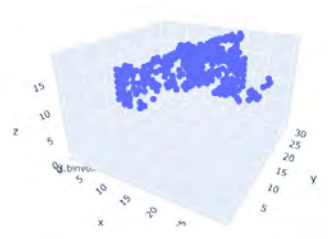
Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN):

The sampled voxels next to the actual instance of the results regarding the DuctSilencer category were created due to the different positions (rotations, translations) of each instance. This results in training instances with a combination of those differently rotated instances (see Figure 4.5).


(a)


(b)


(c)


(d)

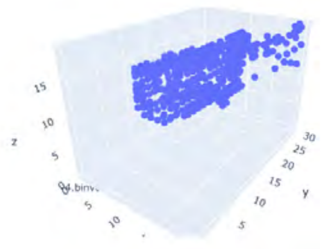

(e)

Figure 4.5: **Completion results of Voxel-DCGAN for DuctSilencer category**

In general the Voxel-DCGAN creates more random and not so accurate PCN results. The testing of the Voxel-DCGAN on noisy partial data can not be done due to the bad triangulation step of the the initial simulated partial PCs which is required for the voxelization. In Voxel-DCGAN more investigation should be made on how to use the implemented validation and testing steps results on the Machine Learning (ML) model. Further explanations and guidelines for using the Voxel-DCGAN would be helpful as well. Additionally, there was limited support for the Voxel-DCGAN repository on Github in comparison to PCN repository.

4.4 Reconstruction of completed Point Cloud Instances

The Aurivus (Stefan Hörmann, Martin Bach, 2019) and Cloud2Model (BIM Facility, Digital Twins, 2017) plugins in Revit (Autodesk, 2000) were firstly considered for the reconstruction process, but the required import Point Clouds (PCs) formats for those plugins have prevented their use. Aurivus is e.g., only supporting .aurivus and PLY file formats generated from their neural network.

The reconstruction step was finally done by using MeshLab (CIGNONI et al., 2008). It is an open source system for processing and editing 3D triangular meshes. It provides a set of tools and features for processing raw data produced by 3D digitization tools or devices and for preparing models for 3D printing.

The output PCs results of the PCN are in the PLY format and were directly imported in MeshLab. As a first step, a `PC Simplification` in the `Sampling` filter with 1000 points was done for receiving a better mesh. For the SpacehHeater the `PC Simplification` was done with 300 points due to worse results in a mesh with a higher amount of sampled points. Afterwards, the `Surface reconstruction: Ball Pivoting` was applied with its initial settings. The completed PCs and their reconstructed meshes of the PCN in the category DuctSilencer and SpaceHeater are shown in Figure 4.6.

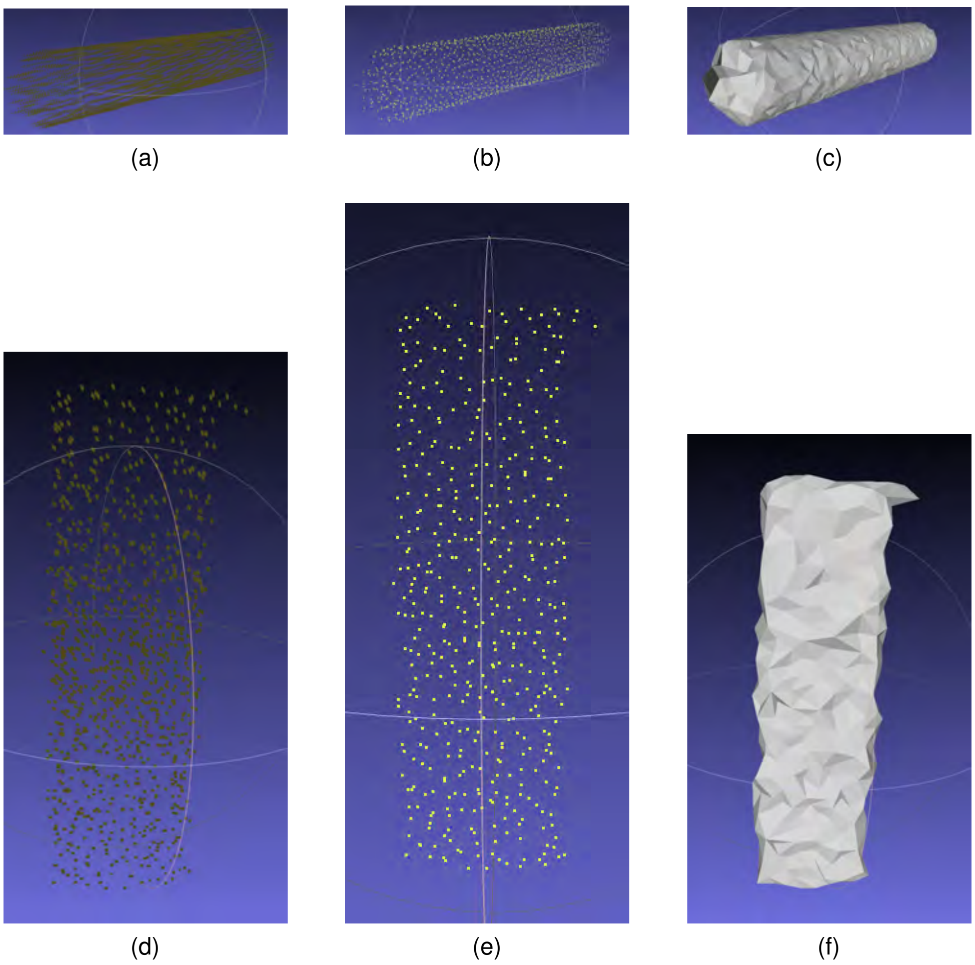
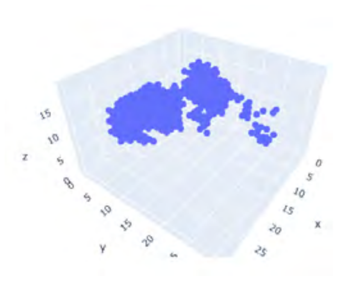


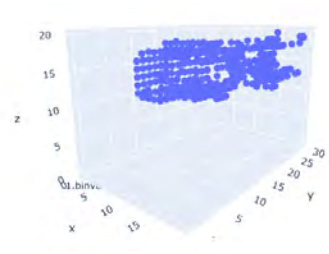
Figure 4.6: **Examples of reconstructed meshes of the PCs results through PCN.** (a) and (d): Completion results of the PCN. (b) and (e): Simplified PCs. (c) and (f): Reconstructed meshes.

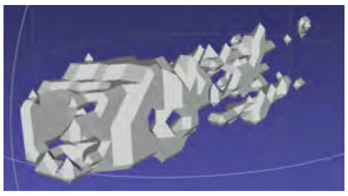
In the images c) and f) in Figure 4.6 the approach for the shape completion is visible. Despite of, the reconstructed meshes can still haver a better quality. The lack of input data in the PCN step is the reason that no more accurate meshes were obtained.

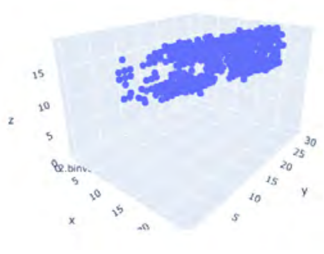
The results of the Voxel-DCGAN were in the .binvox format and it was not possible to directly import in MeshLab. Therefore, a .binvox to PC or STL conversion was done for further reconstruction steps in MeshLab. The conversions can be done by using (Source Code) on a Linux system or with (Source Code), which converts a PC directly to a mesh file (STL). The reconstructed mesh files of the Voxel-DCGAN results are shown in Figure 4.7.


