Performance and Accuracy Assessment of Nvidia's Omniverse Isaac Sim for Generating Synthetic Data from Real-world Scenarios

BachelorThesis

von

Patrick Noras

October 16, 2023

Technische Universität Kaiserslautern,
Department of Computer Science,
67663 Kaiserslautern,
Germany

Examiner: Prof. Dr. Christoph Garth
Alexander Witton, M.Sc.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die von mir vorgelegte Arbeit mit dem Thema "Performance and Accuracy Assessment of Nvidia's Omniverse Isaac Sim for Generating Synthetic Data from Real-world Scenarios" selbstständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Kaiserslautern, den 16.10.2023

Patrick Noras

Abstract

Synthetic data has become an ever-increasing important tool for countless use cases in research and today's Industry 4.0. Deep neural networks (DNN), artificial intelligence (AI), and other machine learning (ML) applications require exceedingly large and well-annotated data sets, to be trained with. Apart from many other synthetic data generation methods, simulators remain to be the standard technique for synthesizing large amounts of data. However, closing the domain gap between synthetic and real data proposes to be a difficult challenge. Many conventional industry simulators do not implement common sensors utilized in robotics accurately. This thesis focuses on delivering an assessment for NVIDIA's Omniverse Isaac Sim, a robotics simulator and synthetic data generation toolkit, in the context of simulating a real-world scenario. For the evaluation, a stereo camera and a light Detection and Ranging (LiDAR) sensor were chosen to determine the accuracy and performance in creating accurate image and point cloud data. A simple cube tower structure was built, which was later reconstructed within Isaac Sim virtually. The sensors were as realistically modeled as possible and enhanced with post-processing methods. Results showed that, with an approximated reconstructed environment of our real scenario, we achieved good synthetic image quality measures. Additionally, with Isaac Sim's newly added RTX LiDAR sensor, we were able to synthesize similar point clouds of our real scans.

Contents

List of Figures

List of Tables

List of Code

1. Introduction

1.1. Problem Statement

Data synthesis has emerged as a potent tool to accelerate research and development in the domain of AI applications. As data that is used for training is required in great quantity and quality, the generation of desired data sets has become a bottleneck in development due to its crucial effect on the model that is trained. Unfortunately, there are still many issues that remain unsolved in the domain of data synthesis. One of them is the applicability of models trained with synthetic data to real domains. Furthermore, the generation of synthetic data sets is also a challenge itself. Capturing realism for sensors, such as LiDAR and stereo cameras, poses to be a difficult task. Replicating noise and phenomenons, where sensors interact with reflecting or transparent surfaces, are challenges today's industry faces.

1.2. Motivation

To illustrate the importance of data synthesis, consider a scenario, where a task has been given to train an autonomous vehicle (AV). The traditional approach would be, obtaining a car with a handful of sensors and drive on public roads. Currently, developers and researchers often require exceedingly large, accurately labeled data sets. Therefore we would need to spend a vast amount of time on public roads to collect as much data as possible, to eventually train our AV. After data acquisition, humans have to manually label data sets and verify their usability, spending even more time on acquiring an applicable data set for training. At this point, in addition to considerable time loss, we also very likely spent an exceedingly amount of money and exhausted other resources. This problem is not an entirely new one and has been a big challenge for today's industry. To put things into perspective, Waymo an autonomous driving company of Alphabet Inc., announced publicly that their AVs have driven more than 20 million miles on public roads [1].
The scenario mentioned above is merely one among many facing similar data challenges. Today's Industry 4.0 opened many brand-new possibilities to approach these challenges like autonomous robots, simulations, Internet of Things and many more [2]. The most promising solution to the data problem is the one of generating synthetic data. Waymo is one of many industry examples that already make use of synthetic data to train their models. More than 10 billion miles of simulated driving were leveraged to improve the robustness of their self-driving vehicles [3].
Synthetic data is information that has been created artificially using advanced

algorithms. These can then be used as training data, instead of relying on data that has been produced by real-world scenarios. An important aspect of generating synthetic data is, that it fulfills provided conditions. Synthesized data should capture the phenomenon of real data, but it should also include scenarios real data does not encapsulate. The biggest profiteers of synthetic data are modern AI and ML models. A major benefit is that synthetic data can be generated during the training of ML models, alleviating the need to store data. Most of the time synthetic data is used for transfer learning in ML applications. When synthesizing data, one wants to achieve, as previously stated, realism. This implies for example, that synthetically generated images need to look as photo-realistic as possible. However sometimes realism isn't necessary, but this depends on the use case.
Generative adversarial networks (GANs), which are a type of unsupervised learning method, are a powerful tool for generating synthetic data. The idea is, that the discriminator neural network evaluates how realistic the generator creates for example images, and thus fine-tunes itself to minimize the distance between artificial and real data. This type of approach is used a lot in computer vision tasks [4]. Another common trend in generating synthetic data is the combination of a digital 3D world with the physical world, creating what is nowadays called a digital twin (DT). This technology enables replicating and simulating our real world where testing, analyzing, modeling, and predicting becomes much easier and faster [5]. Various applications already exist that for example use Reinforcement Learning (RL) to train a smart robot to navigate through a building. Here robots perceive their surroundings by applying scene reconstruction, object detection, tracking, etc. with virtual sensors.

A range of existing industry tools are available for tackling the above mentioned tasks. Tools such as: Unity and the Unreal Engine are powerful real-time 3D development platforms mostly used by the gaming and film industry to create content. But they are also practised to build DTs to perform physics-based simulations. Researchers Steve Borkman et al. proposed a package for Unity to generate synthetic data for computer vision tasks [6]. NVIDIA Omniverse which is an upcoming and promising industry tool, is a real-time 3D graphics collaboration platform that is attempting on realizing authentic and accurate DTs. With NVIDIA Isaac sim, Omniverse offers a robotics simulation platform that promises photorealistic and physically accurate virtual environments. Additionally, it makes use of Omniverse Replicator to provide tools for synthetic data generation [7]. Automotive companies like Mercedes-Benz used NVIDIA Omniverse to replicate their manufactoring and assembly facilities to accelerate production [8]. Therefore, it can be said that Omniverse is a state-of-the-art tool for Industry 4.0, that implements all of the mentioned characteristics above.

1.3. Outline

The goal of this thesis is to analyze NVIDIA's Isaac Sim performance and capabilities in generating accurate synthetic data. Before diving deeper into the topic of synthetic data and its current challenges, Chapter 2 will provide the required preliminaries. In Chapter 3 of this thesis we will discuss the current state of research regarding data synthesis. However, we will focus more on generated synthetic point clouds and image data, which are common sensory inputs for today's smart robots and AVs to navigate through their environment. The need for an in-depth analysis will be expended upon and what the current research gaps are for generating synthetic data. Furthermore, in Chapter 4 of this work, an overview of the selected approach for analyzing Isaac Sim and the used setup, both physical and virtual, will be presented. In the following Chapter 5 various metrics will be discussed and the final results shown. Finally, in Chapter 6 of this thesis, a summary of this work will be stated, with closing thoughts on how the future work, building up on this thesis, might look like.

2. Preliminaries

This chapter will introduce the fundamentals necessary to comprehend the reviewed literature and methodology of this thesis. Section 2.1 demonstrates the ROS 2 system, which is widely popular in the robotics community and has thus found its usefulness for the topic that this thesis is concerned with. Following Section 2.1, Section 2.2 establishes an understanding on how LiDAR sensors work, such that the virtual implementation within Isaac Sim and other simulators can be comprehended more effectively. Lastly, to gain deeper insights into how the simulation and data synthesis in Isaac Sim functions, Section 2.3 will introduce many of the essential concepts of Isaac Sim utilized in this work. This will be necessary to efficiently demonstrate the selected workflow for the following chapters.

2.1. ROS 2 - Robot Operating System 2

ROS 2 is, as the name falsely suggests, not an operating system. Instead, it is considered a middleware aimed at providing a service for robotics applications. This allows developers to write software not for one manufacturer, but for many different robots since they all share the common interface ROS 2 offers. This also enables to apply written code for ROS 2, to be employed on virtual robots and vice versa [9].
The most crucial part of a ROS 2 system is the ROS graph. All processes, computations, and communication happen in the ROS graph, which is built on an anonymous publish/subscriber model. Communication in the graph occurs between nodes over topics. A node executes computations in the ROS Graph and utilizes the ROS client library to communicate with other nodes. The purpose of nodes is to split up the work in a robotics system, i.e. each node implements a different task in the robot. As already mentioned, communication takes place over topics. Each node can decide what type of messages it receives from other nodes, by subscribing to a particular topic, that is published by another node in the graph. Many standard message types are already provided by the ROS 2 system, which includes pose, image, velocity, and other sensory or motor data.
To write custom code for ROS 2, client libraries provide the functionality to do API calls to the core ROS client library. Language-specific client libraries, such as rclcpp and rclpy for C++ and Python, allow for example nodes that were written in different programming languages to communicate with each other.
Lastly, ROS 2 is augmented by numerous of the tools developed for it. Such tools enable for example to visualize and record data of published topics by

nodes in the ROS graph. One of those tools employed in this work is called rosbag2. The primary functionality of this command line tool is to record and playback ROS 2 messages and store them in a database.

2.2. LiDAR - Light Detection and Ranging

LiDAR is a Time of Flight (ToF) sensor, that utilizes lasers to determine the distance of an object in the environment relative to the sensor. It adopts ToF given that the object's distance is determined by the time taken of the emitted laser pulse traveling through the air until it hits a surface which then reflects to the sensor. Therefore the formula to calculate the distance of an object is as follows:

$$d = \frac{c \cdot t}{2} \tag{2.1}$$

where d is the distance, c is the speed of light, and t is the time spent for the laser pulse to travel to the object and back to the detector [10]. Figure 2.1 illustrates this concept visually.
Besides the distance, a LiDAR sensor also captures the intensity and spherical coordinates of the obstacle. Former, describes the strength of the laser that has returned from the surface it encounters, thus the value represents the reflectivity of an object for the particular wavelength of light. Current state-of-the-art LiDAR sensors employ wavelengths of 905nm and 1550nm [11]. For this reason, laser pulses interact similarly to various materials and structures as does light, i.e. phenomenons like scattering, diffraction, and reflection apply to light rays from LiDAR sensors. Another crucial ability of modern LiDAR technology is to distinguish multiple returns from a single laser pulse. This is possible because of the diameter of an emitted laser pulse. Sometimes only a part of the light ray reflects off an object and the rest keeps traveling until it hits another obstacle. This can result in multiple reflections from a single laser pulse that the detector registers. We speak of First, Second, and Last Returns. With this ability, we can extract different kinds of elevations of numerous objects [12].
As already mentioned above, a LiDAR sensor returns point data in spherical coordinates. In a spherical coordinate system, the radial distance r, elevation angle zenith θ, and horizontal rotation angle azimuth φ describe a point (r, θ, φ). In Figure 2.2, we can observe how such a point can be determined visually in a three-dimensional space. However, converting to the commonly used Cartesian coordinate system can be done with the following equations:

$$\begin{aligned} x &= r \sin\theta \cos\varphi \\ y &= r \sin\theta \sin\theta \\ z &= r \cos\theta \end{aligned} \tag{2.2}$$

There exists a wide variety of use cases where LiDAR has been successfully employed. Accordingly, we find many different types of LiDAR sensors, such as stationary or rotary ones with different wavelength properties and pulse

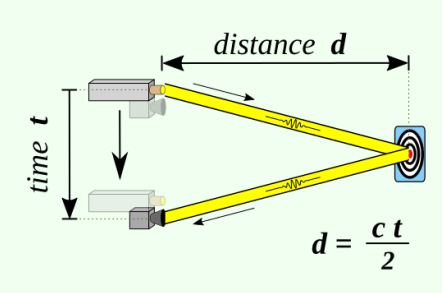


Figure 2.1.: *Visual demonstration of the ToF concept, to calculate distance [14]*

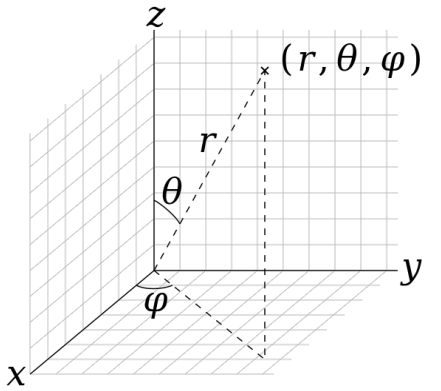


Figure 2.2.: *Spherical coordinates of the point (r, θ, φ) [15]*

times. Apart from most other sensors, acquiring a LiDAR sensor can be very costly, ranging from US$1200 to more than $12000 [13].

