3DGENie: An Extensible and Procedural Pipeline for Synthetic Point Cloud Generation in Industrial Applications

Anthony Yaghi[1,2], Joe Tekli[3*], Marc Kamradt[1], Raphaël Couturier[2]

[1]TechOffice Munich, BMW Group, Petuelring 130, 80809, Munich, Germany.
[2]FEMTO-ST, University of Franche-Comté, 25030, Besançon, France.
[3]E.C.E. Dept, Lebanese American University, 36, Byblos, Lebanon.

*Corresponding author(s). E-mail(s): joe.tekli@lau.edu.lb;
Contributing authors: anthony.yaghi@bmw.de; marc.kamradt@bmw.de;
raphael.couturier@univ-fcomte.fr;

Abstract

A main R&D pillar in the modern car manufacturing industry revolves around investigating the use of digital assets to train semantic segmentation models, before deploying them in the real-world. Yet, training deep learning models requires huge amounts of data, where there is a clear lack of datasets to facilitate 3D model training, especially for industrial applications. To address this problem, we propose a new synthetic data generation pipeline called 3DGENie designed to generate controlled 3D point clouds. 3DGENie uses state of the art procedural layout generation to produce region layout trees. It then applies 3D scene construction and asset randomization to produce scenes populated with 3D assets. This allows different types of set-ups that are adequate for different kinds of industrial scenarios. Synthetic sensors are subsequently placed in the virtual environment to simulate data capture from the 3D scenes as if monitored by real-world sensors, allowing to study the influence of various environmental variables on model performance. 3DGENie uses Nvidia Omniverse as its scene-building platform and Pixar's Universal Scene Description (USD) for 3D graphics representation to allow for seamless interchange across platforms. While the main application focuses on the generation of car assembly lines, yet, 3DGENie can be used across a range of fields, from autonomous vehicle development, to healthcare virtualization, and augmented and virtual reality applications. Various experiments are conducted to evaluate the quality of the generated 3D point clouds, using several semantic segmentation models. Results highlight the quality and potential of the proposed pipeline.

Keywords: Synthetic Data, 3D Point clouds, Data Generation Pipeline, Procedural Generation, Computer Vision, Semantic Segmentation.

1 Introduction

3D computer vision and perception is becoming critical in various industrial applications, where digital twins and virtual assets show great promise for training computer vision solutions. A main R&D pillar in the modern car manufacturing industry revolves around investigating the usage of digital assets to train 3D computer vision models, before deploying them in the real-world. Yet, there is a clear absence of 3D vision datasets for industrial applications, at least when compared with their 2D counterparts Xiao A.. et al. (2021). However, creating real datasets with the level of scale and complexity required in industry can sometimes be expensive or even impractical, especially when generating 3D point cloud datasets. This requires the usage of industry-scale 3D scanners to acquire accurate 3D mappings, followed by manual labelling, which entails huge financial, logistical, and temporal challenges.

To address these challenges, we propose a novel synthetic data generation pipeline called 3DGENie designed to facilitate the generation of 3D point cloud datasets. It uses procedural generation to produce region layout trees. It then applies 3D scene construction and asset randomization algorithms to produce 3D scenes populated with 3D assets according to user-chosen generation strategies, allowing different types of set-ups (e.g., generating a synthetic assembly line requires layouts and randomizations that are different from generating a supply chain storage post for instance). Synthetic sensor placement allows to simulate data capture from the generated 3D scenes as if it were monitored by real-world cameras and sensors. 3DGENie uses Nvidia Omniverse Nvidia (2024a) as its scene building platform which leverages the latest achievements in GPU technology, and Pixar's Universal Scene Description (USD) Nvidia (2024c) for 3D graphics representation to allow a seamless interchange across multiple industry platforms. We conducted various experiments to evaluate the quality of the generated 3D point clouds, using several deep learning semantic segmentation models. Results highlight the quality and potential of our pipeline. While our main application focuses on the generation of synthetic car assembly lines, nonetheless, our solution can be used across a range of fields, from autonomous vehicle development Song Z.. et al. (2024), to healthcare virtualization Yunas M.. et al. (2023)Chen R.. et al. (2021), as well as augmented and virtual reality applications Morbidoni C.. et al. (2020)Lu Y.. et al. (2023).

The remainder of this paper is organized as follows. Section 2 reviews the literature around 3D datasets and data generation pipelines. Section 3 describes our 3DGENie pipeline. Section 4 describes our experimental evaluation and results, before concluding in Section 5 with ongoing works and directions.