(a)

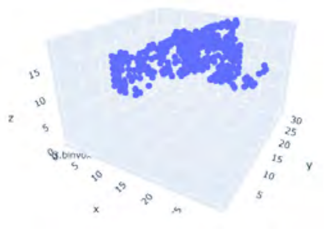
(b)

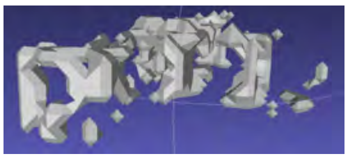

(c)

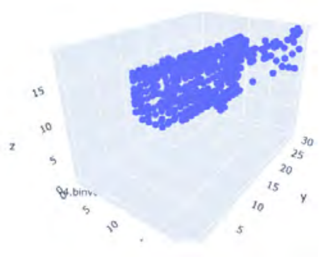

(d)


(e)

(f)


(g)


(h)


(i)

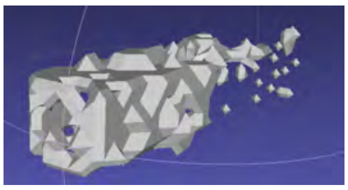
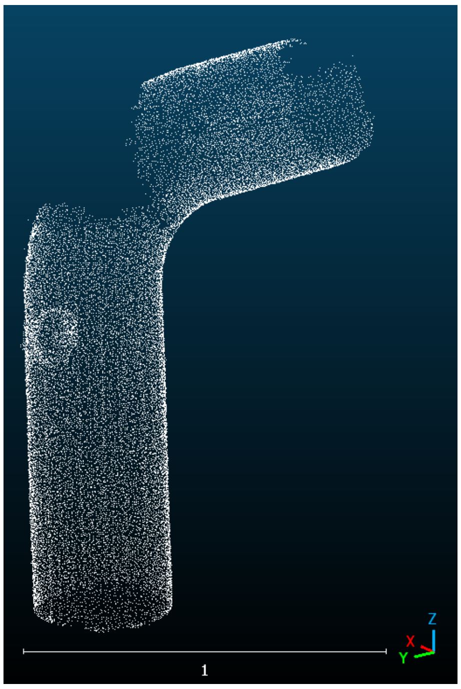

(j)

Figure 4.7: **Examples of reconstructed meshes of the .binvox results through Voxel-DCGAN**. (a), (c), (e), (g) and (i): Completion results of the Voxel-DCGAN. (b), (d),(f), (h) and (j): Reconstructed meshes

The floating points e.g., in (b) and (h) were created due to the fact that the Voxel-DCGAN was trained on DuctSilencer instances with different positions (rotated and translated).

4.5 Testing Results on Pre-segmented Real-World Point Cloud Data

Aurivus provided two segmented real-world Point Clouds (PCs) which includes Mechanical, Electrical and Plumbing (MEP) components; those files were only provided in their .aurivus format. No further investigation was made regarding those examples. On the other hand, Aurivus unlocked five MEP projects to be able to segment five PCs by their software. Therfore, the available open-source post-processed NavVis PC was segmented with Aurivus, in which still some points are occluded and missing. It was split into different groups e.g., "Pipes" and "Pipe_Fittings" etc. and stored in the E57 format. To extract certain partial PC instances out of the segmented NavVis "Pipes" PC in CC the `Label Connected Components` segmentation function was used. The following instance examples were exported in the PLY format, zero-centered, normalized:



(a)

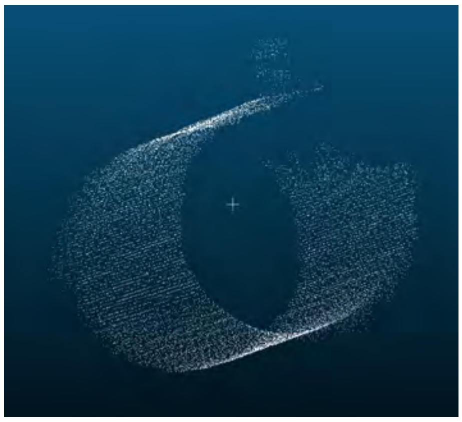

(b)

Figure 4.8: (a) and (b) shows two different pre-segmented real-world partial pipe PCs captured by the NavVis VLX3

Surface reconstruction ball pivoting was applied for both PLY to receive needed OBJ files for applying the Voxel-DCGAN. Afterwards, both instances were used to test the results obtained with the pre-trained model of the DuctSilencer as the PipeSegment category was quite challenging to create.

Point Completion Network (PCN):

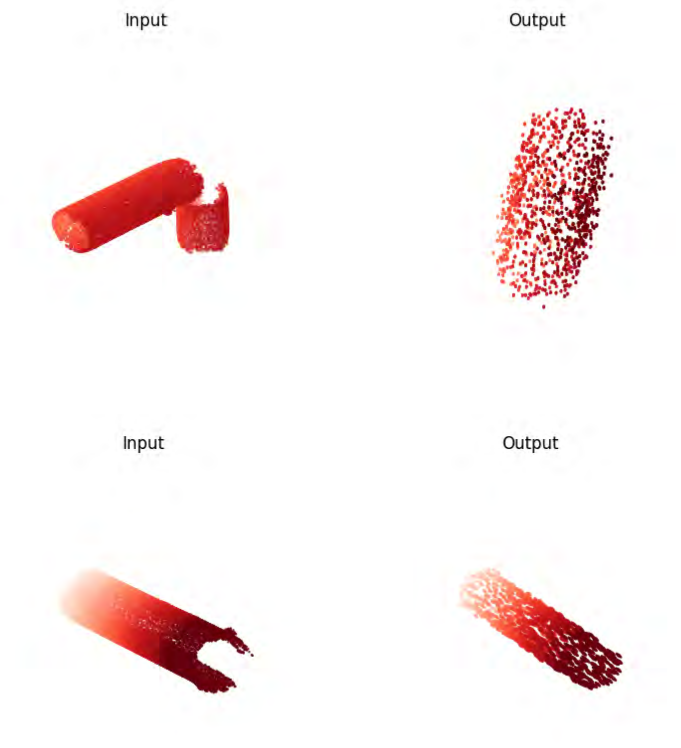


Figure 4.9: Completion results for real-world occlued PCs captured by the NavVis VLX3

Figure 4.9 clearly shows that the model is not trained on those types of instances. No Ground Truth for each of those instances was available which makes it difficult or impossible to quantitatively evaluate the network's performance. This makes the training and evaluation process more difficult. Without a Ground Truth Point Cloud (PC), the assessment of the PCN is more subjective. But based on virtual inspection, the completed PCs are not reflecting the same input PC due to the lack of training data which would make the model more stable in its variety of instances in this category. Additionally, the pre-trained model of the DuctSilencer was chosen due to the limited appropriate dataset, even though those real-world instances are usually tested with the pre-trained model of the PipeSegment category.

Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN):

The following results derived from the used partial NavVis data: The average loss of the Discriminator was 16.29866219 and the average loss of the Generator was 5.06487799, which means it is not converging to a useful solution. No further results were created, only the created instances during the Training step were produced. To get better results out of this method, the Ground Truths of those NavVis input instances have to be obtained to train Voxel-DCGAN on this data.

4.6 Discussion

First of all, instance segmentation with the help of my Python File (PY) scripts was successfully achieved. Usually, the path planning for real-world scans is a whole area by itself. Due to the computational and time efforts of planning the best paths for the virtual scans, the paths used for doing the virtual scanning in each storey were simplified. All the paths were chosen to cover all areas of each storey for the most part.

Moreover, comments to the steps for preparing the instances to apply the PCN and the Voxel-DCGAN are as follows:

Initially, the naming of each instance file was planned with the GUID as filenames. But as the differentiation of capital and small letters in different files in the same folder cannot be handled from the computer, the numbering of the HeliosID was taken. For each different storey an additional numbering was added to be able to differentiate between the files from each of them.

Additionally, in most cases a slight offset was created after executing the zero-centering and normalizing on the partial PCs (see Figure 4.10). This is likely to occur in the zero-centering step as the calculation of the mean (average) of the x, y, and z coordinates separately for all points, in the partial PCs, differs from the Ground Truth calculation. A different centroid is subtracted of each point in the partial PC. This offset is influencing the training process in the PCN by training on wrongly positioned partial PCs.

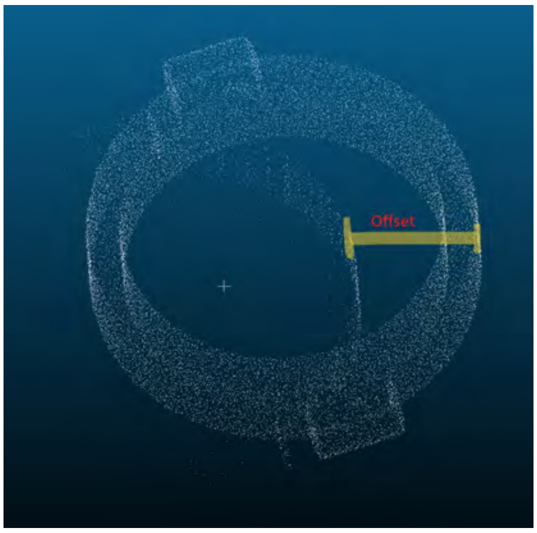


Figure 4.10: Offset from Ground Truth and partial PCs due to zero-centering (Instance HeliosID:10217)

Another challenge was to handle the initial positions of each Point Cloud (PC) instance. In consideration, every different positioned instance, even when it was the same instance as another same positioned instance, was handled as a completely independent instance with its partial PCs in its position. This made it quite challenging to be able to get the necessary amount of partial PCs of each complete PC in the training step for my dataset.

The partial instances of each instance are rarely available in my dataset in comparison to the amount of data used in the PCN dataset given in the repository. This lead to only partly or wrongly completed instances. In consideration, all Ground Truths and their corresponding partial PCs should be re-positioned to the Ground Truth position of one instance to get better results. Another suggested approach is to get a substantial amount of complete and incomplete PCs with its different instances positions in the future, but it seems to be quite challenging, thus investigation should be made on the first one. For the mentioned offset of Ground Truth and partial PC as well as on the multiple different orientations of Ground Truths and their corresponding partial PC, using (Open3D:Point Cloud Transformation) and (Open3D:ICP Registration) could be the first approaches to solve it. For different positioned instances (PCs) regarding one Ground Truth the Iterative Closest Point (ICP) algorithm is already repositioning them. But for the partial observed PCs it was quite challenging and not solvable in this Thesis.