2.3. Isaac Sim

As already introduced in the motivation of this thesis, NVIDIA Isaac Sim is a robotics simulation and synthetic data generation toolkit on the NVIDIA Omniverse platform. Isaac Sim offers the ability to create virtual robotic experiments and the necessary tools to build physically accurate simulations with various sensors for data collection [16].
Omniverse, and thus Isaac Sim, use the Universal Scene Description (USD) file format as a backbone to describe scenes. Developed originally by Pixar, USD is an open-source framework for collaboratively constructing large-scale 3D scenes [17]. It allows us to define geometries, materials, lights, cameras, and more. The root of a scene described within a USD file is the stage. As an abstraction for a scene graph, a stage composes all elements and references within a scene. It allows for quick traversal of the scene and efficient data queries. The primary elements inside a USD stage are called prims. Prims can contain other prims and properties representing important data, e.g. a Mesh with properties defining its vertices and faces. If a USD scene is saved, it can be loaded or referenced in another scene.
Within a USD file, materials can be defined and assigned using the Material Definition Language (MDL). Developed by NVIDIA, the open-source shading language is specifically designed to define materials and the behavior of light [18]. Therefore, it serves a different design purpose than a shading language such as GLSL, whose main motivation is to describe shaders for real-time graphic applications. However, MDL still supports traditional computer graphics techniques such as normal mapping and cut-outs.
Included heavily in the system architecture of Isaac sim, are extensions and Omniverse Kit. To build or interact with objects within a scene in Isaac sim,

NVIDIA offers besides the standard USD API by Pixar, Omniverse Kit, which provides simple commands to accelerate the process of creating robots or simplifying commonly used USD API tasks. Extensions on the other hand, as the name already suggests, extend the capabilities of Isaac Sim. ROS 2 Bridge and Replicator are core extensions of Isaac Sim to achieve the full power of realizing accurate DTs. Latter is NVIDIA's synthetic data generation tool for training CV AI and other ML models [19]. By generating a render product with Replicator and connecting it to one of the sensors Isaac Sim offers, the renderer will produce realistic 3D graphics, thus enabling us to extract synthetic sensory data. Additionally, by attaching annotators to our render products, we are able to collect ground truth annotations such as distance to the image plane, instance segmentation, point cloud, camera parameters, and many more [20]. With writers, Replicator offers compact and easy-to-use modules that process different kinds of annotations from render products and produce specific data formats that for example can be used for training ML models. Replicator is built upon PhysX, USD, and MDL which means one can easily manipulate material, light, sensor, or geometrical properties in Isaac Sim. One of Replicator's main purposes is the realization of domain randomization, which will be explored more in-depth in Section 3.4.
Lastly, Isaac Sim offers a handful of tools for researchers and engineers to collaboratively work on a project. One of which is Omniverse Nucleus, a database where multiple users connect to exchange and access USD and MDL files. Changes in the database are transmitted in real-time to clients under a publish/subscriber model [21].

3. Related Work

In this chapter, a deeper look at the current state of research and development regarding synthetic data and its methods of generation will be shown. However, this work primarily investigates the accuracy of synthetic images and synthetic point clouds. Due to this fact, a detailed investigation of these types of synthetic data will be presented in Section 3.1 and Section 3.2, instead of offering a general outlook. For each of the presented data types, a short overview of applications and state-of-the-art data sets will be introduced, along with current drawbacks. The problem of closing the domain gap between synthetic and real data proposes to be the biggest challenge for researchers and has thus gained the most focus. Therefore we will investigate current data synthesis methods and their issues in Section 3.3. Section 3.4 will end this chapter by illustrating the ongoing problems in closing the domain gap and its proposed solutions.

3.1. Synthetic Images

As already mentioned synthetic images and synthetic point clouds will be the main focus for analysis in this thesis. Concerning the former, researchers, especially in the CV community, have so far made extensive use of it.
Here problems such as optical flow estimation, stereo image matching, object detection/recognition, and tracking, benefit greatly from correctly labeled images. In many cases, it is advantageous to use already modeled and well-studied synthetic data sets. In a survey by Sergey I. Nikolenko a great overview is given of synthetic data sets for various applications, that mainly affect computer vision tasks, which involve synthetic images predominately [22]. For example, Dosovitsky et al. published a synthetic data set Flying Chairs that illustrates 3D modeled chairs on real backgrounds, intending to tackle the optical flow problem [23]. Here, like in many cases, the use of synthetic data to train models outperformed previously state-of-the-art models that traditionally were trained on real images. As mentioned in the introduction of this thesis, the discussion of synthetic data requiring to capture realism, has been further investigated for the use case of optical flow estimation by Nikolaus Mayer et al. [24]. Results showed that realism in this case is not strictly necessary, although knowing certain camera parameters, such as lens distortion or blur may improve training. In another study by Yair Movshovitz-Attias et al., the question if photo-realism is practical for training models has been explored in greater depth [25]. Their results illustrated that for view-point estimation in a 3D environment, using complex materials decreases the error of estimation. Further investigating the need for realistic synthetic images, researchers

S. Hartwig and T. Ropinski examined the impact of reflecting materials in synthesized images [26]. We observe for the case of synthesized images that realism is dependent on the use case and that it can promise improvements for many scenarios, as we will see in further work done by other researchers.
One of the most popular computer vision tasks, which benefits from the use of synthetic data, is the one of face detection and eye tracking. A great effort has been made to approach this challenge and with the classic and popular framework of the Viola Jones face detector by P. Viola and M. Jones, a breakthrough has been made in 2001 [27]. This method for face detection, which makes use of the learning algorithm AdaBoost, has been a building block for much research to come. Since then, countless other methods have been developed and a sufficiently large amount of manually labeled data sets have been collected. The problem of face detection is considered to be mostly solved by the academic community. However common problems are face expressions, lighting, occlusion, viewpoints, and bias. Thus collecting enough real data for all scenarios is unlikely. Synthetic data can be used to clear up the remaining challenges. In a recent publication by Erroll Wood et al. it has been shown that using synthetic data alone, in this case synthetically generated faces, match or even outperform models trained on real data [28]. In this particular context, the faces have been photo-realistically rendered, with a wide variety of randomized faces, exhibiting furthermore the importance of realism in synthetic data.

3.2. Synthetic Point Clouds

For many applications, a LiDAR-based sensor is very beneficial for various deep learning tasks, such as detection [29] and segmentation [30]. AVs, drones, and other autonomous systems are one of many popular safety-critical applications using LiDAR successfully, that require a steady stream of precise point cloud data to tackle such challenges. However as illustrated in the beginning of this thesis, obtaining such data sets is a costly operation. Acquiring equipment alone, that means a LiDAR sensor, can be very expensive, as demonstrated in Section 2.2.
The generation of accurate and labeled synthetic LiDAR point cloud data has therefore been a big challenge for researchers. Many synthetic point cloud data sets exist, as they did, however in much greater quantity, for synthetic image data. One such data set is the publicly available SynthCity large-scale synthetic point cloud data set [31]. In this instance, the researchers D. Griffiths and J. Boehm constructed an urban environment within Blender and simulated a laser sensor with the help of a plugin for Blender to then receive synthetic point cloud data for their virtual setting. The intention is to provide a large-scale data set for pre-training a classification or segmentation model. In another work by Fei Wang et al., CARLA (CAR Learning to Act) [32], an open-source simulator for autonomous driving, has been used in combination with a hand full of digital assets, to generate synthetic point cloud data [33]. It was shown that using synthetic point cloud data can improve overall per-

formance and generalization efforts for deep learning models. This is the case when real data is limited in its scenarios, which is the reason why synthetic data in such cases aids in achieving better performance.
However, many of these approaches suffer from a lack of realism. A common obstacle is the need for high-quality 3D assets with accurate materials so that scenarios with phenomena such as reflectance and transparency can be realistically modeled. Researchers V. Zyrianov, X. Zhu, and S. Wang, proposed their LiDARGen generative model, to produce asset-free LiDAR point cloud data, as an attempt to alleviate the problem of needing realistic 3D assets [34]. Another issue is that the majority of LiDAR simulations are not accurate when it comes to simulating LiDAR effects, namely noise or raydrop. Latter, is the effect when laser rays hit a glass object, the rays are refracted and often don't return. Many simulators implement virtual LiDAR sensors with a ray casting method to replicate the ToF effect. This means, that on each ray casting step, we check if the ray collided with an object in the 3D environment. To accelerate the raytracing process, it's common to test collision on bounding volumes of objects. This limitation can often cause, mostly transparent or specular objects, to appear fully opaque, since they only detect a hit on the bounding box and return the distance of the object to the sensor. This is one potential outcome of overfitting a model with data produced this way. Since the virtual LiDAR sensor produces too precise point cloud data, which doesn't resemble real-world data at all.
The underlying mentioned problem of correctly simulating noise and raydrop has since been the next milestone for achieving accurate synthetic LiDAR point cloud data. Benoît Guillard et al. have proposed a data-driven approach for simulating realistic LiDARs [35]. With a model called RINet, the researchers showed that phenomenons like raydrop can be learned from real data. When applied to naively ray casted synthetic point clouds, improvements in realism can be observed, by dropping points that would have not returned to the sensor. In another work, by Sivabalan Manivasagam et al., a similar approach was chosen [36]. Here a neural network called LiDARsim, learns the sensor's raydrop characteristics on real data, to then create realistic virtual LiDAR point cloud data. To achieve the noisiness of real LiDAR rays, researchers Gusmão G.F., Barbosa C.R.H., and Raposo A.B. suggest applying Gaussian noise to represent error in the measurement [37].

3.3. Generating Synthetic Data

So far we discussed the specific types of synthetic data and available data sets, that are most popular and desired by researchers and today's industry. What remains to be investigated further, is the generation of synthetic data. One of the simplest and most common approaches is to augment already existing real data. Straight-forward augmentations like rotating an image already aid in increasing the size and variety of an existing data set. However, data augmentation does not create new synthetic data, it is a technique to modify existing real data. Another method that can be considered as synthetic data

generation is to cut and paste existing data together. Researchers Dwibedi et al. demonstrated in their paper the practical usage of such an approach in the context of object detection [38].
Regardless, the most popular and common method is the usage of a virtual 3D environment to produce data for a specific scenario. Many simulators exist for various settings that offer photorealistic rendering or accurate sensor simulations to provide a means of generating synthetic images or 3D point clouds of virtual 3D objects. Simulators are especially favored to approach problems such as Simultaneous Localization and Mapping (SLAM) and various other RL tasks. The reason is, having full control over the simulated environment and sensors, that makes detailed and precise analysis possible. In the introduction of this thesis, we shortly introduced two powerful 3D game engines: Unity and Unreal Engine as industry standard tools for creating video games and other types of content. These engines already provide many tools for rendering photorealistic 3D scenes and can be used as building blocks to successfully generate synthetic data and build simulators for specific cases. For example, the urban driving simulator CARLA is based on the Unreal Engine 4 which provides sensors such as RGB cameras, depth cameras, optical flow cameras, LiDAR sensors, and many more. However, some researchers prefer to build their own custom engines, such that certain desired computations are faster. Gazebo [39] is a custom-built robotics simulator using OpenGL that primarily aims to simulate common robotics sensors and supports the ROS 2 system. It has been an industry standard for robotics simulation for a long time, because of its realistic physical engine and its integration of ROS. As an alternative for Gazebo, MuJoCo (Multi-Joint Dynamics with Contact) [40] is an open-source physics engine also for robotics simulations. Many of the existing engines for simulations can also be further categorized into simulators for outdoor and indoor scenes. Simulators such as CARLA or AirSim [41], a flight simulator created by Microsoft, are known for their strengths in outdoor scenarios but can be used for indoor scenes as well.
Lastly, a new trend is to use GANs to produce synthetic data, or to refine it, making it more realistic with smart augmentations. To create new synthetic data, GANs first need an input of real data to approximate new data that resembles the underlying phenomenon of the input. Researchers such as Zhao et al. used GANs successfully to improve the realism of synthetically created faces, by applying their model DA-GAN (Dual-Agent GAN) [42]. On the other hand, directly creating synthetic data from random noise with GANs, is considered to still be a hard challenge to solve and is currently one of the limitations GANs have. Additionally, computation and training can be costly and time-consuming, moreover, problems such as non-convergence and mode collapse are still problems from which many GANs suffer [4]. However, in this thesis, we will not further investigate the capabilities of GANs. Instead, we will examine NVIDIA's Omniverse Isaac Sim ways of solving the above-mentioned problems of simulators generating accurate synthetic image and point cloud data.

3.4. Research Gap

Synthetic data and its generation are without a doubt one of the crucial techniques used in today's state-of-the-art methods for numerous computer vision, RL, and other ML tasks. Its promise of providing limitless unique and labeled data is appealing to many researchers and has thus been researched extensively for many years. It has so far found its usage in fields like healthcare, business, education, AI-generated content, Natural Language Processing, and more [43].
Nevertheless, as it has been pointed out in the sections on synthetic image and point cloud data, there are still many problems left unsolved. Challenges, like how well synthetic data represents the real world, are important for ML models to be robust. Researchers have shown that synthetic data encounters problems in embodying the real underlying phenomenon of real data and that it includes data bias [44]. Another issue can be that the generated synthetic data overestimates the real world, ensuring an overfitted model [45].
Many of these difficulties can be associated with the domain gap between synthetic and real data. The gap can even be further divided into two parts: the appearance and content gap [46]. Former, refers to the difference between the real and synthetically rendered data which is influenced by materials, assets, and rendering systems used. The content gap, on the other hand, is concerned with the difference between the real and synthetic domain. That is, how diverse is the synthesized data set in comparison to the variety of real data that can occur. The attempt to close the domain gap, for both subcategories, has so far attracted many researchers in the academic community. Existing methods such as including real data in training processes, which originally only contained synthetic data, or fine-tuning a model with real data after pre-training it with synthetic data are common ideas to optimize. However, improving the quality of synthetic data such as increasing photorealism has so far been mostly focused on. In terms of synthetic images, the goal is to use highly advanced rendering techniques to improve data quality. As discussed in the section on synthetic images, it is still unclear for some applications if photorealism is necessary. Nevertheless, it has been shown, that tasks such as face detection and 3D pose estimation greatly benefit from it. In the context of the domain gap between synthetically generated and real point cloud data, many effects like raydrop or LiDAR noise are sometimes not realistically modeled, considered as a post-processing step or even not regarded at all. Thus they propose to be challenges in closing the domain gap. Note that the computational resources necessary and time spent are often a tradeoff when synthesizing highly realistic data.
A popular method on the rise for lowering the domain gap, more specifically the content gap, is domain randomization. The idea of domain randomization is that, given the data distribution μ_{real} for our real data, we want to approximate or ideally cover μ_{real} with μ_{syn} the synthetic data distribution. Realistically μ_{syn} will never be equal to μ_{real}. Nevertheless, to achieve a robust model trained on synthetic data, we only need to make it diverse enough.