Additionally, it was observed that the sampling option in CloudCompare (CC) does not always create exactly 16384 points for each instance even if it is set to do so. For this reason, first the surface reconstruction and afterwards the sampling was made by using Open3D in a PY script (Algorithm B.3) for the Ground Truths.

The reason that the PC file for the storey 00 is much larger than the others is that it has many more components. Additionally, the files for all storeys are larger than the NavVis raw PC data with similar scanning conditions, as HELIOS++ gives more output information for each point.

The created mapping tables for each storey proved to be a strong tool and useful for steps for e.g., checking the correctness of each instance's HeliosID in the filenames and for sorting the files in into the same folder structure for the PCN as provided.

In general, HELIOS++ supports different sources of error to mimic real laser scanning systems. Noise can result from multiple returns, caused by the virtual beams hitting the surface at steep incidence angles or by partial hits, i.e., only part of the laser beam footprint is hitting a surface. Additionally, the reason for the noisy partial PC obtained from the simulations in HELIOS++ is the `accuracy_m` parameter set in the scanner XML. When this value is set from 0.01 to 0.0 it avoids this range error, but changes the initial setting given for the Hesai LiDAR sensor. Furthermore, noisy PC could be also the reason of coordinate precision problems. Such errors were avoided in the simulations performed for the purposes of this Thesis by setting the `maxNOR` (maximum number of returns) to 1 in the scanner XML and the `beamSampleQuality` to 1 which controls how the beam footprint is approximated using subrays.

Despite of the fact that the PCN can handle noisy input PCs for training, it was trained using the partial PCs that excluded this noise error. Therefore, PCN model focused on learning strong shape prior and can remove noisy points, as well as predict reasonable outputs even if the input data is noisy. This, to further strengthen the aforementioned PC capacities. When executing the EMD Testing with the initial settings a CUDA errors occurred. This is a current PyTorch problem, but some workarounds are a) lower batch sizes and b) setting specific gpu torch.cuda.set_device(1). Here the batch size was lowered to 8 for the EMD.

4.7 Review of Research Questions

This section focuses on the reflection of the Thesis. Every scientific works aims to provide a benefit to its research topic and so did this work. Looking back to the objectives of this Thesis described in section 2.7 the following results can be stated:

- A virtual simulation framework that uses a given Building Information Modeling (BIM) model for the data generation part, was derived. This includes data from partial and Ground Truth instances in needed data formats in an automatized process to handle a larger amount of data.

- Multiple virtual laser scanning simulations were performed in Heidelberg Light Detection and Ranging (LiDAR) Operations Simulator (HELIOS++) to gather realistic segmented partial Point Cloud (PC) instances captured by the Mobile Laser Scanning (MLS) device NavVis VLX3. For setting this up, all specifications and needed settings were collected either from Hesai Technology or with the help of the HELIOS++ support.

- The dataset preparation of the segmented partial PCs instances and the segmented Ground Truths PCs instances was done successfully. For the Point Completion Network (PCN) the respective partial and complete Polygon File Format (PLY) files were created and sorted in an appropriate folder structure. For the Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN), related partial and complete Wavefront Object Format (OBJ) files were created which are possible to voxelize and therefore are usable for the Voxel-DCGAN.

- Both methods' codes were adapted to my dataset and set up to mainly work with them in combination with Jupyter Notebooks and a respective Google Drive folder which contains all the data.

- Conclusions and recommendations were made for all the steps, from the data generation part until the output of the completion methods.

For applying the results of this Thesis on real world laser scanning data, the scanning data needs to be pre-segmented, in the supported data format. Additionally, the instances need to for e.g., be firstly change to zero-centered coordinates and then back to its original coordinates or with an advanced approach to directly complete the single instances in their original position in the total PC.

4.8 Method's Usability in Scan-to-BIM

The PCN has a huge potential for using it to complete Mechanical, Electrical and Plumbing (MEP) Point Cloud (PC) instances in the real world environment. It takes directly PC as inputs and gives PC as outputs. As this method is already working in the application of e.g., autonomous cars, the completion of MEP components can be well integrated in the Mobile Laser Scanning (MLS) process. Additionally, after a discussion with a NavVis colleague, the completion step could be integrated e.g., either in the real-time laser scanning step in which it would get noisy PC instances as an input or even in the post-processing step in which it would get non-noisy PC instances. In both stages the occluded areas are almost the same. It would better for the PCN to implement the efficiency of bounding boxes to be able to complete each incomplete PC instance "on-the-run" as it is e.g., already applied in the mentioned autonomous driving. This Thesis, through the virtual simulation workflow contributes to get better real-world sensor data than the input of incomplete data used in Yuan et al. (2019). In general, the output of the PCN can improve the results of PC registration.

It became evident that the Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN) has less potential in this whole workflow, considering that for voxelizing as a first step, the following file formats are only possible: OBJ, Geomview OFF, Autocad DXF, PLY and Standard Triangle Language (STL), only when they contain polygons. In comparison to the PCN input, which does not require the voxelization step and can be directly applied on raw or post-processed partial PC instances. It is probable that this conversion step can already lead to slight changes to the input which makes the method more inaccurate. The output format of the completed instances is .binvox, which means that for directly modeling the instances, some steps in between are necessary to get an appropriate format for modeling. Therefore, the same inaccuracy can occur for the output .binvox files to PC files. Furthermore, it is suggested that the validating and testing steps are further implemented in this method, as it is probable that they lead to an improved model and better results.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This work tried to contribute to the enhancement of the workflow initial receiving raw occluded Mechanical, Electrical and Plumbing (MEP) Point Cloud (PC) instances until the completion of those components. A synthetic approach was investigated and validated to its extend. Therefore the given Building Information Modeling (BIM) model was used to generate realistic segmented PC instances which were further used for PC completion purposes. Two different approaches were performed and evaluated based on their workflows and their results. Moreover, simple solid geometries of completed instances were manually generated as examples. Additionally, the Machine Learning (ML) model results were tested on pre-segmented already post-processed real-world PC data captured by the NavVis VLX3. NavVis has a quite powerful PC post-processing tool which reduces the noise in the partial PC instances to low percentage (see Figure 4.8).

Moreover it is important to point out that to train large datasets, massive compute power is needed, and ML practitioners must be able to procure and leverage hundreds of GPUs to train their models. Here it was identified that Google Colab is a good browser based ML tool to get access to strong GPUs and advanced RAM. Furthermore, to improve the ML methods, high-quality data needs to get used. Oftentimes, generative ML models are used to produce synthetic data for different use cases. However, while troves of data are being generated globally every day, not all data can be used to train ML models. In the case here exposed, the occlusions of the training data can be synthetically generated, but the real-world scanning occlusions are not considered then. Therefore, the training is done for wrong data.

Furthermore, compounding the issue of a lack of high-quality data, the need to get commercial license to use existing datasets or to build bespoke datasets to train generative models is challenging. Additionally, many datasets do not exist for certain domains. Nonetheless, many companies such as NVIDIA, Cohere, and Microsoft have a goal to support the continued growth and development of generative ML models with services and tools to help solve these issues. These products and platforms abstract away the complexities of setting up the models and running them at scale.

As a conclusion, the Point Completion Network (PCN) and Voxel-Deep Convolutional Generative Adversarial Networks (Voxel-DCGAN) algorithms can be used to complete PC instances, that are indistinguishable from their Ground Truth. They both need a larger amount of dataset for each category which was challenging for each instance due to the different positioning of instances for the same object type. Manual PCs inspections are needed to create such small dataset for one category.

Overall, generative ML algorithms can help automate and accelerate a variety of tasks and processes, saving time and resources for businesses and organizations. It has the potential to significantly impact a wide range of industries and applications and is an important area of ML research and development.

5.2 Outlook

After completing all Polygon File Format (PLY) files they need to get unnormalized and assigned, with their original local coordinates, to be able to place them back in their original position in the total Point Cloud (PC).

More data has to be created, especially more Ground Truths and partial PCs, which are different types of instances and different positioned (rotated and translated) in one category, which will result in an increase of the data available for the validation and testing steps. This is also particularly true for the training data, which requires a minimum of two different partial PCs for each complete PC, resulting in more data needed to train the ML model for different rotated instances captured from various positions.

A further enhancement of the process described in this Thesis would be if the PLY output files including surface normals are possible in HELIOS++ in the future as they would contain PC and mesh information directly from the input scene. That would simplify the process using the partial PC instances, allowing the PCN and the Voxel-DCGAN methods to be directly applied. Additionally, as adjusting the differentiation of the laser emitter and laser receiver in HELIOS++ is not possible yet, it is considered that implementing this in HELIOS++ would improve the output data making it even more realistic. Moreover, as the syntax of varying `scanFreq_hz` can not be yet done in HELIOS++ yet, the settings for the highest `scanFreq_hz` should be taken, as well as its dependent settings to get the best results.