By randomizing properties such as the 3D/2D position, quantity, shape, and material of objects or changing the light intensity, position, and orientation, we cover a lot of different scenarios for our synthetic data set. In the work of J. Tremblay et al., it was demonstrated that with the help of domain randomization, a model trained for object detection performed equally well as with traditionally used real data sets [47]. By randomizing textures of simple geometric shapes, it was concluded that even photorealistically rendered images were outperformed.

4. Methodology

This chapter is dedicated to discussing the selected approach in assessing the accuracy and performance of Isaac Sim. Section 4.1 presents in a high-level manner the design principles under which we want to test Isaac Sim as a simulator and synthetic data generation tool. For this purpose, a system has been built that can be separated into three main components. Moving beyond Section 4.1, Section 4.2 dives deeper into the realization of a physical setup and data acquisition. Additionally, Section 4.3 will illustrate the usage of Isaac Sim to simulate our real-world scenario from Section 4.2, by demonstrating the modeling and simulation approach. Finally, we will exhibit post-processing methods in Section 4.4.

4.1. Methodological Overview

The initial idea to asses NVIDIA's Isaac Sim's proficiency in data synthesis is the creation of an accurate DT within this simulator. With this in mind, we want to audit Isaac Sim for two key areas that we explored in Chapter 3. First, for the case of synthetic images, we want to examine how well we can model and photorealistically render our scene so that it matches our real data. Furthermore, as our second category, we want to check if, Isaac Sim is subject to the same issues for virtual LiDAR sensors as it does for other simulators and data sets. In other words, an effort will be made to realistically model our real LiDAR sensor within Isaac Sim and analyze both outputs against each other.
To achieve such an evaluation against synthetically generated and real data, we want to compare the similarity of each real data attribute to its synthetic counterpart. In other words, we will feed Isaac Sim position and orientation data recorded during real scans and replicate the process virtually while synthesizing data. The fundamental process of this thesis and the system architecture can be split into three modules. The very first one entails everything with the physical world. This means, that questions such as: which sensors to use, what setup to design (for it to be later reconstructed virtually), which features are necessary, and how we acquire them, are all topics associated with the first step of this pipeline. In Section 4.2 we will explore in more detail how all of these problems have been approached. Playing a more integral part of this architecture, the second module involves data synthesis and virtual recreation within NVIDIA's Isaac Sim. To go more into detail, it will be discussed how Isaac Sim operates with our given scenario, what challenges have been encountered, and additionally what post-processing methods have been used. Section 4.3 will, with the knowledge of the previous section about the physi-

cal setup, get to a greater extent about the mentioned properties of this part in the research process. Finally, the last component and Chapter 5 of this work is related to the assessment of Isaac Sim's capabilities in synthetic data generation. The choice of metrics for synthetic to real image and point cloud comparison will be examined with the inclusion of performance measurements. Figure 4.1 illustrates in a high-level manner the proposed system architecture used for this work. In this context, we can see that the recorded data from our physical setup will be used as an input, as it is typical for a DT. After a careful process of extracting important features for our needs, we begin by providing position and orientation data to our synthetic data generation module, which in this case is Isaac Sim. By additionally rendering our reconstructed scene and modeling our sensors, we can then proceed by replicating our real scenario virtually and thus synthesize data with our digital sensors. After the generation of our data, we will compose a series of calculations to assess the accuracy of our simulation and as a result, we will receive measurements for further analysis.

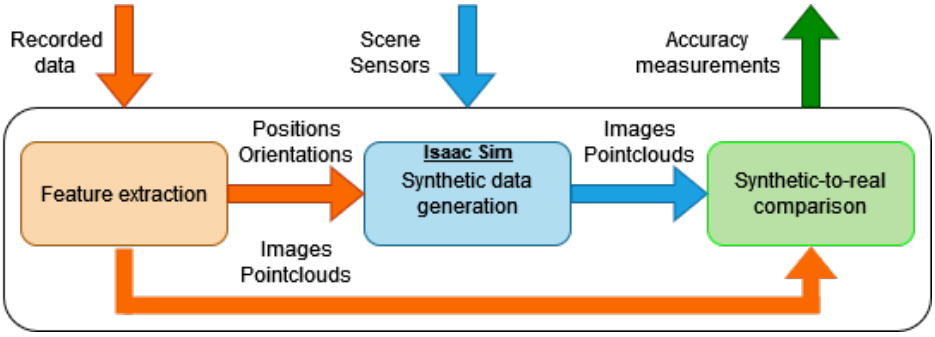


Figure 4.1.: *System architecture. Components colored in orange represent the parts dealing with real data. Blue components involve synthetic data. Lastly green depicts the process of comparing synthetic to real data*

4.2. Physical Setup

Directly following up the overview of this chapter, will be the section of the physical setup. Here we will be exploring in more detail how the first component of the system architecture has been realized. More specifically it will be demonstrated under what research goals the real setup was chosen to test Isaac Sim. Subsection 4.2.1 presents the selected sensors for data collection and testing environment where all experiments have been conducted. In Subsection 4.2.2 the process of data acquisition and feature extraction with the help of ROS 2 and interpolation will be illustrated.

4.2.1. Experimental Environment and Sensors

To capture raw RGB and depth images, the ZED 2 stereo camera has been selected for this work. In the context of collecting point cloud data, the Li-

DAR RS-Helios-32-5515 3D Laserscanner was chosen. The characteristics of both sensors can be found in Table 4.1 and Table 4.2. Both sensors have been mounted on a remote-controlled mobile robot, as can be seen in Figure 4.2. The open-source robotics middleware ROS 2 is applied, to collect sensor data and to control the motors of the robot. Connecting the ZED 2 with the ROS node of the ROS 2 wrapper for the ZED SDK and the RS-Helios LiDAR sensor with the ROS node of the ROS 2 wrapper for the RoboSense LiDAR SDK, enables us to publish all of the necessary data into the ROS ecosystem. The sensors and their parameters remained unchanged throughout all tests.
During this thesis, several experimental design choices have been made for the purpose of having an easy-to-replicate setup while still including important testing parameters for the virtual sensors of Isaac Sim. As we already explored in Chapter 3 of this thesis, many domain gap problems arise due to the lack of realistically implemented sensors or rendering techniques. For this reason, the goal is to evaluate how NVIDIA's realization of a simulator for generating synthetic data, in the context of images and point clouds, operates. Therefore we want to include assessments for how Isaac Sim's camera and LiDAR sensor interact with different material properties, such as transparency and high specularity. Consequently, our initial hypothesis according to the reviewed literature, is that the generated synthetic data will be too detailed.

ZED2 Stereo Camera

parameters	values
Baseline	120mm
Focal Length	2.12mm
Field of View	110°(H) × 70°(V)
Aperture	$f/1.8$
Output Resolution	896 × 512

Table 4.1.: *Specifications of the ZED2 stereo camera*

LiDAR RS-Helios-32-5515

parameters	values
Wavelength	905nm
Range	0.2m - 150m
Range precision	±2cm @ 1m - 100m
FoV Horizontal/Vertical	360°/70°
Horizontal/Vertical resolution	0.2°/1.33°
Rotation speed	10Hz (600rmp)
Number of Emitters	32

Table 4.2.: *Specifications of the LiDAR RS-Helios-32-5515 3D-Laserscanner*

Considering the above-mentioned testing parameters, the experimental setup has been built to satisfy these as much as possible with the given resources. First, the structure to scan with our real sensors and later test the virtual sensors with has been the main concern for this work. For this case, the idea

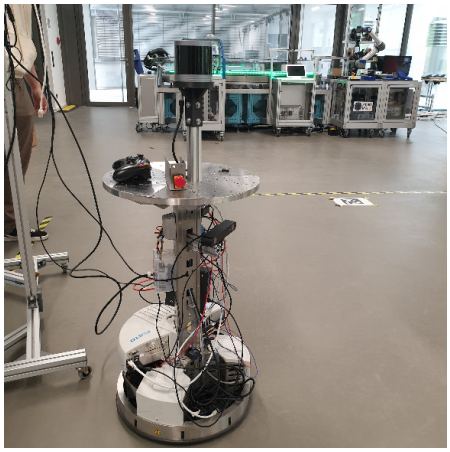


Figure 4.2.: *Mobile robot used for scans. Located on the top is the LiDAR sensor and in the middle the ZED 2 camera*

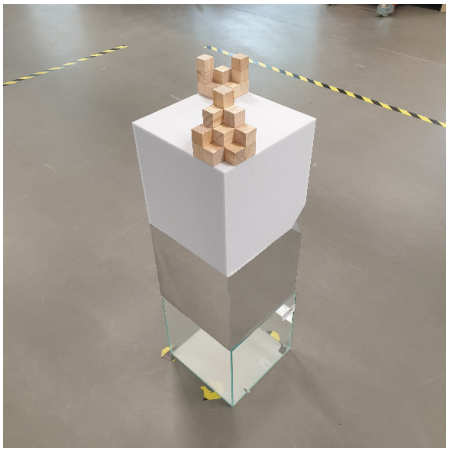


Figure 4.3.: *Simple cube tower construction. Inclusion of transparent and reflecting materials as test cases*

of using a simple structure is important, such that the problem of modeling a complex structure within Isaac Sim will be alleviated. In addition, less room for modeling errors is given, since these can impact the realism of the resulting synthetic data of our virtual sensors. Therefore a simple cube tower structure has been built, consisting of three 30x30x30cm and thirty 3x3x3cm cuboids. The larger cubes have been stacked on top of each other, while the thirty smaller cuboids have been utilized to construct two small structures on top of the tower. However, two of the large cuboids, namely the middle and top ones, are not actual cubes. Geometrically speaking, are those two structures heptahedrons, which are polyhedrons with seven faces. The intention behind the two small cube structures on top of the tower and the heptahedrons is to give the structure some sense of complexity, while still making it easy to replicate it inside Isaac Sim. A demonstration of the final construction can be seen in Figure 4.3. Immediately apparent from the presented figure, are the choices of materials. From our review of related literature to synthetic data and our testing goals for Isaac Sim, the inclusion of four different materials has been made. Starting with the smaller cuboids that form two structures on top of the tower, the choice of oakwood as a simple testing parameter for diffuse surfaces has been settled upon. Furthermore, the heptahedron on the top, which is made out of foam, serves as another test for diffuse surfaces. However, due to the nature of foam, we also include an assessment of materials with subsurface scattering properties. In the middle of the tower, a decision has been reached to incorporate materials with high specularity, Thus, the heptahedron in the middle resembles an almost perfect mirror on all five faces. Lastly, as an evaluation for transparency, the bottom cuboid is a glass cube which is hollow on the inside.
An occurring problem during experiments has been the factor of natural light

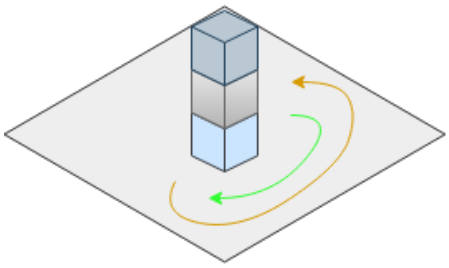


Figure 4.4.: *Illustration of a path taken to scan the cube tower with various distances*

Figure 4.5.: *Cube tower inside the isolated structure with illuminating light source*

and other sources of light illuminating complex geometry surrounding our cube tower. The reason for this being an issue is that reflections and other illumination factors from objects around the cube structure unpredictably influence the lighting. It proves to be a challenge to position light sources within our virtual scene since determining the intensity and position of all light sources is unfeasible. As a result, an environment that isolates the constructed cube tower from almost all complex geometry surrounding it is desired, since this gives again less room for modelling errors. Therefore, the challenge of reconstructing the surroundings of the cube tower, which would be visible by our camera, will be less of a problem. Thus, an encapsulating environment has been fabricated with dimensions of 2.90m in length/width and 1.50m in height. Six partition walls, consisting of thin cardboard placed within oak frames, were placed together to form the back, left, and right isolation walls for the environment. The front has been left open, in order to see and drive the robot inside of it. At last, thin cardboard has been used once more to seal the constructed enclosure, preventing many of the unwanted complex structures from illuminating the scene with reflections. Unfortunately, the thin cardboard used as partition walls is only 1m tall, and thus leaves 50cm of the back wall open for some of the original environment outside the isolating surroundings to be seen. Additional problematic inconsistencies of the encapsulation structure, which should be kept in mind, are the open front view and the material the structure has been made out of. For illuminating the cube tower from inside the controlled environment, a tubular LED light source with a dimension of 90x2x2cm has been selected and placed in the back right corner. Measuring the light source with the MT-912 Light Meter yields 29.6klx. The final testing environment, with the light source and cube tower, can be seen from the inside in Figure 4.5.

4.2.2. Data Acquisition and Feature Extraction

As already introduced in the methodological overview of this chapter, a process of data acquisition and feature extraction is applied. Illustrated in Figure 4.4, the remotely controlled mobile robot, seen in Figure 4.2, has been utilized to collect data from our cube tower setup, which has been discussed in detail in the previous section of this chapter. The robot was instructed to always face the cube construction with the stereo camera and multiple rounds have been driven with various distances to broaden our variability in the acquired data set.

Recorded ROS 2 topics

Name	Type	Amount
/zed2/zed_node/pose	geometry_msgs/msg/PoseStamped	3281
/zed2/zed_node/left/camera_info	sensor_msgs/msg/CameraInfo	6771
/zed2/zed_node/left/image_rect_color	sensor_msgs/msg/Image	3332
/rslidar_points	sensor_msgs/msg/PointCloud2	1161

Table 4.3.: *Table of topics that have been collected during scans*

ROS 2 has been employed on the mobile robot to collect sensor, pose, and configuration data from both of our applied sensors. Utilizing rosbag2, ROS 2 records and stores its published topics in a database, with the functionality to replay this data again. The list of all topics and the amount of data collected are listed in Table 4.3. In this instance, the data was stored in an SQLite database, which saved each topic in the table with a respective timestamp. Since all of the data stored is encoded by ROS 2, a parser has been written that deploys rclpy (ROS Client Library for Python) to deserialize messages while querying for the according to topic IDs. The code block seen in Code 4.1 first queries timestamp and data from the table *messages* where the topic id matches our desired topic. Afterwards, rclpy is being used to deserialize all messages from the collected rows of our query result.

```
import sqlite3
from rclpy.serialization import deserialize_message

topic_id = 0 # Or any desired topic id

conn = sqlite3.connect(bag_file)
cursor = self.conn.cursor()
query = f"SELECT␣timestamp,␣data␣FROM␣messages␣WHERE"
        + "topic_id␣=␣{topic_id}"
rows = self.cursor.execute(query).fetchall()
deserialized_data = [deserialize_message(data, topic_name)
                     for timestamp, data in rows]
```

Code 4.1: *Example code on how to query and deserialize ROS 2 messages*

In the context of our scenario, ROS 2 collected more image data than pose information, where occasionally timestamps between pose and image don't match. The problem of no matching timestamps becomes an even greater

problem when observing the amount of collected point cloud data in contrast to the pose data. In this case not a singular timestamp from the point cloud data set can be found in the trajectory information. This indicates to be a potential difficulty when we want to match synthetic and real data. If we provide position and orientation information to our DT in Isaac Sim, we will possibly synthesize data that doesn't capture the actual position and rotation when the sensors in our real setup record data. A straightforward solution would be to drop real data for which we don't have any pose information. This leads however to a loss of approximately 10% of our collected image data and a total loss of our point cloud data. We want to assemble as much data as possible to achieve robust results for our selected metrics. Therefore to overcome the problem of missing path information for some sensory data, we need to apply interpolation. Since interpolation can be considered as data synthesis, we will additionally keep a list of the original trajectory data without interpolation, to assess the accuracy of data captured with and without interpolated poses. In other words, instead of interpolating poses for missing timestamps, we will just assign the pose information to the timestamp that is closest to it. We define the sets as P and $\hat{P}$ respectively.
The idea of our interpolation approach is to approximate the position and orientation between timestamp intervals, from which we know that the interval start/end pose and sensor timestamp match. In other words, we want to generate the missing pose information for a sensor timestamp that is in between such an interval and is without a match. Given two finite sets of tuples for our path and sensory data:

$$\begin{aligned} P &= \{(t_0, (p_0, o_0)), (t_1, (p_1, o_1)), ... | \forall j > i : t_i < t_j\} \\ S &= \{(\tau_0, d_0), (\tau_1, d_1), ... | \forall j > i : t_i < t_j\} \end{aligned} \tag{4.1}$$

where, t_i, τ_i are timestamps, d_i sensor data, and the tuple (p_i, o_i) representing a position in XZY coordinates and orientation as a unit quaternion, at time i. Let $(\tau_\lambda, d_\lambda) \in S$ be a sensor tuple without a match, i.e. $\forall (t, (p, o)) \in P : t \neq \tau_\lambda$, for which we want to interpolate position and rotation of the robot. In order to interpolate the pose of a robot at τ_λ the sensor timestamp must be in between the start and end of the path recording, i.e. $t_0 \leq \tau_\lambda \leq t_{|P|}$. This means if we have a sensor timestamp without a match that does not fulfill this condition, we can not interpolate position and orientation for this scenario and are left with the option to discard this data. When disposing of position and orientation information at time 0, i.e. $(t_0, (p_0, o_0))$, we need to adjust our path data set to assume a new origin point and orientation of our trajectory. Let $(t_i, (p_i, o_i)) \in P$ be our new origin position and orientation. First, we simply translate the positions in our trajectory by subtracting p_i from every position coming after it, i.e. $p_j - p_i$ where $j \geq i$. Secondly, we need the inverse of our rotation o_i, which is simply the conjugate of a unit quaternion, and multiply it with every rotation after it, i.e. $o_j o_i^{-1}$ for $j \geq i$. Once we do all necessary translations and rotations, we discard all tuples coming before $(t_i, (p_i, o_i))$. After adjusting our path data set, we are ready to apply interpolation for sensor tuples without position and orientation data.

The goal now is to find an index k such that the interval $t_{k-1} < \tau_\lambda < t_{k+1}$ holds, so that we can generate a new tuple $(t_k, (p_k, o_k))$, where $t_k = \tau_\lambda$ and p_k, o_k are interpolated. The position is linearly interpolated the following way:

$$\begin{aligned} 0 \leq a &= \frac{\tau_\lambda - t_{k-1}}{t_{k+1} - t_{k-1}} \leq 1 \\ p_k &= (1-a)p_{k-1} + ap_{k+1} \end{aligned} \tag{4.2}$$

Since we are using unit quaternions to describe rotations of our robot, spherical linear interpolation (SLERP) was applied to interpolate o_k subsequently:

$$\begin{aligned} 0 \leq a &= \frac{\tau_\lambda - t_{k-1}}{t_{k+1} - t_{k-1}} \leq 1 \\ \theta &= \arccos\left(\frac{o_{k-1} \cdot o_{k+1}}{\|o_{k-1}\|\|o_{k+1}\|}\right) \\ o_k &= \frac{\sin((1-a)\theta)o_{k-1} + \sin(a\theta)o_{k+1}}{\sin(\theta)} \end{aligned} \tag{4.3}$$

If however the quaternions are very close to each other, we simply linearly interpolate, as we did in Equation 4.2. We insert this newly generated tuple $(t_k, (p_k, o_k))$ into the set P at index k and repeat this process for sensor tuples for which we don't find a match. Considering that our path P is a trajectory, better interpolation methods could have been used, e.g. a cubic-spline or Bézier curve-based interpolation method. However, since the motion of our robot does not involve any abrupt changes and the intervals used to interpolate poses do not have significant time differences, SLERP, and linear interpolation are more than satisfying for our application. Ultimately after collecting, extracting, and processing the trajectory of our robot, it is ready to be used as input to simulate the path of our virtual robot inside Isaac Sim.

4.3. Virtual Setup

In this part of the thesis, the implementation of the second component of our pipeline will be explained. Subsection 4.3.1 demonstrates the process of modeling our sensors and recreating our scene from Subsection 4.2.1. Here we will apply many of the tools Isaac Sim offers to achieve as much realism as possible and additionally explain the steps taken to reduce modeling errors. Furthermore, Subsection 4.3.2 discusses in more detail how Isaac Sim was employed to simulate and generate synthetic data from our virtual scene with the given input data. Lastly, Section 4.4 exhibits applied post-processing methods on synthesized image and point cloud data.

4.3.1. Scene Recreation

The effort of recreating our real scenario, which was demonstrated thoroughly in Subsection 4.2.1, can be separated into three modeling tasks. We first want to utilize USD and MDL to build the geometric structure and assign materials

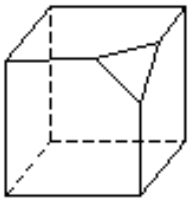


Figure 4.6.: *Heptahedron with 7 faces, 10 vertices and 15 edges [48]*

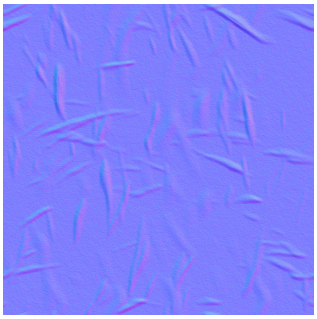


Figure 4.7.: *Normal map applied to the Mirror.mdl material [49]*

for our cube tower. Secondly, we want to model the encapsulating environment, that isolated our real cube tower from complex geometry. This also includes modeling the lights we utilized to illuminate the scene. Finally, since no CAD model of the utilized robot was available, we need to model a simple structure of our mobile robot, to later place our virtual sensors on.
By initially choosing a simple cube tower for our real scans, the task of modeling became less challenging. Since our structure consists of three 30x30x30cm and thirty 3x3x3cm cubes, no complex geometries have to be modeled manually or imported. However, as it was illustrated in Subsection 4.2.1, two of the three cubes, namely the mirroring middle and diffusing top cube, are not perfect cuboids. We are dealing with heptahedrons, which are polyhedrons with seven faces as can be seen in Figure 4.6. Therefore we need to utilize USD and create such a mesh manually. Code 4.2 shows how a heptahedron mesh can be defined inside the human-readable usda file format. The USD file for the heptahedron is referenced twice in the cube tower USD scene. The rest of the structure has been built with the GUI of Isaac Sim since the remaining construction only consists of cuboids. In the context of materials, NVIDIA already offers templates that include physically based glass, multi-lobed materials for dielectric and non-dielectric materials, skin, hair, liquids, and other materials requiring subsurface scattering or transmissive effects [50]. Hence, instead of defining our materials with MDL, we used the following presets: Oak.mdl, and Styrofoam.mdl, Mirror.mdl, and Clear_Glass.mdl. Since our real mirroring cube has some slight surface deformation and distortion, we included a normal map of the Mirror.mdl material and set the roughness parameter to 0.03. The employed normal map can be seen in Figure 4.7, with a map strength of 0.06.
Going further with recreating our real scene within Isaac Sim, modeling our encapsulating environment is the next step. Separating every geometry of our constructed surroundings, we observe that only cuboids are utilized again. As a result, all of the mentioned parts of the environment in Subsection 4.2.1, have been geometrically modeled 1:1 with Isaac Sim's GUI. To give the structure a realistic look, as an attempt to close the appearance gap, the following material presets have been utilized: Oak.mdl, Paper.mdl, Gypsum.mdl. In the

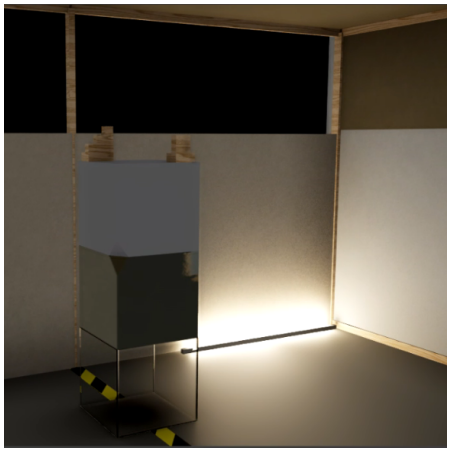


Figure 4.8.: *Virtual cube tower, isolated by walls with a single light source*

Figure 4.9.: *Simple reconstruction of the mobile robot utilized for scans*

case of modeling the light source, a cylinder light was chosen, as its tubular lighting is beneficial for simulating our light tube. All listed parameters in the previous section of this chapter for the light source have been transferred to the virtual version and the ambient light intensity has been set to 0.91.
Lastly, after recreating our cube structure and isolating the environment, we need to model our mobile robot. Since the LiDAR and stereo camera are both placed on the robot, it won't be visible in the point cloud and image data. The remotely controlled robot becomes visible in raw RGB image data exclusively because of the mirroring middle cube. However, due to the distance and angle of the camera, along with the surface roughness and deformation of the mirror cube, the robot is never to be seen fully or clearly. Hence, no exact replication of the robot is needed, rather only an approximation. Most of the robot has been modeled again with the GUI Isaac Sim offers since the majority of it was constructed with cylinders and cuboids. The final robot geometry has then been assigned two materials, namely Plastic.mdl and Aluminum_Polished.mdl. The resulting cube tower construction inside the isolating environment and the virtual mobile robot can be seen in Figure 4.8 and Figure 4.9.

```
#usda 1.0
(
    metersPerUnit = 1
    upAxis = "Z"
)

def Xform "Heptahedron" (
    kind = "component"
)
{
    def Mesh "HeptahedronMesh"
    {
        int[] faceVertexCounts = [4, 4, 4, 5, 5, 5, 3]
        int[] faceVertexIndices = [0, 1, 2, 3, 0, 1, 5, 4, 1, 2,
                                   6, 5, 3, 2, 6, 7, 9, 0, 3, 9, 8,
```

```
                                4, 4, 8, 7, 6, 5, 8, 7, 9]
        point3f[] points = [(-0.15, 0.15, -0.15),
        (-0.15, -0.15, -0.15), (0.15, -0.15, -0.15),
        (0.15, 0.15, -0.15), (-0.15, 0.15, 0.15),
        (-0.15, -0.15, 0.15), (0.15, -0.15, 0.15),
        (0.15, 0.075, 0.15), (0.075, 0.15, 0.15),
        (0.15, 0.15, 0.075)]
        uniform token subdivisionScheme = "none"
    }
}
```