In the future, it would be desirable to check if glass structures and other elements can be or are included in the used BIM model for the virtual scanning. Such, considering that those instances can lead to erroneous realistic results in HELIOS++. If such instances are included in the BIM model then they should e.g., be changed to a wall type in the BIM model or OBJ file, so that those instances are taken into account in the virtual scanning with HELIOS++.

One additional point that needs to be considered is that not all ObjectTypes that are equal in the rows of the mapping tables are actual exactly the same object. They differ in various categories e.g., with regards to the diameter of the pipe which is not directly representing it in the mapping tables; manual inspection was needed to keep them separated.

For gathering more data, Ground Truths, partial data, and simulation results of the other storeys can be used. Additionally, for collecting the non-noisy partial data, the simulations for the other storeys need to be done with setting the `accuracy_m` parameter in the scanner XML to 0.

For the upcoming ZHU et al. (2023), which is relocating and reconstructing multi-instances in changing 3D environments, the results of this Thesis can be used for further focus on Mechanical, Electrical and Plumbing (MEP) components. ZHU et al. (2023) already considered the mentioned different positioning (rotated, translated) of instances in comparison to the training data. As my approach mainly focused on the completion and reconstruction goals, ZHU et al. (2023) will add the relocation step which can be combined with the reconstruction one, here exposed. Furthermore, it shows that their method is accumulating on a instance level while doing more scans on different time stamps. The completeness of the instance PCs and reconstruction of the instances are increasing. For re-scans in different time stamps in which instances were moved or just different positioned (rotated an translated) e.g., in a factory this upcoming repository plays a crucial role.

Finally, including Vertex coloring in the completion step could be an interesting approach in the future. PLY files are also able to take colors by vertex which means that the training, validating and testing steps could focus as well on the coloring in the completion step.

Appendix A

Material Referencing - Python Scripts

Algorithm A.1: Materialclasses_Assignments_mtl.py

```
 # -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
import random

def insert_classification_numbers(file_path):
    # Read the file and save its content
    with open(file_path, 'r') as file:
        lines = file.readlines()

    # Tracking the numbers used
    used_numbers = set()

    # Scrolling through the lines and inserting the classification numbers
    new_lines = []
    for line in lines:
        line = line.strip()
        new_lines.append(line)

        if line.startswith('newmtl'):
            # Check if the previous line meets the requirements
            if len(new_lines) > 1:
                prev_line = new_lines[-2].strip()
                if prev_line.startswith('Ka') or prev_line.startswith('Kd')
                    ↪ or prev_line.startswith('Ks') or prev_line.startswith(
                    ↪ 'Ns') or prev_line.startswith('d') or prev_line.
                    ↪ startswith('newmtl'):
                    # Generation of a unique random number
                    classification_number = 1 # Change for other .mtl files
                    while classification_number in used_numbers:
                        classification_number += 1

```

```python
                    # Adding the classification number to the list of used numbers
                    used_numbers.add(classification_number)

                    # Inserting the line with classification number
                    new_lines.insert(-1, f'helios_classification {classification_number}')

    # Generation of a unique random number for the last line
    classification_number += 1
    while classification_number in used_numbers:
        classification_number = random.randint(1, 100000)

    # Adding the classification number to the list of used numbers
    used_numbers.add(classification_number)

    # Adding the last line with classification number
    new_lines.append(f'helios_classification {classification_number}')

    # Writing the updated content back to the file
    with open(file_path, 'w') as file:
        file.write('\n'.join(new_lines))

# Example call with a .mtl file named 'Example.mtl' - Change for other .mtl files
insert_classification_numbers('SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.mtl')
```

Algorithm A.2: Calculation_objmaterials_amount.py

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Jul 3 23:05:04 2023

@author: Hannes
"""

def count_word_occurrences(filename, word):
    count = 0
    with open(filename, 'r') as file:
        for line in file:
            count += line.count(word)
    return count

filename = 'SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.obj' # Change for
    ↪ other .obj files
word = 'usemtl'

frequency1 = count_word_occurrences(filename, word)
print(f'The frequency of "{word}" in the file "{filename}" is: {frequency1}
    ↪ ')

def count_word_occurrences(filename, word):
    count = 0
    with open(filename, 'r') as file:
        for line in file:
            count += line.count(word)
    return count

filename = 'SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.mtl' # Change for
    ↪ other .mtl files
word = 'newmtl'

frequency2 = count_word_occurrences(filename, word)
print(f'The frequency of "{word}" in the file "{filename}" is: {frequency2}
    ↪ ')
```

Algorithm A.3: Adding_material_mtl.py

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 26 16:51:55 2023

@author: Hannes
"""
# Adds the number of new materials to have the same number of objects and materials
def count_word_occurrences(filename, word):
    count = 0
    with open(filename, 'r') as file:
        for line in file:
            count += line.count(word)
    return count

filename = 'SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.obj' # Change for other .obj files
word = 'usemtl'

frequency1 = count_word_occurrences(filename, word)
print(f'The frequency of "{word}" in the file "{filename}" is: {frequency1}')

def count_word_occurrences(filename, word):
    count = 0
    with open(filename, 'r') as file:
        for line in file:
            count += line.count(word)
    return count

filename = 'SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.mtl' # Change for other .mtl files
word = 'newmtl'

frequency2 = count_word_occurrences(filename, word)
print(f'The frequency of "{word}" in the file "{filename}" is: {frequency2}')

diff=frequency1-frequency2
print(f'The difference is: {diff}')

import random
```

```

def generate_random_number(existing_numbers, start, end):
    random_number = random.randint(start, end)
    while random_number in existing_numbers:
        random_number = random.randint(start, end)
    return random_number

def add_surface_styles(filename, num_blocks):
    existing_surface_styles = set()
    existing_classification_numbers = set()

    # Reading the existing data in the .mtl file
    with open(filename, 'r') as file:
        lines = file.readlines()

        # Search for existing row blocks
        for line in lines:
            if line.startswith('newmtl '):
                existing_surface_styles.add(line.strip().split('-')[-1])
            elif line.startswith('helios_classification'):
                existing_classification_numbers.add(line.strip().split()[-1])

    # Generate the new row blocks
    new_lines = []
    classification_number = 60 # !! MUST BE CHANGED FOR EACH NEW MTL FILE
        ↪ !!! -> number must be the number of the last "
        ↪ helios_classification number" in the file.
    surface_style_number = 0
    for i in range(num_blocks):
        surface_style_number += 1
        classification_number += 1

        new_surface_style = f'newmtl surface-style-{surface_style_number}\n'
        new_classification = f'helios_classification {classification_number
            ↪ }\n'

        lines = '\n'

        new_lines.append(lines)
        new_lines.append(new_surface_style)
        new_lines.append(new_classification)

    # Adding the new row blocks to the end of the file
```

```python
    with open(filename, 'a') as file:
        file.writelines(new_lines)

print(f'{diff} materials have been added.')

# Example call: Adding appropriate row blocks in 'example.mtl'. - Change
    ↪ for other .mtl files
add_surface_styles('SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.mtl', diff)
```

Algorithm A.4: Check_identical_or_multiple_used_Materials.py

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Jul 3 23:52:38 2023

@author: Hannes
"""

def find_duplicate_usemtl_lines(file_path):
    lines_seen = set()
    duplicate_lines = []

    with open(file_path, 'r') as file:
        lines = file.readlines()

        for line in lines:
            if line.startswith("usemtl"):
                if line in lines_seen:
                    duplicate_lines.append(line.strip())
                else:
                    lines_seen.add(line)

    return duplicate_lines


# Example call
obj_file_path = 'SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.obj' # Change
    ↪ for other .obj files
duplicate_usemtl_lines = find_duplicate_usemtl_lines(obj_file_path)

if duplicate_usemtl_lines:
    print("The file contains the following identical usemtl lines:")
    for line in duplicate_usemtl_lines:
        print(line)
else:
    print("The file does not contain identical usemtl lines.")