Code 4.2: *Heptahedron mesh in USD*

4.3.2. Simulation and Synthetic Data Generation

For simulating our real scenario and generating synthetic data from it, an adaptive standalone Isaac Sim script has been written in Python. Initially, with Omniverse Kit, we can create a *SimulationApp* and load all necessary modules to begin our robotics simulation. By providing a config to the *SimulationApp* we are additionally capable of specifying renderer settings. After successfully creating our *SimulationApp*, we begin by loading the trajectory data of our real scenario, into two separate arrays for positions and rotations, along with camera and LiDAR configurations. To create a 3D world in our simulator, we dynamically open the USD stage of our recreated scenario and add the USD file of our virtual mobile robot as a reference.
After successfully loading our scene, the task of replicating our real sensors virtually is next. For simulating and modeling our real sensors within Isaac Sim, several different components offered by Omniverse have been utilized to achieve as much realism as possible. With the help of the USD API, Omniverse Kit, ROS 2 as well as the extensions ROS 2 Bridge and Replicator, we want to model our stereo camera and LiDAR sensor.
Starting with replicating our stereo camera, to initially create a camera in Isaac Sim we use the USD Geometry Schema. We simply built a primitive in our simulation stage and set, with the appropriate USD commands, the parameters of the ZED 2 stereo camera, which were provided dynamically as a configuration file. We repeat this process for a second time and position the left and right camera in our ZED 2 primitive which is placed in the local coordinate system of our virtual robot. The coordinate conventions of Omniverse Isaac Sim tell us, that the world axes use the +Z axis as the up and the +X axis as the forward direction. On the other hand, the default camera axes are different from the world axes. Here the up direction is the +Y axis and the forward direction is the -Z axis. In other words, cameras that we create look down the -Z axis with the +Y axis upwards. To make the cameras look at our cube structure, we need to convert the camera axes to the world axes. Given the camera transform C, of our ZED 2 primitive, which is the parent prim of both cameras, we rotate the camera by $-90°$ around the Z and Y axis as

illustrated in Equation 4.4.

$$C \leftarrow C \cdot \begin{bmatrix} R(-90°, z) & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \cdot \begin{bmatrix} R(-90°, y) & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{4.4}$$

Now that we successfully created, placed, and adjusted our virtual stereo camera, we need to take additional steps to prepare our camera for image and camera info data collection. For this purpose, the synthetic data generation tool Replicator and the ROS 2 Bridge extension will be utilized. First, we need to enable the ROS 2 bridge extension and then create for each camera a render product and assign the resolution we want for the out-coming images. Doing so enables us to take advantage of custom writers Replicator provides, to process the image data stream on every physics step. In our scenario, the *ROS2PublishCameraInfo* writer is employed, leveraging the identical named ROS 2 node to publish the intrinsic camera matrix and several other parameters. Furthermore, the *LdrColorSDROS2PublishImage* writer is applied to publish the raw RGB image data of each camera. By publishing these data topics we are now able to record all synthetically created data with ROS 2, as we did in our real scenario.
If one wants to create a virtual LiDAR sensor, Isaac Sim offers two methods built on different architectures. By using the PhysX based LiDAR sensor, our simulated LiDAR utilizes ray casting and the PhysX Collision API to determine how far an emitted ray traveled before colliding with an object in the scene [51]. For this, it is important to create a Physics Scene in our stage beforehand, because by creating it we provide the necessary physics properties that the PhysX based LiDAR sensor needs. For objects to be detected by the LiDAR sensor, one needs to add a Physics collider to it. With the USD API and Omniverse Kit, we can create such a PhysX based LiDAR sensor with all its properties, e.g. make it stationary or rotary, and also attach a Physics collider to objects that we want to detect. We can then collect data such as depth, zenith, azimuth, and point cloud data at each physics step. One big advantage of PhysX based LiDAR sensors is, that we can attach semantics to our objects, i.e. labeling them. The LiDAR sensor will return semantic data, which is ideal for training models that deal with problems such as segmentation and object recognition. However, we will not make use of this functionality, as it is not necessary for our evaluation in this thesis. Unfortunately, the PhysX based LiDAR comes with a significant disadvantage considering realism. Since the PhysX LiDAR sensor applies ray casting and Physics Collisions, it doesn't take into account what type of material the object consists of when hit with a ray. Therefore, objects that are completely transparent or highly reflective appear to be fully opaque when observing the resulting point cloud data from the LiDAR sensor. As we reviewed in Section 3.2, this appears to be a common ToF replication and leads to realism problems for simulated LiDAR sensors.
As a response to that issue, NVIDIA proposed an alternative to the PhysX based LiDAR sensor. On the 17th of December 2022, NVIDIA released Isaac Sim 2022.2.0, where the RTX LiDAR sensor has been added to the repository of sensors Isaac Sim offers. The RTX LiDAR sensors utilize the RTX ray

tracing technology in render time on the GPU with RTX hardware [52]. The underlying crucial difference to the former introduced LiDAR variant is that now the type of material is considered. By creating a customized LiDAR config file we are able to simulate the exact parameters of our real sensor. Apart from specifying the parameters of our LiDAR sensor, we can also make our simulated LiDAR noisy. For example, we can employ a Gaussian beam as a ray type instead of an idealized one or add mean and standard deviation (SD) errors to the azimuth and elevation of our sensor. Sensor materials are another important factor for the accomplishment of realism in simulated LiDAR sensors. By assigning predefined sensor material types to material names on a USD stage, we can make rays interact differently with materials when they hit the surface of an object. Contrary to its benefits, material properties such as IoR, specularity, and other material properties do not affect the sensor material. In other words, if we assign the sensor material *GlassStandard* on two different glass materials with different properties, they will behave the same when a beam hits the surface of the glass.
From this observation, we can already assume, that the latter introduced sensor variant is a greater attempt at achieving realism. We utilize both sensor variants, to determine how well both LiDAR sensor implementations perform. For the RTX LiDAR, a custom LiDAR config file has been created with all the mentioned specifications listed in Table 4.2. Moreover, we assigned the following sensor types to their respective materials: *OakTreeBark*, *FabricStandard*, *MetalSilver* and *GlassStandard*. When creating our RTX LiDAR with Omniverse Kit, the sensor behaves like a camera and thus we need to rotate it like we did in Equation 4.4. To set up our PhysX LiDAR sensor, we only need to provide our LiDAR parameters during creation and add a collider to our virtual cube tower, since we are only interested in points from our cube construction when acquiring the synthesized data. By setting up all necessary configurations for our RTX and PhysX LiDAR sensor, we are now ready to place it in the local coordinate system of our virtual robot. Finally, we make use of Replicator and ROS 2 Bridge again, to create a render product for our RTX sensor, since it is a camera. We attach this render product to the custom writer *RtxLidarROS2PublishPointCloud* which publishes, as the name already suggests, point cloud data at each physics step in our simulation. To acquire the point cloud data from our PhysX LiDAR sensor, we apply a LiDAR sensor interface and save point clouds at each physics step.
Lastly, we need to register a physics callback function, that will be executed at each physics step inside our simulation. We require this step function, to move and orient our robot with all its sensors to the according position and rotation. At each time step in the simulation, we provide the next recorded pose of the real robot, to a *set_pose* function, that will move and rotate our virtual robot primitive in our 3D world. For this, we utilize the USD API, to add a transform operation. By applying the Graphics Foundation module of the USD API, we can create a homogenous 4x4 transformation matrix T for our robot prim. We set the translation t and rotation R of our transform T with our provided 3D position and quaternion vectors. Doing so will now

translate and orient our virtual robot at each time step, exactly as our real robot was for each recorded timestamp. Additionally, we publish raw RGB images, camera info, and point cloud data via Replicator's writers that we attached to our sensors render products.

4.4. Post-processing

While acquiring data from our real and virtual environment, a handful of unnecessary information was collected. Moreover, in the context of our synthetic data, no post-processing methods have been employed yet and thus will be discussed here. We apply these processing methods such that we create more realism in our synthesized data and make the data evaluation fair.
The collected real and virtual RTX LiDAR point cloud data recorded the surrounding environment during all tests and thus saved an exceedingly amount of points, which are not used for our evaluation. As we can see, Figure 4.10 demonstrates this problem and we observe that the sensor detects humans and other objects in its range. As already mentioned, the majority of those points are not of interest to us, since we only focus on creating accurate synthetic data of our cube tower. Therefore, to alleviate this issue, we can apply an initial mask to our real and synthetic point cloud data, that cuts off all points outside a specific area around the cube tower. In Figure 4.11, we visualize what a point cloud sample looks like after this process. Following the initial masking of our point cloud, a statistical outlier removal approach was chosen to eliminate points that were not cut off during the initial cleansing. 20 neighboring points have been taken into account to calculate the average distance with an SD of 3. Figure 4.12 visualizes in red the removed outliers of our sample point cloud. As a last processing step, the DBSCAN (Density-based spatial clustering of applications with noise) cluster algorithm was utilized to detect groups of points and then eliminate the clusters that were furthest away from the point of origin. This processing step was necessary since some surface artifacts with many neighboring points were not outsourced during initial masking and outlier elimination. We applied a maximal distance ϵ of 0.2 with a minimum of 8 samples, for a point to be considered the core point of the neighborhood. The final resulting point cloud can be seen from a sample in Figure 4.13. As advised by researchers Gusmão G.F., Barbosa C.R.H., and Raposo A.B., after cleansing our point cloud data, we apply a Gaussian noise filter $N(0, 1 \cdot 10^{-3})$ to our synthetic point cloud data. The Euclidean distance has been selected for all distance calculations in the above-mentioned processing steps.
In terms of the recorded image data for our real images, not much post-processing has been done, since we are interested in evaluating the raw data here. However, for our synthetic images, we want to achieve more realism by applying post-processing methods. For real and synthetic images no unnecessary information has to be cut out, like we did for point clouds, considering we want to determine the accuracy of the full image. On account of the fact that virtual cameras are not affected by noise like real cameras are, we first want to

apply a Gaussian noise filter to our image render pipeline. This is best done by utilizing Replicator, to augment an existing Annotator that we attached to our camera render products. We create a simple Gaussian noise kernel function, register it as a new Annotator, and compose it with the existing RGB Annotator, as can be seen in Code 4.3. Doing so enables us to directly manipulate the render pipeline instead of applying a noise kernel after collecting image data. Additionally, we can introduce many post-processing methods that Isaac Sim offers out of the box, such as chromatic aberration and motion blur. Considering these post-processing methods require additional resources and time to render, most of the parameters were left on default since these are more than satisfying. Additionally, many of the post-processing methods, that could have been introduced to our image synthesis pipeline, would add effects to our synthetic images which are not found in our real images.

```
import numpy as np
import omni.kit
import omni.replicator.core as rep

# ... create camera and render product ...

def gaussian_noise(data_in: np.ndarray, kernel_seed,
                   mu: float = 0.0, sigma: float = 25.0):
                np.random.seed(kernel_seed)
                gn = np.random.normal(mu, sigma, data_in.shape)
                return np.clip(data_in + gn, 0, 255)

gauss_annot = rep.annotators.Augmentation.
                from_function(gaussian_noise_np, kernel_seed=123)

rgb_annot = rep.AnnotatorRegistry.get_annotator("rgb")
rgb_annot.augment(gauss_annot)

rgb_annot.attach([render_product])
```

Code 4.3: *Augment existing Annotator with custom noise function*

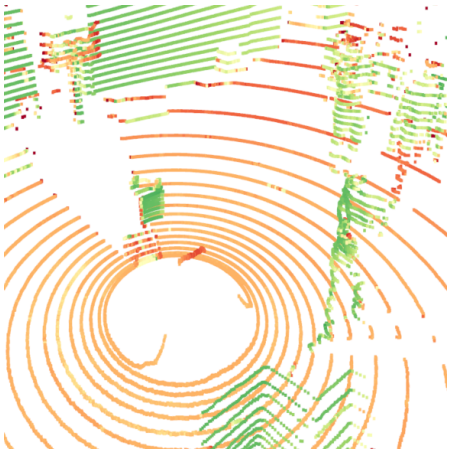


Figure 4.10.: *Point cloud sample before processing*

Figure 4.11.: *Point cloud sample after initial masking*

Figure 4.12.: *Removal of outliers with statistical approach*

Figure 4.13.: *Final result of point cloud sample after cluster removal*

5. Results

To evaluate Isaac Sim's capabilities in creating accurate synthetic data, a series of simulations with several parameters have been conducted to acquire synthesized images and point clouds. Out of 6 synthetically generated data sets, 4 consist of point cloud data, while the remaining two are image data sets. Since we are interested in the quality of our synthetically generated images and point clouds, a handful of image quality and point cloud similarity measures have been selected. Section 5.1 will discuss in more detail the employed measurement metrics. Following this section of this chapter, Section 5.2 and Section 5.3 will present and discuss the results of our image quality and point cloud similarity assessment.

5.1. Metrics

To judge the performance and accuracy of Isaac Sim, we ran all simulations on a multi-GPU-equipped machine. The exact specifications are: Four NVIDIA RTX A4000 GPUs with 46GB GDDR6 VRAM, AMD EPYC 7443 24-Core Processor, and 64GB DDR4 RAM with Linux as the operating system. However, during all test only one RTX A4000 was utilized and Isaac Sim only exhausted 1.2-2.2Gb of VRAM during all simulations. Isaac Sim has been executed on a docker container in headless mode for all conducted experiments. For rendering, the Ray Tracing algorithm, provided by NVIDIA, has been utilized instead of Path Tracing, since the latter variant took too much time in generating synthetic images in our scenario. DLSS, NVIDIA's anti-aliasing technology, was employed and is also the standard anti-aliasing method when using Isacc Sim. Since virtual LiDAR sensors are not affected by render settings, no further adjustments had to be done during point cloud collection.
For our similarity benchmark, a collection of spatial and frequency domain image quality and point cloud similarity measures have been selected. Table 5.1 demonstrates an outline for all metrics and their respective value range. Much of the demonstrated literature, in the context of data synthesis methods, in this work, assesses the quality of the synthetic data by utilizing a ML model. This is done, by comparing the accuracy of a model trained with a well-studied or self-obtained real data set, in contrast to a model trained with synthetic data [23, 28, 33, 34, 35]. For this work, the choice was made to use numerical measurement benchmarks, as utilizing ML models would require more preparation and divert the focus away from this work's primary objective. To calculate all listed metrics *image-similarity-measures* [53] and *point-cloud-utils* [54] were utilized, to accomplish this task.
For our evaluation, we split the image and point cloud data sets into two cat-

egories, depending if a trajectory with interpolated poses has been used to acquire the synthesized data. Furthermore, in the context of our point cloud data, we differentiate here more, depending if the PhysX or RTX LiDAR was employed.