def find_duplicate_newmtl_lines(file_path):
    lines_seen = set()
    duplicate_lines = []

    with open(file_path, 'r') as file:
        lines = file.readlines()

        for line in lines:
            if line.startswith("newmtl"):
                if line in lines_seen:
                    duplicate_lines.append(line.strip())
                else:
                    lines_seen.add(line)

    return duplicate_lines


# Example call
obj_file_path = 'SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.mtl' # Change
    ↪ for other .mtl files
duplicate_newmtl_lines = find_duplicate_newmtl_lines(obj_file_path)

if duplicate_newmtl_lines:
    print("The file contains the following identical newmtl lines:")
    for line in duplicate_newmtl_lines:
        print(line)
else:
    print("The file does not contain identical newmtl lines.")
```

Algorithm A.5: Check_replace_newref.py

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Jul 3 23:21:45 2023

@author: Hannes
"""

# Replacing duplicated lines & replacing "newmtl" with "usemtl" in .obj
    ↪ file

def replace_usemtl_with_mtl_lines(obj_file_path, mtl_file_path):
    obj_lines = []
    mtl_surface_styles = []

    # Reading the .mtl file and collecting the available surface-style
        ↪ lines
    with open(mtl_file_path, 'r') as mtl_file:
        mtl_lines = mtl_file.readlines()
        current_surface_style_index = 1

        for line in mtl_lines:
            if line.startswith("newmtl "):
                mtl_surface_styles.append(line.strip())
                current_surface_style_index += 1

    # Replacing the lines in the .obj file
    current_surface_style_index = 0

    with open(obj_file_path, 'r') as obj_file:
        obj_lines = obj_file.readlines()

    with open(obj_file_path, 'w') as obj_file:
        for line in obj_lines:
            if line.startswith("usemtl "):
                if current_surface_style_index < len(mtl_surface_styles):
                    surface_style_line = mtl_surface_styles[
                        ↪ current_surface_style_index]
                    obj_file.write(surface_style_line + '\n')
                    current_surface_style_index += 1
                else:
                    print("Not enough newmtl lines in the .mtl file.")
            else:
                obj_file.write(line)

# Example call
obj_file_path = 'SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.obj' # Change
    ↪ for other .obj files
mtl_file_path = 'SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.mtl' # Change
    ↪ for other .mtl files
replace_usemtl_with_mtl_lines(obj_file_path, mtl_file_path)

# Replace "newmtl" with "usemtl
def replace_newmtl_with_usemtl(obj_file_path):
    # Lesen des Inhalts der .obj-Datei
    with open(obj_file_path, 'r') as obj_file:
        obj_lines = obj_file.readlines()

    # Replace "newmtl" with "usemtl
    for i, line in enumerate(obj_lines):
        obj_lines[i] = line.replace('newmtl', 'usemtl')

    # Writing the updated content to the .obj file
    with open(obj_file_path, 'w') as obj_file:
        obj_file.writelines(obj_lines)

    print('Replacement completed.')

# Example call of the function
replace_newmtl_with_usemtl('SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.obj')
    ↪  # Change for other .obj files
```

Algorithm A.6: Reducing_materials_to_one.py

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 11 16:03:44 2023

@author: Hannes
"""

def process_obj_file(input_file, output_file):
    with open(input_file, 'r') as f:
        lines = f.readlines()

    new_lines = []
    current_usemtl = None
    in_product_section = False

    for line in lines:
        if line.startswith("g "):
            in_product_section = True
            current_usemtl = None
            new_lines.append(line)
        elif line.startswith("g "):
            in_product_section = False
            new_lines.append(line)
        elif in_product_section and line.startswith("usemtl"):
            if current_usemtl is None:
                current_usemtl = line
                new_lines.append(line)
        elif in_product_section:
            new_lines.append(line)
        else:
            new_lines.append(line)

    with open(output_file, 'w') as f:
        f.writelines(new_lines)

if __name__ == "__main__":
    input_file_path = "SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.obj" #
        ↪ Change for other .obj files
    output_file_path = "SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.obj" #
        ↪ Change for other .mtl files
    process_obj_file(input_file_path, output_file_path)
    print("Processing complete.")
```

Appendix B

Other helpful Python Scripts

Algorithm B.1: Transform_waypoints.py

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 26 23:55:28 2023

@author: Hannes
"""
import re

# Function for extracting the values from the round brackets with semicolon
    ↪  separation
def extract_values(text):
    pattern = r'\((.*?)\)'
    match = re.search(pattern, text)
    if match:
        values = match.group(1).split(';') # Separation with semicolon
        if len(values) == 3:
            return tuple(values)
    return None

# File names for the input and output files
eingabe_datei = 'Waypoints(picked_unmanipulated).txt'
ausgabe_datei = 'Waypoints(manipulated).txt'

# Open the input file and output file
with open(eingabe_datei, 'r') as input_file, open(ausgabe_datei, 'w') as
    ↪ output_file:
    for line in input_file:
        values = extract_values(line)
        if values:
            output_line = f'<leg>\n\t<platformSettings template="platform1"
                ↪ x="{values[0]}" y="{values[1]}" z="{values[2]}" />\n\t<
                ↪ scannerSettings template="scanner1"/>\n</leg>\n'
            output_file.write(output_line)

```

```
print(f'The values were extracted from {eingabe_datei} and written to {
    ↪ ausgabe_datei}.')
```

Algorithm B.2: zero-center_normalized.py

```python


import os
import numpy as np
import open3d as o3d

def zero_center_normalize(vertices):
    # Zero-centering
    center = np.mean(vertices, axis=0)
    vertices -= center

    # Normalizing to the unit cube
    max_coord = np.max(np.abs(vertices))
    vertices /= max_coord

    return vertices

def process_ply_file(input_path, output_path):
    # Read PLY file using open3d
    mesh = o3d.io.read_triangle_mesh(input_path)

    # Extract vertices and faces
    vertices = np.asarray(mesh.vertices)
    faces = np.asarray(mesh.triangles)

    # Zero-center and normalize vertices
    vertices = zero_center_normalize(vertices)

    # Create a new mesh
    processed_mesh = o3d.geometry.TriangleMesh()
    processed_mesh.vertices = o3d.utility.Vector3dVector(vertices)
    processed_mesh.triangles = o3d.utility.Vector3iVector(faces)

    # Write the processed PLY file using open3d
    o3d.io.write_triangle_mesh(output_path, processed_mesh, write_ascii=
        ↪ True)

def main(input_folder, output_folder):
    # Create output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Process each PLY file in the input folder
    for filename in os.listdir(input_folder):
        if filename.endswith(".ply"):
            input_path = os.path.join(input_folder, filename)
            output_path = os.path.join(output_folder, filename)
            process_ply_file(input_path, output_path)
            print(f"Processed: {filename}")

if __name__ == "__main__":
    input_folder = "E:/Simulations/-1/tum_data__-1_mls/2024-01-25_11-47-41(
        ↪ accuracy_0_0)/output_xyz_files/PLY_files"
    output_folder = "E:/Simulations/-1/tum_data__-1_mls/2024-01-25_11
        ↪ -47-41(accuracy_0_0)/output_xyz_files/PLY_files/Zero-
        ↪ centered_normalized_ascii"
    main(input_folder, output_folder)
```

Algorithm B.3: Sample_and_convert_to_ply.py

```python


import os
import open3d as o3d
import numpy as np

def sample_obj_to_ply(obj_filename, ply_filename, num_points=16384):
    # Load the .obj file using open3d
    mesh = o3d.io.read_triangle_mesh(obj_filename)

    # Convert the mesh to a point cloud
    point_cloud = mesh.sample_points_uniformly(number_of_points=num_points)

    # Get the points as a numpy array
    points = np.asarray(point_cloud.points)

    # Create a new PointCloud object with the sampled points
    sampled_point_cloud = o3d.geometry.PointCloud()
    sampled_point_cloud.points = o3d.utility.Vector3dVector(points)

    # Write the sampled points to a .ply file in ASCII format
    o3d.io.write_point_cloud(ply_filename, sampled_point_cloud, write_ascii=True)

# Create a folder for the output .ply files
output_folder = "PLY_files_sampled_correctly"
os.makedirs(output_folder, exist_ok=True)

# List of input .obj files in the current directory
input_obj_files = [file for file in os.listdir() if file.endswith(".obj")]

# Sample and save each .obj file
for obj_file in input_obj_files:
    obj_path = os.path.join(os.getcwd(), obj_file)
    ply_filename = os.path.join(output_folder, f"{os.path.splitext(obj_file)[0]}.ply")
    sample_obj_to_ply(obj_path, ply_filename, num_points=16384)
```

Algorithm B.4: Surface_reconstruction_obj.py

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 21 13:42:40 2024

@author: Hannes
"""

import os
import pymeshlab as ml

def screened_poisson_reconstruction(input_ply_path, output_obj_path):
    # Initialize MeshLab server
    ms = ml.MeshSet()

    # Load the input point cloud
    ms.load_new_mesh(input_ply_path)

    # Apply Screened Poisson reconstruction
    ms.apply_filter("generate_surface_reconstruction_screened_poisson")

    # Save the reconstructed surface as OBJ
    ms.save_current_mesh(output_obj_path)

def process_ply_files(input_folder, output_folder):
    # Create output folder if it doesn’t exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # List all PLY files in the input folder
    ply_files = [f for f in os.listdir(input_folder) if f.endswith(’.ply’)]

    for ply_file in ply_files:
        input_ply_path = os.path.join(input_folder, ply_file)
        output_obj_path = os.path.join(output_folder, os.path.splitext(
            ↪ ply_file)[0] + ’.obj’)