Image metrics		
Metric	Description	Value Range
RMSE	root-mean-square error	$[0, \infty)$
PSNR	peak signal-to-noise ratio	$[0, \infty)$
SRE	signal-to-reconstruction error ratio	$[0, \infty)$
SSIM	structural similarity index	$[0, 1]$
Point cloud metrics		
RMSE	root-mean-square error	$[0, \infty)$
HD	Hausdorff distance	$[0, \infty)$
CD	Chamfer distance	$[0, \infty)$

Table 5.1.: *Applied image and point cloud metrics*

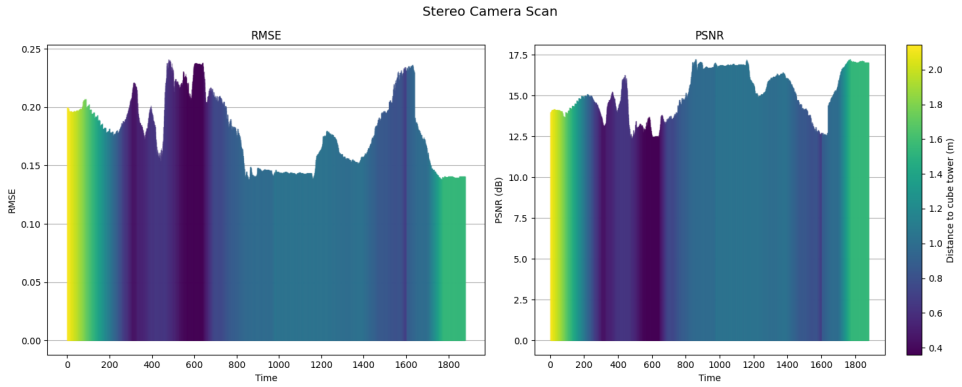
5.2. Image Quality Measurements

After an initial collection of 3332 RGB images from our real scan, only 2237 images were selected for a fair evaluation. Of these selected raw images approximately 4% had no matching pose information. Observing Table 5.2, we can examine that, there are no differences in values between synthesized images from interpolated and non-interpolated poses. For our pixel-by-pixel comparison measurements, namely RMSE and PSNR, we receive relatively moderate average values of 0.18 and 15.04dB, with an SD of 0.03 and 1.46. Illustrated in Figure 5.1 (a) we can see the sometimes rapid deviations of the collected measurements. We also observed that measurements, where the robot had a medium distance of 1.2-1.8m from the cube tower, resulted in improving results. Whereas, worse values emerged from relatively close distances to the cube tower. However, since RMSE and PSNR are hard to interpret, we are not fully able to deduce, how well we were able to reconstruct our real images from synthesizing with Isaac Sim.
For SRE and our perception-based metric SSIM, we receive more information about the true quality of our synthetic images. First of all, since SRE is better suited for images where the brightness can vary, we receive higher and more consistent values than we do for our PSNR evaluation, as can be seen in Figure 5.1 (b). Additionally, SSIM also remains during all distances fairly stable, with peaks at 0.4m gaps to the cube tower. Figure 5.3 and Figure 5.2 illustrate the real and synthetic images for which we received the highest SSIM and SRE along with RMSE and PSNR values. Here we can examine more closely the differences between the real and reconstructed setup. The virtual environment

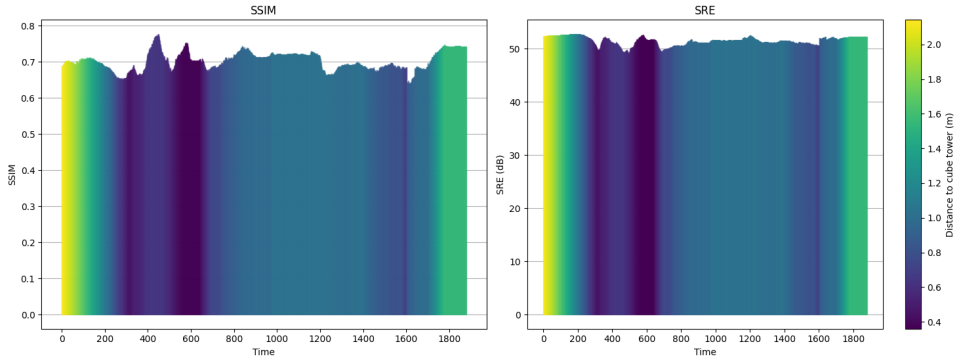
inside Isaac Sim seems to be less illuminated by other light sources, making it appear darker. Moreover, we can see that the front side of the mirroring cube reflects bright objects in the background, which were not modeled in Isaac Sim. As already mentioned in Chapter 4, the issue of the front being fully open and the upper back wall also having no cover to eliminate light reflections, is proposed to be the drawback of our reconstruction. Additionally, during our real scans with our stereo camera, the ZED 2 was not perfectly horizontally aligned and the movement of the robot created some uncertainty in the cameras true orientation. While modeling the virtual stereo camera it was assumed that the camera is perfectly horizontally aligned, which caused for some synthetic images to have slightly different orientations, in contrast to their real counterpart.
By extracting contours from the difference of our real and synthetic image, we are able to receive more insight into fall-backs of our virtual environment. Figure 5.4 demonstrates this process on the previously examined image pair sample. Highlighted on the left top and bottom images with green contours, we observe that regions where no exact replication of our real setup was done, like reflecting duct tape, and slight shifts in environment or robot positions, lead to the biggest influence in image dissimilarity. In the top right gray-scale image, the divergence of positioning is illustrated best.
Note that, the geometric reconstruction in a simulator plays a crucial role in achieving high similarity measures for images, as we saw in our image pair sample. Considering, that only an approximation of our real setup was virtually modeled, the outcome of these results can be viewed as a success for a good approximation. Therefore, With an average SSIM value of 0.71 and an SD of 0.02, we can reason that we achieved a good replication of our real RGB images.

	Average Value		Best Value	
Path / Metric	P	$\hat{P}$	P	$\hat{P}$
RMSE	0.18	0.18	0.13	0.13
PSNR	15.04	15.04	17.23	17.23
SRE	51.52	51.52	52.80	52.80
SSIM	0.71	0.71	0.78	0.78

Table 5.2.: *Average metric values for images collected with and without interpolated positions. No difference between both data sets can be observed*



(a)



(b)

Figure 5.1.: *Evolution of metrics during image synthesis. Color encodes the distance the robot had to the cube tower. (a) Charts RMSE and PSNR values during image synthesis. Measurements are slightly volatile. (b) Charts for SSIM and SRE values during image synthesis. Measurements are much more stable then in (a).*

Figure 5.2.: *On the left we can find the recorded image of the ZED 2 camera and on the right the synthesized image from Isaac Sim. RMSE of 0.17 and PSNR of 17.22 at a distance of 0.96m to the cube tower*

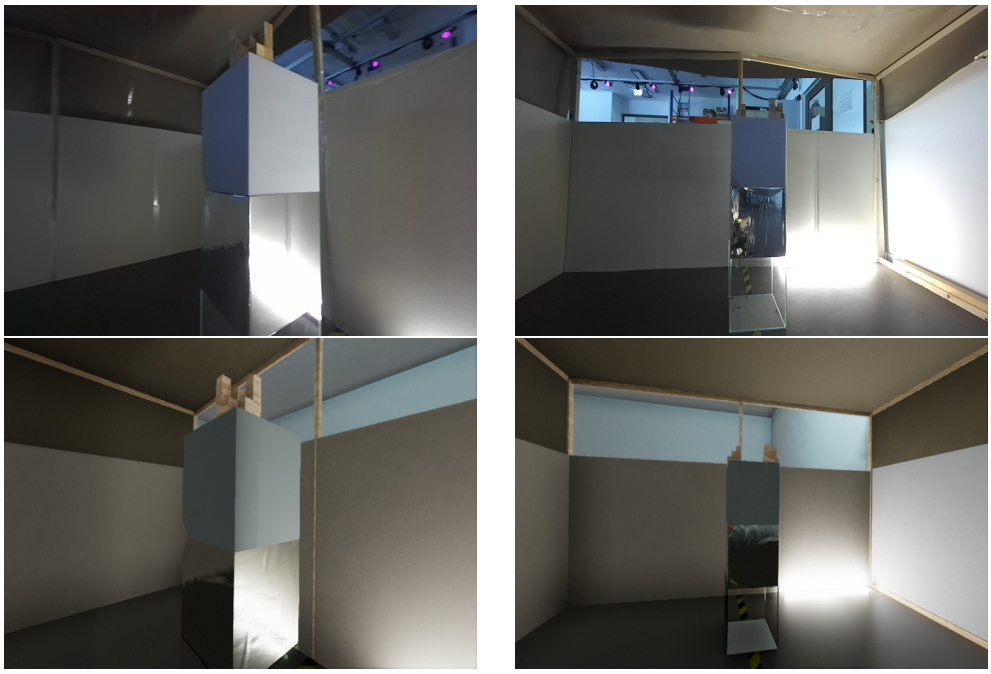


Figure 5.3.: *Top are the real recorded images and on the bottom the respective synthesized images, generated at the same timestamp, from Isaac Sim. SSIM of 0.78 and SRE of 51dB with a distance of 0.6m on the left. SSIM of 0.69 and SRE of 52.8dB with a distance of 1.1m on the right*

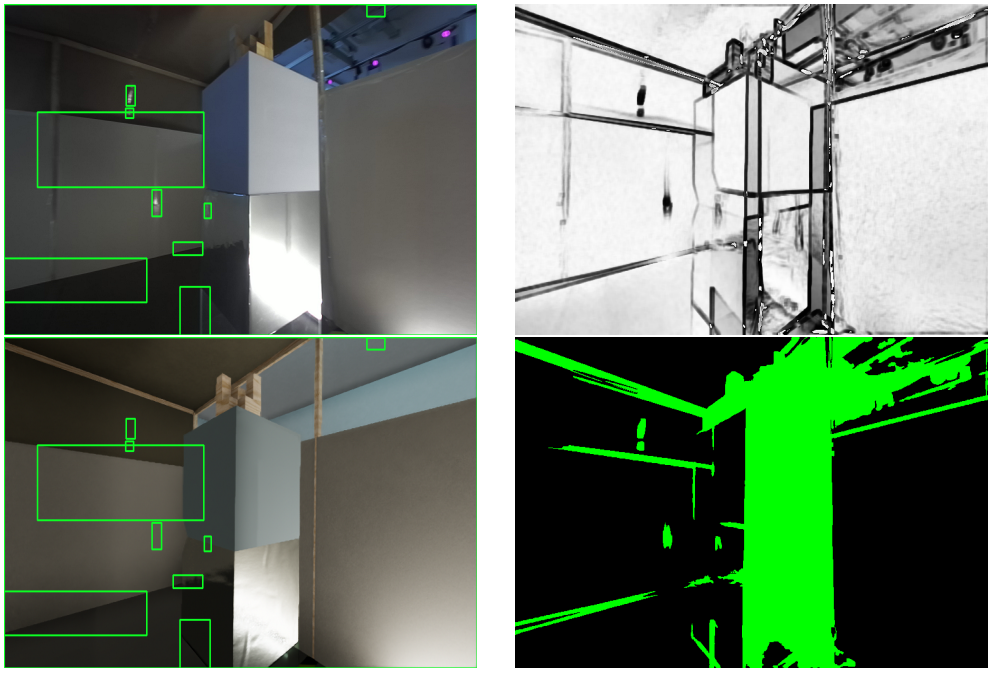


Figure 5.4.: *On the left are contours, highlighting the differences between the top real and bottom synthetic image. Top right illustrates in gray-scale disparity. Darker regions mean greater disparity. Bottom right shows thresholded regions with biggest dissimilarity.*

5.3. Point Cloud Similarity Measurements

During our real scan, we collected a total of 1161 point clouds, from which we selected 748 to evaluate the similarity between the real and synthetically generated data. The remaining 413 point clouds, which were cut off from this assessment, were either at the start of the official recording or were outside of the timestamp range of the recorded pose data. Table 5.3 illustrates the average and best values for PhysX and RTX LiDAR sensors with and without interpolated trajectories. Since no path timestamps matched the timestamps of the recorded point cloud data, the whole trajectory had to be interpolated. In contrast to our evaluation of the synthetic image data, our synthetic point clouds showed more similarity to the real scans when interpolating poses, instead of assigning the pose of the nearest timestamp.

	Average Value				Best Value			
Path / Metric	P_{RTX}	$\hat{P}_{RTX}$	P_{PhysX}	$\hat{P}_{PhysX}$	P_{RTX}	$\hat{P}_{RTX}$	P_{PhysX}	$\hat{P}_{PhysX}$
RMSE	0.051	0.054	**0.046**	0.047	0.023	0.023	0.018	0.019
CD	0.12	0.13	**0.1**	**0.1**	0.052	0.051	0.045	0.046
HD	**0.27**	0.28	0.28	0.28	0.1	0.11	0.084	0.084

Table 5.3.: *Average metric values for RTX and PhysX point clouds collected with and without interpolated positions. Trajectories without interpolated poses show in general weaker similarities to the real point cloud data. Point clouds collected from the RTX LiDAR have the best HD distance, while PhysX yields better RMSE and CD distance*

Furthermore, we can observe from our measurement data that the PhysX LiDAR sensor holds the best RMSE and Chamfer distance results with an average dissimilarity of 0.046m and 0.1m. One might assume, that from these values, the PyhsX LiDAR sensor is a greater attempt at simulating a real LiDAR sensor. However, RMSE may not capture outliers or localized differences, and Chamfer, on the other hand, is very sensitive to outliers and also may overlook dense groups of points [55]. Hausdorff can also be influenced by extreme values. Figure 5.5 illustrates the problem of the evaluated metrics for some point cloud samples. Here we can see how the RMSE, Chamfer, and Hausdorff distances are equal, even though the RTX point cloud sample is a greater replication of the real scan than the PhysX point cloud sample. Additionally, we examine a mutual drawback in replicating LiDAR behavior with reflecting surfaces. Both sensors detected the middle mirroring cube, while the real LiDAR sensor did not receive any rays from hitting the mirror surface. Despite that, the RTX LiDAR sensor successfully replicated the raydrop behavior with our bottom glass cube, while the PhysX sensor detected it as fully opaque. The average amount of data points collected from our real scan is 291 points, while our RTX sensor collected an average of 350 and the PhysX sensor 1036 points. From the illustrated point cloud sample in Figure 5.5, we

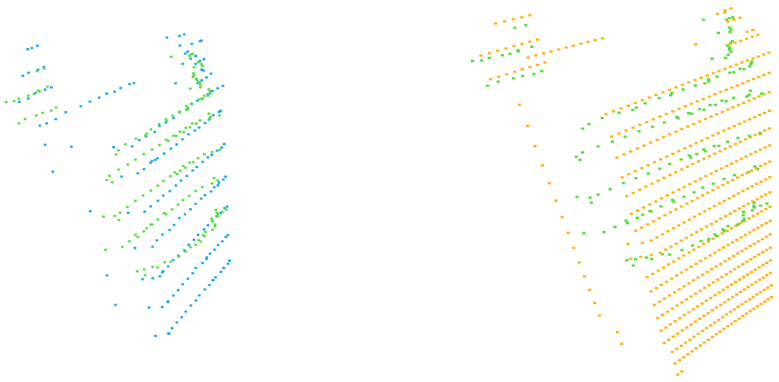


Figure 5.5.: *Left and right, we have the actual point cloud depicted in green. The blue colored point cloud on the left originates from the RTX LiDAR, the one on the right in orange is from the PhysX LiDAR. Both RTX and PhysX share an RMSE of 0.02, a CD of 0.11, and an HD of 0.37*

directly observe that the RTX LiDAR sensor almost collects as many points as the real scan did, while the PhysX sensor collected an exceedingly large amount of points. The scan lines of the PhysX LiDAR sensor also appear to be much more frequent than those of its real counterpart. Thus, we can see that the PhysX variant detects too much detail as illustrated in Figure 5.6.
Illustrated in Figure 5.7 (a) we can observe how our similarity measurements changed during simulation with various distances. From the Chamfer and Hausdorff chart, we can interpret the increase in dissimilarity at far ranges for distances to the cube tower at > 1.6m, to be the issue of detecting the glass cube. The lowest values can be found for ranges under 1.6m, where the mirror cube is in the dead zone of the PhysX sensor. In the rightmost chart for the Hausdorff measurements, this phenomenon is seen the most. Going from the PhysX implementation to the RTX LiDAR variant, we can see that the RMSE chart behaves almost identically. In fact, the SD of both RMSE measurements is 0.02. The Chamfer chart of the RTX LiDAR sensor, however, examines lower dissimilarity in far regions in contrast to the PhysX sensor. For the Hausdorff measurements, we can also see that results are much more robust and not as rapidly changing as for the PhysX variant. The RTX sensor yields an SD of 0.02 for the calculated Hausdorff distances, while the PhysX sensor holds an SD of 0.09.
We can conclude from this assessment that the RTX LiDAR sensor produced more accurate point cloud data than the PhysX variant. Therefore are the dissimilarity measurements more to be trusted from the RTX sensor. Since no exact positions and orientations were collected for the recording of the real point cloud data, the average distances can be considered low and consequently a good approximation of a real LiDAR sensor. Unfortunately, no evaluation could be made with the different ray types that Isaac Sim offers for its RTX LiDAR sensor. Since real light rays emitted from LiDAR are not ideal rays, and instead lose intensity when further away from the center, the Gaussian and uniform beam types could improve realism even more.

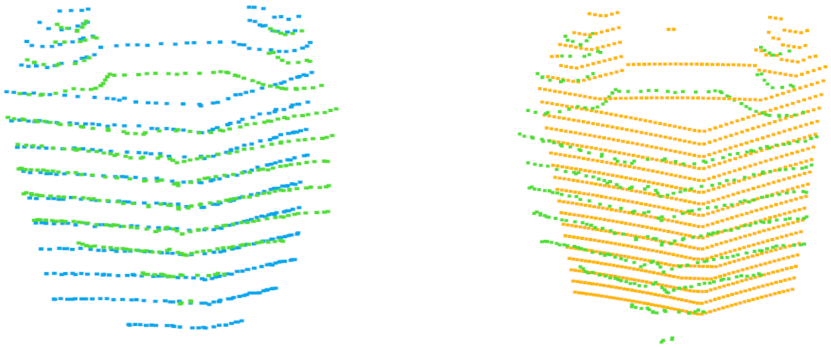


Figure 5.6.: *On either side, the genuine point cloud is illustrated in green. The left showcases the blue point cloud from the RTX LiDAR, while the right features an orange point cloud from the PhysX LiDAR sensor. In comparison to both the RTX and the actual point cloud, the PhysX point cloud exhibits too much detail*

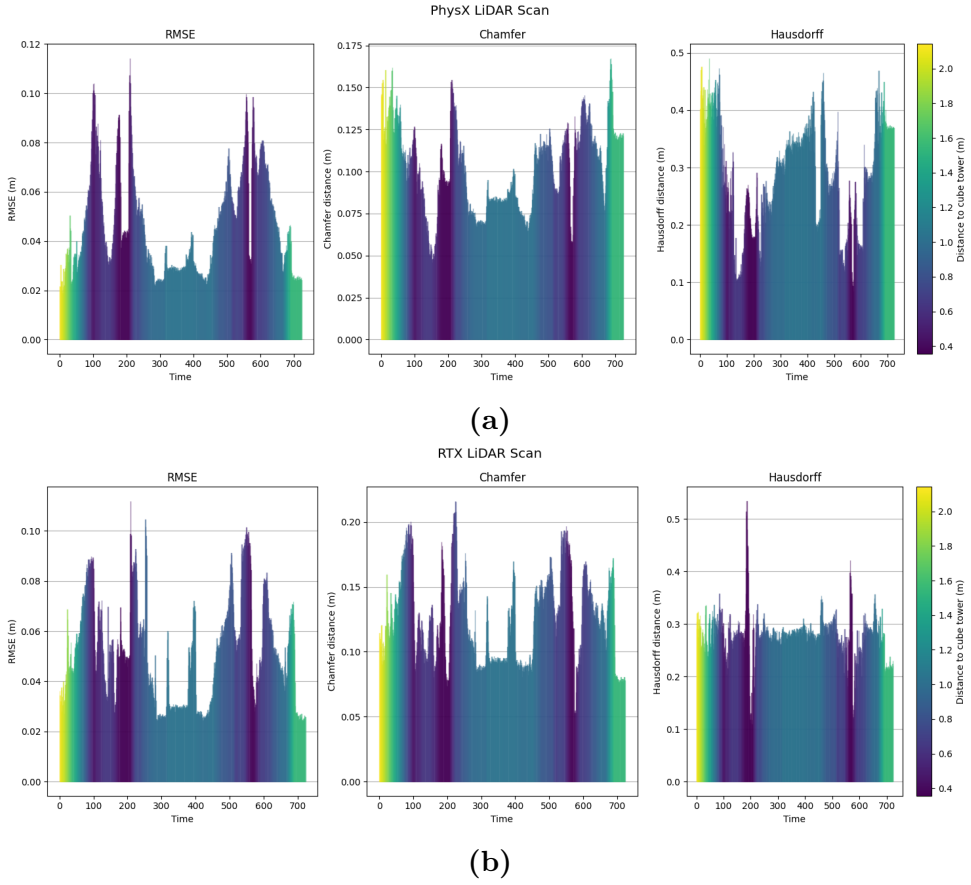


Figure 5.7.: *Metrics of point cloud synthesis during simulation. Color encodes distance to the cube tower. (a) Charts of the PhysX LiDAR scan for RMSE, CD and HD values. Measurements deviating drastically. (b) Charts of the RTX LiDAR scan for RMSE, CD and HD values. RMSE and CD are similar to PhysX, while HD seems to be much stabler.*

6. Conclusion

The main motivation of this thesis was to evaluate NVIDIA's Omniverse Isaac Sim capabilities as a simulator and synthetic data generation tool. Thus, an in-depth analysis of the need for synthetic data and the approach chosen to asses Isaac Sim was demonstrated.
Chapter 1 started by, introducing the importance and need of high quality and large data sets, in the context of AVs. With current real-world applications, we saw that time and other resources are suffering under traditional data collection procedures. Synthetic data and its generation methods have thus, proposed to be the best solution to alleviate this issue.
Following the introduction of this thesis, Chapter 2 provided important fundamentals, necessary for a better understanding of the used technologies. This included the popular robotics middleware ROS 2, the ToF sensor LiDAR, and Isaac Sim.
With Chapter 3, the ongoing state of research considering synthetic image and point cloud data, along with data synthesis methods, was shown. For both data categories, a collection of well-studied data sets was demonstrated, in the company with their current pitfalls when regarding modeling realism. Closing this chapter of this work, was a discussion about the domain gap problem between real and synthetic data.
In Chapter 4, design and research goals for analyzing Isaac Sim were demonstrated. All employed technologies and the chosen scanning environment were discussed. Following the data collection of our physical setup, an in-depth presentation of the system architecture, built with Isaac Sim, has been proposed to realistically and accurately simulate our real scan. Here, many of the introduced preliminaries of Isaac Sim were applied, to geometrically model our environment and construct our virtual stereo camera and LiDAR sensor.
Finally, in Chapter 5 of this thesis, the results of our quality and similarity measurements between real and synthetic data have been presented. In this instance, we observed that interpolated trajectories had no significant influence on the quality of our generated synthetic images. Yet, point cloud similarity decreased when no interpolation of poses was used. Given that only 4% of image timestamps and all timestamps of the recorded point cloud data had no match, we can reason that the point cloud measurements had an increase of similarity to the real data since we approximated missing poses. Whereas, the majority of poses where the stereo camera captured an image, were given and therefore path interpolation had no influence. Furthermore, we have seen good approximation values of our synthetic images to their real counterparts, considering that no exact replication of the real scene was built. As was pointed out in the methodology of this work, it was suspected that

the influence of the accurate 3D reconstruction in Isaac Sim, will have a big influence on the resulting image similarity measurements. Utilizing more sophisticated rendering techniques provided by NVIDIA, like path tracing, and an accurate replication of the real scanning environment would very likely boost the quality of our synthetic images. For our virtual LiDAR variants, we deduced that the RTX LiDAR sensor was a greater attempt at modeling our real employed LiDAR sensor, instead of the PhysX one. We have seen that the PhysX LiDAR implements current pitfalls of common ToF implementations that we investigated in Section 3.2, where raydrop and noise were not modeled realistically. However, in the case of the RTX LiDAR sensor, we received a very well approximation of our real point cloud data. In contrast to the former variant, the RTX LiDAR sensor simulated accurate raydrop behavior for our glass material scenario. Moreover, with the inclusion of error parameters to the RTX LiDAR sensor, noise was also replicated in a realistic way and could have been even more improved if Gaussian or uniform beams would have been employed. Within this work, we were unfortunately unable to reproduce the behavior of our actual LiDAR in the case of the mirroring cube using either the PhysX or RTX LiDAR sensors.

6.1. Outlook

Throughout this thesis, several problems and strategies to improve the analysis were encountered. Considering the real scanning environment, limitations existed on how well a controlled surrounding, in which we know all significant parameters, could be constructed. An optimal scanning environment in this case would be, a barely illuminated and accurately measured enclosure without any structures inside it. In our test, we could see that complex geometry outside our structure still influenced the illumination of the scene. Few or multiple controlled known light sources, casting light on a simple scanning structure (like our cube tower), would simplify reconstructing the real environment as well. Another beneficial implementation would be the inclusion of a well-known start and end position of the robot that scanned the environment. Moving forward, a handful of standard robotics simulators were presented, but no attempt to recreate this experiment with industry-standard tools like Gazebo or Unity has been made. Comparing generated synthetic image and point cloud data from Isaac Sim against other simulators would be an interesting evaluation, to determine how well Isaac Sim operates against such tools for which a lot of literature already exists.
Unfortunately for this work, no evaluation of Isaac Sim's depth sensing capabilities could have been made, since an error during data collection prevented us from recording depth of our stereo camera. Additionally, the employed ROS 2 wrapper for the ZED SDK, currently does not support input from virtual cameras, and thus would not enable us to asses the resulting depth images fairly anyways. Despite this, NVIDIA offers their own DNN stereo disparity model, that can be supplied with ROS image and timestamp topics, to calculate stereo disparity [56]. This would make another interesting research topic,

to compare, if accurate depth images could be achieved with NVIDIA's stereo disparity DNN model.
As already mentioned at the beginning of Chapter 5 in Section 5.1, many researchers assess the accuracy of their synthesized data, by training ML models with real and generated data sets. For future work, it would be beneficial to examine the synthetically generated images and point clouds, for problems such as object recognition, segmentation, or 3D pose estimation, against models trained with our real data.
Looking ahead, more research in decreasing the domain gap between synthetic and real data could be made, by applying hybrid solutions of simulators and GANs along with domain randomization.