        # Apply Screened Poisson reconstruction
        screened_poisson_reconstruction(input_ply_path, output_obj_path)

if __name__ == "__main__":
    input_folder = "E:/Groundtruths/-1/PLY_export_CC_after_sampling/
        ↪ renamed_GT_xyz/XYZ_files_local_COS/PLY_files"
    output_folder = "E:/Groundtruths/-1/PLY_export_CC_after_sampling/
        ↪ renamed_GT_xyz/XYZ_files_local_COS/PLY_files/OBJ"

    process_ply_files(input_folder, output_folder)
```

Algorithm B.5: Sorting_to_categories.py

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 25 22:39:11 2024

@author: Hannes
"""

import os
import csv
import shutil

def extract_ply_numbers(folder_path):
    ply_numbers = []
    for filename in os.listdir(folder_path):
        if filename.startswith("output_") and filename.endswith(".ply"):
            # Extracting numbers from the .ply file names
            number = ''.join(filter(str.isdigit, filename))
            ply_numbers.append(number)
    return ply_numbers

def search_csv_for_names(csv_path, ply_numbers):
    names_dict = {}
    with open(csv_path, 'r', newline='') as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=';')
        next(csv_reader) # Skip header row
        for row in csv_reader:
            if len(row) >= 5:
                name = row[4] # Assuming column 5 (0-indexed) contains the
                    ↪ names
                number = row[2] # Assuming column 3 (0-indexed) contains the
                    ↪ numbers
                if number in ply_numbers:
                    names_dict[number] = name
    return names_dict

def copy_files_to_directory(source_folder, destination_folder, names_dict):
    for ply_number, name in names_dict.items():
        source_path = os.path.join(source_folder, f"output_{ply_number}.ply"
            ↪ )
        destination_subfolder = os.path.join(destination_folder, name)
        destination_path = os.path.join(destination_subfolder, f"output_{
            ↪ ply_number}.ply")

        if os.path.exists(destination_subfolder):
            shutil.copy(source_path, destination_path)
            print(f"Copied {name} to {destination_subfolder}")
        else:
            print(f"Skipped {name} as folder not found in {
                ↪ destination_folder}")

# Replace these paths with your actual directory paths
ply_folder_path = 'E:/Finale_Daten/partial_accurate/Zero-
    ↪ centered_normalized_ascii'
csv_file_path = 'E:/Finale_Daten/Mapping_Table_HeliosID_GUID_Typ_-1.csv'
destination_folder_path = 'E:/Finale_Daten/
    ↪ Partial_accurate_in_all_categories'

# Step 1: Extract numbers from .ply files
ply_numbers = extract_ply_numbers(ply_folder_path)

# Step 2: Search for names in the CSV file using the extracted numbers
names_dict = search_csv_for_names(csv_file_path, ply_numbers)

# Step 3: Copy files to the destination folder
copy_files_to_directory(ply_folder_path, destination_folder_path,
    ↪ names_dict)
```

Appendix C

Instance Creation - Python Scripts

Algorithm C.1: Example_merging_xyz_files.py

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 28 18:53:51 2023

@author: Hannes
"""

import os

# Directory in which the .xyz files are located
input_dir = r'E:\Simulations\-1\tum_data__-1_mls\2024-01-25_11-47-41(
    ↪ accuracy_0_0)' # !!!Change for different point cloud

# Name of the output file
output_file = 'Total.xyz'

# List to save the contents of the .xyz files
xyz_data = []

# Search the directory for .xyz files and add them to the list
for filename in os.listdir(input_dir):
    if filename.endswith(".xyz"):
        with open(os.path.join(input_dir, filename), 'r') as file:
            xyz_data.extend(file.readlines())

# Write the collected .xyz data to the output file
with open(output_file, 'w') as output:
    output.writelines(xyz_data)

print(f'The .xyz files have been combined into {output_file}.')
```

Algorithm C.2: Segmentation_reg_HeliosID.py

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 28 19:27:30 2023

@author: Hannes
"""

# Function for splitting the .xyz file into separate files
def split_xyz_by_value(input_xyz, output_directory):
    data_dict = {} ## A dictionary to group the data by values in the
        ↪ penultimate column

    # Open the input file
    with open(input_xyz, 'r') as input_file:
        for line in input_file:
            parts = line.strip().split()
            if len(parts) < 3:
                continue
            value = parts[-2] # Value in the penultimate column

            # Create a separate file for this value if it does not already
                ↪ exist
            if value not in data_dict:
                data_dict[value] = []

            data_dict[value].append(line)

    # Write data to separate .xyz files
    for value, data in data_dict.items():
        output_file = os.path.join(output_directory, f'output_{value}.xyz')
        with open(output_file, 'w') as output:
            output.writelines(data)

if __name__ == "__main__":
    import os

    # Paths to the input file and the output directory
    input_xyz_file = r'E:\Simulations\-1\tum_data__-1_mls\2024-01-25_11
        ↪ -47-41(accuracy_0_0)\Total.xyz' # !!!Change for different point
        ↪ cloud
    output_directory = 'output_xyz_files'

    # Create the output directory if it does not exist
    os.makedirs(output_directory, exist_ok=True)

    # Call function to split .xyz file
    split_xyz_by_value(input_xyz_file, output_directory)

print(f'The Total.xyz file was segmented in {output_directory}.')
```

Algorithm C.3: Surface_reconstruction_ball_pivoting_obj.py

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 21 13:42:40 2024

@author: Hannes
"""
import os
import pymeshlab as ml

def get_file_size(file_path):
    return os.path.getsize(file_path) / 1024 # Convert bytes to kilobytes

def ball_pivoting_reconstruction(input_ply_path, output_obj_path):
    # Initialize MeshLab server
    ms = ml.MeshSet()

    # Load the input point cloud
    ms.load_new_mesh(input_ply_path)

    # Apply Screened Poisson reconstruction
    ms.apply_filter("generate_surface_reconstruction_ball_pivoting")

    # Save the reconstructed surface as OBJ
    ms.save_current_mesh(output_obj_path)

def process_ply_files(input_folder, output_folder, max_file_size_kb=5000):
    # Create output folder if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # List all PLY files in the input folder
    ply_files = [f for f in os.listdir(input_folder) if f.endswith('.ply')]

    for ply_file in ply_files:
        input_ply_path = os.path.join(input_folder, ply_file)
        output_obj_path = os.path.join(output_folder, os.path.splitext(
            ↪ ply_file)[0] + '.obj')

        # Check file size before processing
        file_size_kb = get_file_size(input_ply_path)
        if file_size_kb > max_file_size_kb:
            print(f"Skipping {ply_file} due to size ({file_size_kb} KB > {
                ↪ max_file_size_kb} KB)")
            continue

        # Apply Screened Poisson reconstruction
        ball_pivoting_reconstruction(input_ply_path, output_obj_path)

if __name__ == "__main__":
    input_folder = "E:/Simulations/-1/tum_data__-1_mls/2024-01-25_11-47-41(
        ↪ accuracy_0_0)/output_xyz_files/PLY_files"
    output_folder = "E:/Simulations/-1/tum_data__-1_mls/2024-01-25_11
        ↪ -47-41(accuracy_0_0)/output_xyz_files/PLY_files/Ball_pivoting_obj
        ↪ "

    process_ply_files(input_folder, output_folder, max_file_size_kb=5000)
```

Algorithm C.4: Panda_Data_Frame_GUID_Typ_Material_HeliosID.py

```python
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 9 17:25:42 2023

@author: Hannes
"""
import os
import pandas as pd

file_name = 'SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.obj'
file_path = os.path.join(r'C:\Users\Hannes\Desktop\Masterarbeit\helios-
    ↪ plusplus-win\data\sceneparts\owndata\storeys\-1', file_name)

# Open the .obj file for reading
with open(file_path, 'r') as obj_file: # Change for each storey (.obj file)
    lines = obj_file.readlines()

# Initialize empty lists to store the data
guids = []
materials = []

current_guid = None
current_material = None

# Go through the lines of the .obj file
for line in lines:
    if line.startswith('g '):
        if current_guid is not None:
            guids.append(current_guid)
            materials.append(current_material)
        current_guid = line.strip().split(' ', 1)[1]
        current_material = None
    elif line.startswith('usemtl '):
        if current_guid is not None:
            current_material = line.strip().split(' ', 1)[1]

# Add the last "g" block
if current_guid is not None:
    guids.append(current_guid)
    materials.append(current_material)

# Create a Pandas DataFrame
```

```python
data = {'GUIDs': guids, 'Material': materials}
df = pd.DataFrame(data)

# Print the DataFrame
print(df.head)

# Save the DataFrame in an Excel file
df.to_excel('GUIDS_Materials_-1.xlsx', index=False)


# !!!Adding Helios IDs to Materials & GUIDs!!!
# Open and read the .mtl file
file_name = 'SCG_mergedall(except_AR)_IFC4_(EDITED)_-1.mtl'
file_path = os.path.join(r'C:\Users\Hannes\Desktop\Masterarbeit\helios-
    ↪ plusplus-win\data\sceneparts\owndata\storeys\-1', file_name)

with open(file_path, 'r') as mtl_file: # Change for each storey (.mtl file)
    mtl_lines = mtl_file.readlines()