Bibliography

[1] Kyle Wiggers. *Waymo's autonomous cars have driven 20 million miles on public roads*. VentureBeat, Jan. 2020. URL: https://venturebeat.com/ai/waymos-autonomous-cars-have-driven-20-million-miles-on-public-roads/ (visited on 06/25/2023).

[2] Shahryar SOROOSHIAN and Shrikant PANIGRAHI. "Impacts of the 4th Industrial Revolution on Industries". In: *Walailak Journal of Science and Technology (WJST)* 17.8 (Aug. 2020), pp. 903–915. DOI: 10.48048/wjst.2020.7287. URL: https://wjst.wu.ac.th/index.php/wjst/article/view/7287.

[3] Darrell Etherington. *Waymo has now driven 10 billion autonomous miles in simulation*. TechCrunch, July 2019. URL: https://techcrunch.com/2019/07/10/waymo-has-now-driven-10-billion-autonomous-miles-in-simulation/?guccounter=1 (visited on 06/25/2023).

[4] Alvaro Figueira and Bruno Vaz. "Survey on Synthetic Data Generation, Evaluation Methods and GANs". In: *Mathematics* 10.15 (2022). ISSN: 2227-7390. DOI: 10.3390/math10152733. URL: https://www.mdpi.com/2227-7390/10/15/2733.

[5] Stefan Mihai, Mahnoor Yaqoob, Dang V. Hung, William Davis, et al. "Digital Twins: A Survey on Enabling Technologies, Challenges, Trends and Future Prospects". In: *IEEE Communications Surveys and Tutorials* 24.4 (2022), pp. 2255–2291. DOI: 10.1109/COMST.2022.3208773.

[6] Steve Borkman, Adam Crespi, Saurav Dhakad, Sujoy Ganguly, et al. *Unity Perception: Generate Synthetic Data for Computer Vision*. 2021. arXiv: 2107.04259 [cs.CV].

[7] *NVIDIA Omniverse*. https://www.nvidia.com/en-us/omniverse/. Accessed: 14 June 2023.

[8] Danny Shapiro. *Mercedes-Benz to Build Factories With Omniverse*. NVIDIA Blog, Jan. 2023. URL: https://blogs.nvidia.com/blog/2023/01/03/mercedes-benz-next-gen-factories-omniverse/.

[9] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022), eabm6074.

[10] Misha Urooj Khan, Syed Azhar Ali Zaidi, Arslan Ishtiaq, Syeda Ume Rubab Bukhari, Sana Samer, and Ayesha Farman. "A comparative survey of lidar-slam and lidar based sensor technologies". In: *2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC)*. IEEE. 2021, pp. 1–8.

[11] Velodyne Lidar. *Velodyne's Guide to Lidar Wavelengths*. Velodyne Lidar, Nov. 2018. URL: https://velodynelidar.com/blog/guide-to-lidar-wavelengths/ (visited on 09/25/2023).

[12] Jamie Carter, Keil Schmid, Kirk Waters, Lindy Betzhold, Brian Hadley, Rebecca Mataosky, and Jennifer Halleran. "Lidar 101: An Introduction to Lidar Technology, Data, and Applications. National Oceanic and Atmospheric Administration (NOAA) Coastal Services Center, Charleston, South Carolina". In: *Charleston, SC* (2012).

[13] Paul Lienert and Jane Lanhee Lee. "Lidar laser-sensing technology: from self-driving cars to dance contests". In: *Reuters* (Jan. 2020). URL: https://www.reuters.com/article/us-tech-ces-lidar/lidar-laser-sensing-technology-from-self-driving-cars-to-dance-contests-idUSKBN1Z62AS (visited on 09/25/2023).

[14] *Time of flight*. Wikipedia, Sept. 2020. URL: https://en.wikipedia.org/wiki/Time_of_flight (visited on 09/27/2023).

[15] Wikipedia Contributors. *Spherical coordinate system*. Wikipedia, Dec. 2019. URL: https://en.wikipedia.org/wiki/Spherical_coordinate_system (visited on 09/27/2023).

[16] *Isaac Sim*. NVIDIA Developer. URL: https://developer.nvidia.com/isaac-sim (visited on 09/24/2023).

[17] *USD Home — Universal Scene Description 23.08 documentation*. openusd.org. URL: https://openusd.org/release/index.html (visited on 09/24/2023).

[18] *Material Definition Language from NVIDIA*. NVIDIA. URL: https://www.nvidia.com/en-us/design-visualization/technologies/material-definition-language/ (visited on 09/24/2023).

[19] *Replicator — Omniverse Extensions latest documentation*. docs.omniverse.nvidia.com. URL: https://docs.omniverse.nvidia.com/extensions/latest/ext_replicator.html (visited on 09/24/2023).

[20] *Annotators Information — Omniverse Extensions latest documentation*. docs.omniverse.nvidia.com. URL: https://docs.omniverse.nvidia.com/extensions/latest/ext_replicator/annotators_details.html (visited on 09/24/2023).

[21] *Introduction — Omniverse Nucleus latest documentation*. docs.omniverse.nvidia.com. URL: https://docs.omniverse.nvidia.com/nucleus/latest/index.html (visited on 09/24/2023).

[22] Sergey I. Nikolenko. *Synthetic Data for Deep Learning*. 2019. arXiv: 1909.11512 [cs.LG].

[23] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. *FlowNet: Learning Optical Flow with Convolutional Networks*. 2015. arXiv: 1504.06852 [cs.CV].

[24] Nikolaus Mayer, Eddy Ilg, Philipp Fischer, Caner Hazirbas, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. "What Makes Good Synthetic Training Data for Learning Disparity and Optical Flow Estimation?" In: *International Journal of Computer Vision* 126.9 (Apr. 2018), pp. 942–960. DOI: 10.1007/s11263-018-1082-6. URL: https://doi.org/10.1007%2Fs11263-018-1082-6.

[25] Yair Movshovitz-Attias, Takeo Kanade, and Yaser Sheikh. *How useful is photo-realistic rendering for visual learning?* 2016. arXiv: 1603.08152 [cs.CV].

[26] Sebastian Hartwig and Timo Ropinski. "Training object detectors on synthetic images containing reflecting materials". In: *arXiv preprint arXiv:1904.00824* (2019).

[27] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.* Vol. 1. 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517.

[28] Erroll Wood, Tadas Baltrušaitis, Charlie Hewitt, Sebastian Dziadzio, Matthew Johnson, Virginia Estellers, Thomas J. Cashman, and Jamie Shotton. *Fake It Till You Make It: Face analysis in the wild using synthetic data alone.* 2021. arXiv: 2109.15102 [cs.CV].

[29] Bin Yang, Wenjie Luo, and Raquel Urtasun. "PIXOR: Real-Time 3D Object Detection From Point Clouds". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* June 2018.

[30] Tahir Rabbani, Frank Van Den Heuvel, and George Vosselmann. "Segmentation of point clouds using smoothness constraint". In: *International archives of photogrammetry, remote sensing and spatial information sciences* 36.5 (2006), pp. 248–253.

[31] David Griffiths and Jan Boehm. "SynthCity: A large scale synthetic point cloud". In: *ArXiv preprint.* 2019.

[32] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. "CARLA: An open urban driving simulator". In: *Conference on robot learning.* PMLR. 2017, pp. 1–16.

[33] Fei Wang, Yan Zhuang, Hong Gu, and Huosheng Hu. "Automatic Generation of Synthetic LiDAR Point Clouds for 3-D Data Analysis". In: *IEEE Transactions on Instrumentation and Measurement* 68.7 (2019), pp. 2671–2673. DOI: 10.1109/TIM.2019.2906416.

[34] Vlas Zyrianov, Xiyue Zhu, and Shenlong Wang. "Learning to generate realistic lidar point clouds". In: *European Conference on Computer Vision.* Springer. 2022, pp. 17–35.

[35] Benoît Guillard, Sai Vemprala, Jayesh K. Gupta, Ondrej Miksik, Vibhav Vineet, Pascal Fua, and Ashish Kapoor. "Learning to Simulate Realistic LiDARs". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 8173–8180. DOI: 10.1109/IROS47612.2022.9981120.

[36] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Sazanovich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. "Lidarsim: Realistic lidar simulation by leveraging the real world". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11167–11176.

[37] Guilherme Ferreira Gusmão, Carlos Roberto Hall Barbosa, and Alberto Barbosa Raposo. "Development and validation of LiDAR sensor simulators based on parallel raycasting". In: *Sensors* 20.24 (2020), p. 7186.

[38] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. "Cut, paste and learn: Surprisingly easy synthesis for instance detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 1301–1310.

[39] N. Koenig and A. Howard. "Design and use paradigms for Gazebo, an open-source multi-robot simulator". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. Vol. 3. 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[40] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2012, pp. 5026–5033.

[41] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. "Airsim: High-fidelity visual and physical simulation for autonomous vehicles". In: *Field and Service Robotics: Results of the 11th International Conference*. Springer. 2018, pp. 621–635.

[42] Jian Zhao, Lin Xiong, Jianshu Li, Junliang Xing, Shuicheng Yan, and Jiashi Feng. "3d-aided dual-agent gans for unconstrained face recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 41.10 (2018), pp. 2380–2394.

[43] Yingzhou Lu, Huazheng Wang, and Wenqi Wei. *Machine Learning for Synthetic Data Generation: A Review*. 2023. arXiv: 2302.04062 [cs.LG].

[44] Karan Bhanot, Miao Qi, John S. Erickson, Isabelle Guyon, and Kristin P. Bennett. "The Problem of Fairness in Synthetic Healthcare Data". In: *Entropy* 23 (Sept. 2021), p. 1165. DOI: 10.3390/e23091165. (Visited on 01/30/2022).

[45] Florian Noichl, Alexander Braun, and AndrÃ© Borrmann. "BIM-to-Scan for Scan-to-BIM: Generating Realistic Synthetic Ground Truth Point Clouds based on Industrial 3D Models". en-GB. In: *Proceedings of the 2021 European Conference on Computing in Construction*. Vol. 2. Computing in Construction. Online Conference: ETH, July 2021, pp. 164–172. ISBN: 978-3-907234-54-9. DOI: 10.35490/EC3.2021.166. URL: https://ec-3.org/publications/conference/paper/?id=EC32021_166.

[46] Igor Kviatkovsky, Nadav Bhonker, and Gerard Medioni. "From real to synthetic and back: Synthesizing training data for multi-person scene understanding". In: *arXiv preprint arXiv:2006.02110* (2020).

[47] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. "Training deep networks with synthetic data: Bridging the reality gap by domain randomization". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 969–977.

[48] *Heptahedron*. Wikipedia, Sept. 2023. URL: https://en.wikipedia.org/wiki/Heptahedron (visited on 09/21/2023).

[49] *Fabric with Wrinkles and Creases*. www.texturecan.com. URL: https://www.texturecan.com/details/209/ (visited on 10/07/2023).

[50] *Omniverse Materials — Omniverse Materials and Rendering latest documentation*. docs.omniverse.nvidia.com. URL: https://docs.omniverse.nvidia.com/materials-and-rendering/latest/materials.html (visited on 09/06/2023).

[51] *PhysX Range Sensors (Lidar, Ultrasonic, Generic Range) — Omniverse IsaacSim latest documentation*. docs.omniverse.nvidia.com. URL: https://docs.omniverse.nvidia.com/isaacsim/latest/ext_omni_isaac_range_sensor.html#isaac-sim-range-sensors (visited on 09/17/2023).

[52] *RTX Lidar Sensor — Omniverse IsaacSim latest documentation*. docs.omniverse.nvidia.com. URL: https://docs.omniverse.nvidia.com/isaacsim/latest/isaac_sim_sensors_rtx_based_lidar.html (visited on 09/17/2023).

[53] Markus U Müller, Nikoo Ekhtiari, Rodrigo M Almeida, and Christoph Rieke. "Super-resolution of multispectral satellite images using convolutional neural networks". In: *arXiv preprint arXiv:2002.00580* (2020).

[54] Francis Williams. *Point Cloud Utils*. https://www.github.com/fwilliams/point-cloud-utils. 2022.

[55] Maxim Tatarchenko, Stephan R Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. "What do single-view 3d reconstruction networks learn?" In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 3405–3414.

[56] Nikolai Smolyanskiy, Alexey Kamenev, and Stan Birchfield. "On the importance of stereo for accurate depth estimation: An efficient semi-supervised deep neural network approach". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 1007–1015.

A. Appendix

A.1. Git Repository

For the purpose of this work, a git project under the URL https://git.cs.uni-kl.de/p_noras19/cubetower-dt has been developed. The repository features all scripts and USD scenes utilized in this work.

A.2. Limits of Reproducibility

Collected real image and point cloud data, are not publicly available. On the other hand, the recorded trajectory and sensor timestamps data are available. Along with the provided assets, the virtual simulation can be replicated and synthetic data generated.
Furthermore, the code to calculate similarity measurements between real and synthetic data is also included in the git repository. Real image and point cloud data, however, are prerequisites for calculating the measurements highlighted in this work.