# Initialize an empty list to store the Helios IDs
helios_ids = []

current_material = None

# Go through the lines of the .mtl file
for line in mtl_lines:
    if line.startswith('newmtl '):
        current_material = line.strip().split(' ', 1)[1]
    elif line.startswith('helios_classification '):
        if current_material is not None:
            helios_id = int(line.strip().split(' ', 1)[1])
            helios_ids.append((current_material, helios_id))

# Create a DataFrame from the list of Helios IDs
helios_df = pd.DataFrame(helios_ids, columns=['Material', 'HELIOS IDs'])

# Perform a left join between "df" and "helios_df".
result_df = df.merge(helios_df, on='Material', how='left')

# Save the resulting DataFrame in an Excel file
result_df.to_excel('DF_with_helios_ids_GUIDS_Materials_-1.xlsx', index=
    ↪ False)

```

```
# Print the resulting DataFrame
print(result_df.head)


#!!!merge with .ifc IfcProduct data regarding GUIDs!!!
import ifcopenshell
import ifcopenshell.util
from ifcopenshell.util.selector import Selector
schema = ifcopenshell.ifcopenshell_wrapper.schema_by_name("IFC4")

ifc = ifcopenshell.open('SCG_mergedall(except_AR)_IFC4_(EDITED)_-1(products
    ↪ ).ifc') # change for each storey (.ifc file) -> add ...(products).
    ↪ ifc

ifc_product = schema.declaration_by_name("IfcProduct")
variable=ifc_product.all_attributes()
print(variable)

selector = Selector()

products = selector.parse(ifc, '.IfcProduct')
for product in products:
    global_id = product.GlobalId if hasattr(product, "GlobalId") else "No
        ↪ GUID available"
    name = product.Name if hasattr(product, "Name") else "No Name available
        ↪ "
    object_type = product.ObjectType if hasattr(product, "ObjectType") else
        ↪  "No ObjectType available"
# Here you can further process or output the attributes
    print(f"Produkt ID: {product.id()}, GlobalId: {global_id}, Typ: {
        ↪ product.is_a()}, Name: {name}, ObjectType: {object_type}")

# Create an empty DataFrame
data = []

# Iterate through the products and collect data
for product in products:
    data.append({
       "Produkt ID": product.id(),
       "GlobalId": product.GlobalId if hasattr(product, "GlobalId") else "No
           ↪  GUID available",
       "Typ": product.is_a(),
```

```python
        "Name": product.Name if hasattr(product, "Name") else "No Name
            ↪ available",
        "ObjectType": product.ObjectType if hasattr(product, "ObjectType")
            ↪ else "No ObjectType available"
        })

# Convert products to a DataFrame
ifcproduct_df = pd.DataFrame(data)

# Perform a merge based on the GUIDs in result_df and the GlobalIds in
    ↪ ifcproduct_df
merged_df = result_df.merge(ifcproduct_df[['Produkt ID', 'GlobalId', 'Typ',
    ↪  'Name', 'ObjectType']], how='left', left_on='GUIDs', right_on='
    ↪ GlobalId')

# Remove the column 'GlobalId' which is no longer needed
merged_df.drop(columns=['GlobalId'], inplace=True)

# Save the updated DataFrame in an Excel file
merged_df.to_excel('Mapping_Table_HeliosID_GUID_Typ_-1.xlsx', index=False)

# Print the updated DataFrame
print(merged_df.head)
```

Algorithm C.5: Mapping_table_to_traingobj.py

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Nov 7 13:04:52 2023

@author: Hannes
"""

import os
import pandas as pd

# Specify the directory containing .obj files
obj_directory = r'E:\Simulations\-1\tum_data__-1_mls\2023-12-21_14-35-56\
    ↪ output_xyz_files'
output_directory = r'C:\Users\Hannes\Desktop\Masterarbeit\Kirn_MA_models\
    ↪ SecuFEx_2023-01-30_15-18-07_16965734554749689882\SimpleBIM\
    ↪ IFC_export_storeys\-1\Obj_and_mtl_files\After_manupulating\
    ↪ New_mapped_obj_files'

# List all .xyz files in the directory and sort them numerically
obj_files = sorted([file for file in os.listdir(obj_directory) if file.
    ↪ endswith('.obj')], key=lambda x: int(x.split('_')[1].split('.')[0]))

# Specify the path to the .xlsx table
xlsx_path = 'Mapping_Table_HeliosID_GUID_Typ_-1.xlsx'

# Loop through each .obj file
for obj_file_name in obj_files:
    # Construct the full path to the current .obj file
    obj_file_path = os.path.join(obj_directory, obj_file_name)

    # Load the .obj file into a DataFrame
    with open(obj_file_path, 'r') as obj_file:
        obj_lines = obj_file.readlines()

    # Create a DataFrame from the .obj data
    obj_data = []
    for line in obj_lines:
        if line.startswith('v '):
            parts = line.split()
            if len(parts) > 10:
                obj_data.append({
                    'ObjValue': float(parts[10])
                })
    obj_df = pd.DataFrame(obj_data)

    # Load the .xlsx table into a DataFrame
    xlsx_df = pd.read_excel(xlsx_path)

    # Join the two DataFrames based on matching values
    result_df = obj_df.merge(xlsx_df, left_on='ObjValue', right_on=xlsx_df.columns[2], how='left')

    # Create a new row starting with "g" and adding values from the .xlsx table
    new_row = f'g {" ".join(map(str, result_df.iloc[0].drop(xlsx_df.columns[2]).tolist()))}\n'

    # Create a new .obj file with the added row
    new_obj_lines = [new_row] + obj_lines

    # Save the updated .obj file with a new name
    new_obj_file_path = os.path.join(output_directory, f'New_mapped_{obj_file_name}')
    with open(new_obj_file_path, 'w') as new_obj_file:
        new_obj_file.writelines(new_obj_lines)

    print(f"The new .obj file {new_obj_file_path} has been created and saved.")
```

Algorithm C.6: Origin_to_Local_COS.py

```python
 # -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

import os

def move_coordinates(input_path, output_path):
    with open(input_path, 'r') as input_file:
        lines = input_file.readlines()

    # Move x and y coordinates and keep other columns
    shifted_lines = []
    for line in lines:
        columns = line.split()
        new_line = ""
        if len(columns) >= 3:
            x = round(float(columns[0]) - 2697000, 6)
            y = round(float(columns[1]) - 5349000, 6)
            z = round(float(columns[2]), 6)
            new_line = f"{x} {y} {z} {' '.join(columns[3:])}\n"
        else:
            new_line = line
        shifted_lines.append(new_line)

    # Create the output path
    if not os.path.exists(output_path):
        os.makedirs(output_path)

    # Write the moved coordinates to the new file
    output_filepath = os.path.join(output_path, os.path.basename(input_path)
        ↪ )
    with open(output_filepath, 'w') as output_file:
        output_file.writelines(shifted_lines)

if __name__ == "__main__":
    # Specify the path to the folder containing the .xyz files
    input_folder = "E:/Simulations/-1/tum_data__-1_mls/2024-01-25_11-47-41(
        ↪ accuracy_0_0)/output_xyz_files"

    # Specify the path to the new folder where the shifted files should be
        ↪ saved
    output_folder = "E:/Simulations/-1/tum_data__-1_mls/2024-01-25_11
        ↪ -47-41(accuracy_0_0)/output_xyz_files/XYZ_files_local_COS"

    # Iterate through all .xyz files in the input folder
    for filename in os.listdir(input_folder):
        if filename.endswith(".xyz"):
            input_path = os.path.join(input_folder, filename)
            move_coordinates(input_path, output_folder)

print("Shift completed.")
```

Algorithm C.7: Convert_to_PLY.py

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 14 13:20:18 2024

@author: Hannes
"""

import open3d as o3d
import os


def xyz_to_ply(input_xyz, output_ply):
    # Read the .xyz file
    points = []
    with open(input_xyz, 'r') as file:
        for line in file:
            values = line.split()
            if len(values) >= 3:
                x, y, z = map(float, values[:3])
                points.append([x, y, z])

    # Create an Open3D PointCloud object
    point_cloud = o3d.geometry.PointCloud()
    point_cloud.points = o3d.utility.Vector3dVector(points)

    # Save the PointCloud object in .ply format
    o3d.io.write_point_cloud(output_ply, point_cloud, write_ascii=True)


# Directory where the .xyz files are stored
xyz_directory = "E:/Simulations/-1/tum_data__-1_mls/2024-01-25_11-47-41(
    ↪ accuracy_0_0)/output_xyz_files/XYZ_files_local_COS"

# Directory where the .ply files will be saved
ply_directory = "E:/Simulations/-1/tum_data__-1_mls/2024-01-25_11-47-41(
    ↪ accuracy_0_0)/output_xyz_files/PLY_files"

# Iterate through all .xyz files in the directory
for filename in os.listdir(xyz_directory):
    if filename.endswith(".xyz"):
        input_xyz_path = os.path.join(xyz_directory, filename)

        # Generate the output path for the .ply file
        output_ply_filename = os.path.splitext(filename)[0] + ".ply"
        output_ply_path = os.path.join(ply_directory, output_ply_filename)

        # Convert .xyz to .ply
        xyz_to_ply(input_xyz_path, output_ply_path)
```

Appendix D

Submission Content

The enclosed ZIP file contains the following content:

- The final version of this thesis [.pdf]
- HELIOS++ data and files
- Links to the respective Gitlab LRZ repositories and Jupyter notebooks
- BIM models files of storeys and simpleBIM file
- Mapping Tables for each storey
- Ground Truths and partial (Point clouds and .obj files) of storey -1
- Noisy Point Clouds of all other storeys
- Completed Point Clouds and .binvox files

Bibliography

AUTODESK. (2000). *Autodesk revit: Bim-software für grenzenlose gestaltungsfreiheit* (Version 2021). https://www.autodesk.de/products/revit/overview?term=1-YEAR&tab=subscription

BERNARDINI, F., MITTLEMAN, J., RUSHMEIER, H., SILVA, C., & TAUBIN, G. (1999). The ball-pivoting algorithm for surface reconstruction. *Visualization and Computer Graphics, IEEE Transactions on*, *5*, 349–359. https://doi.org/10.1109/2945.817351

BIM FACILITY, DIGITAL TWINS. (2017). *Cloud2model*. https://www.bim-facility.ch/cloud2model

BORRMANN, A., KÖNIG, M., KOCH, C., & BEETZ, J. (2018, September). *Building information modeling technology foundations and industry practice: Technology foundations and industry practice*. https://doi.org/10.1007/978-3-319-92862-3

BUILDINGSMART CONTRIBUTORS. (2019a). Ifcroot [Online; accessed 07-September-2023]. https://standards.buildingsmart.org/IFC/DEV/IFC4_2/FINAL/HTML/schema/ifckernel/lexical/ifcroot.htm

BUILDINGSMART CONTRIBUTORS. (2019b). Ifcvalve [Online; accessed 24-August-2023]. https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2/HTML/schema/ifchvacdomain/lexical/ifcvalve.htm

CIGNONI, P., CALLIERI, M., CORSINI, M., DELLEPIANE, M., GANOVELLI, F., & RANZUGLIA, G. (2008). MeshLab: an Open-Source Mesh Processing Tool. In V. SCARANO, R. D. CHIARA, & U. ERRA (Eds.), *Eurographics italian chapter conference*. The Eurographics Association. https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136

DATACOMP. (n.d.). *Bimvision kostenloser ifc model viewer | bimvision* (Version 2.27.1). https://bimvision.eu/de/

FEI, B., YANG, W., CHEN, W.-M., LI, Z., LI, Y., MA, T., HU, X., & MA, L. (2022). Comprehensive review of deep learning-based 3d point cloud completion processing and analysis. *IEEE Transactions on Intelligent Transportation Systems*, *23*(12), 22862–22883. https://doi.org/10.1109/TITS.2022.3195555

GOGEOMATICS NEWSMAKERS. (2023). Navvis introduces navvis vlx 3 to take reality capture to the next level [Online; accessed 25-August-2023]. https://gogeomatics.ca/navvis-introduces-navvis-vlx-3-to-take-reality-capture-to-the-next-level/m

HESAI. (2023). Xt32m2x_user_manual_x03-en-230510 [Online; accessed 25-August-2023]. https://www.hesaitech.com/wp-content/uploads/2023/05/XT32M2X_User_Manual_X03-en-230510.pdf

IFCOPENSHELL CONTRIBUTORS. (2022). Ifcopenshell 0.7.0 documentation — ifcconvert [Online; accessed 19-August-2023]. https://blenderbim.org/docs-python/ifcconvert/usage.html

INTELLIGENCE, A., & LAB, E. D. (n.d.). 3d generative adversarial networks [Online; accessed 20-January-2024]. https://www.computationalarchitecturelab.org/3d-generative-adversarial-networks

JIRI HIETANEN, SAKARI LEHTINEN. (2019). *Simplebim* (Version 10). https://simplebim.com/

JULIANNA DELUA. (2021). Supervised vs. unsupervised learning: What's the difference? [Online; accessed 25-January-2024]. https://www.ibm.com/blog/supervised-vs-unsupervised-learning/

KARPATHY, A., ABBEEL, P., BROCKMAN, G., CHEN, P., CHEUNG, V., DUAN, Y., GOODFELLOW, I., KINGMA, D., HO, J., HOUTHOOFT, R., SALIMANS, T., SCHULMAN, J., SUTSKEVER, I., & ZAREMBA, W. (2016). Research generative models [Online; accessed 03-January-2024]. https://openai.com/research/generative-models

KASSIANI NIKOLOPOULOU. (2023). Supervised vs. unsupervised learning: Key differences [Online; accessed 25-January-2024]. https://www.scribbr.com/ai-tools/supervised-vs-unsupervised-learning/

MIN, P. (2004). Binvox [Accessed: 2024-01-26].

NIKOLAJ BUHL. (2023). F1 score in machine learning [Online; accessed 20-January-2024]. https://encord.com/blog/f1-score-in-machine-learning/

NOICHL, F., BRAUN, A., & BORRMANN, A. (2021). "bim-to-scan" for scan-to-bim: Generating realistic synthetic ground truth point clouds based on industrial 3d models [Publisher Copyright: © 2021, European Council on Computing in Construction (EC3). All rights reserved.]. In D. HALL, A. CHASSIAKOS, J. O'DONNELL, D. NIKOLIC, & Y. XENIDES (Eds.), *Proceedings of the 2021 european conference on computing in construction* (pp. 164–172). https://doi.org/10.35490/EC3.2021.166

NOORUDDIN, F. S., & TURK, G. (2003). Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, *9*(2), 191–205.

PAUL BOURKE. (2023). Object files (.obj); minimal example: Box.obj; minimal textured example; vertex colour in obj files [Online; accessed 28-November-2023]. https://paulbourke.net/dataformats/obj/

PAUL BOURKE. (2024). Ply - polygon file format; also known as the stanford triangle format; source code examples: Ply.h, plytest.c, plyfile.c, plydocs.txt; example of an ascii ply file [Online; accessed 15-January-2024]. https://paulbourke.net/dataformats/ply/

QI, C. R., SU, H., MO, K., & GUIBAS, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation.

RADFORD, A., METZ, L., & CHINTALA, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks.

SALIMANS, T., GOODFELLOW, I., ZAREMBA, W., CHEUNG, V., RADFORD, A., & CHEN, X. (2016). Improved techniques for training gans.

SEAN HIGGINS. (2021). Everything you need to know about point clouds | navvis [Online; accessed 03-January-2024]. https://www.navvis.com/blog/everything-you-need-to-know-about-point-clouds-navvis

STEFAN HÖRMANN, MARTIN BACH. (2019). *The fastest way from scan-to-bim* (Version 2019). https://aurivus.com/

Wang, B., Yin, C., Luo, H., Cheng, J. C., & Wang, Q. (2021). Fully automated generation of parametric bim for mep scenes based on terrestrial laser scanning data. *Automation in Construction*, *125*, 103615. https://doi.org/https://doi.org/10.1016/j.autcon.2021.103615

Winiwarter, L., Esmorís Pena, A. M., Weiser, H., Anders, K., Martínez Sánchez, J., Searle, M., & Höfle, B. (2022). Virtual laser scanning with helios++: A novel take on ray tracing-based simulation of topographic full-waveform 3d laser scanning. *Remote Sensing of Environment*, *269*. https://doi.org/https://doi.org/10.1016/j.rse.2021.112772

Xu, L., Chengtao, L., Jian, W., Jingbo, W., Boxin, S., & Xiaodong, H. (2020). Group Contextual Encoding for 3D Point Clouds. https://papers.nips.cc/paper/2020/file/9b72e31dac81715466cd580a448cf823-Paper.pdf

Yuan, W., Khot, T., Held, D., Mertz, C., & Hebert, M. (2019). PCN: Point Completion Network. http://arxiv.org/pdf/1808.00671v3

Zhou, Q.-Y., Park, J., & Koltun, V. (2018). Open3D: A modern library for 3D data processing. *arXiv:1801.09847*.

Zhu, L., Huang, S., & Konrad Schindler, I. A. (2023). Living scenes: Multi-object relocalization and reconstruction in changing 3d environments.

Declaration

I hereby affirm that I have independently written the thesis submitted by me and have not used any sources or aids other than those indicated.

Munich, 30.01.2024 [illegible]

Location, Date, Signature