2 Related Work

This section will briefly cover real and synthetic point cloud datasets for machine learning as well as synthetic data generation pipelines and procedural layout generation methods.

2.1 Point Cloud Datasets

Real-World datasets: SensatUrban Hu Q.. et al. (2021) and Semantics3D Hackel T.. et al. (2017) are two real-world datasets in the area of urban and natural scenes. SensatUrban Hu Q.. et al. (2021) leverages photogrammetry to offer point-level segmentation across 13

semantic classes, utilizing a fixed-wing drone to capture aerial images that are then processed into dense colored 3D point clouds, requiring approximately 600 hours of manual annotation. This dataset stands out for its detailed representation of the urban landscapes. Conversely, Semantics3D Hackel T.. et al. (2017) focuses on static scans to produce dense and accurately annotated point clouds in both urban and natural settings. Its unique approach to colorizing points via cube maps from different angles enriches the dataset's visual details. Although it encompasses fewer classes (8 in total) compared with SensatUrban Hu Q.. et al. (2021), nonetheless, it provides high-quality and high-density point clouds. SemanticKIITI Behley J.. et al. (2019) is a renowned dataset based on the odometry of the KITTI Vision Benchmark Geiger A.. et al. (2012), which is derived from a lidar sensor mounted on a car as it travels through various types of roads in Karlsruhe, Germany, including city traffic, residential areas, highways, and countryside roads. The dataset provides accurate annotations for 28 categories in a sequence of 22 scans. Moreover, the authors of the dataset made public the labeling tool that was used to annotate the data. The annotation process was carried out by a team of human annotators who spent a total of 1,700 hours on the task, including quality control. The authors conducted multiple experiments on semantic segmentation and semantic scene completion using state-of-the-art models to assess the quality and practical usage of their dataset.

While the aforementioned datasets provide high-quality 3D point cloud representations, yet their production required huge amounts of time, manual labor, and resources, and their usage remains limited to their real-world assets and scenarios.

CAD model-based datasets: ModelNet Wu Z.. et al. (2014) and ShapeNet Chang A.. et al. (2015) are large labeled collections of 3D CAD models. The models are organized by category and available to download under different formats like Object File Format (OFF), OBJ, or even in voxel representation in the case of ShapeNet Chang A.. et al. (2015). While CAD-based datasets allow design flexibility and extensibility to different application scenarios, nonetheless, they show many limitations. First, the data representation does not resemble the output of a real sensor like a depth camera or Lidar. The reason for it is the fact that the methods used to generate the point clouds from the 3D CAD models do not emulate, by design, how a real sensor operates. Second, the models themselves, being collected randomly from online sources, do not guarantee realistic and high-quality data. The authors mentioned a quality control step during the data collection step to curate only high-quality models. However, this is probably not enough, especially for industry-grade applications, since the quality of the dataset will still be limited by the quality of the publicly available online models. OmniObject3D Wu T.. et al. (2023) presents another approach that revolves around the scanning of daily objects (i.e., objects that we use every day, e.g., table, chair, guitar, etc., in contract with industrial objects) to produce a dataset encompassing textured meshes, point clouds, and multi-view images. A canonical pose is defined for each group of objects, and common manipulations are performed on the objects to increase the authenticity and variety of the dataset. However, OmniObject3D Wu T.. et al. (2023) generates its point clouds from 3D meshes rather than direct capture from Lidar, which limits its capability of mimicking the fidelity of real-world sensor data, particularly for industrial applications. Also, it focuses on single-object scans which hinders its usage in more complex multi-object annotation scenarios

Advanced annotation datasets: PartNet Mo K.. et al. (2019) contributes to the field by introducing part-level annotations: in addition to labeling each individual object, the dataset

contains hierarchical information on the parts that comprise every object. This approach enables part-level object understanding using deep-learning models. The dataset contains 26,671 3D models made of 571,585 parts, which were collected from ShapeNetCore Chang A.. et al. (2015) and augmented with additional models from the 3D Warehouse. Data annotation was manually performed by a team of 66 trained professionals. While part-level annotations are promising in different industrial applications, nonetheless the authors show that state-of-the-art models do not perform well on fine-grained semantic segmentation. The authors highlight the need to further investigate the interplay between part-level annotation and whole object annotation, and the level of detail that is required in order to improve model performance. ScanNet Dai A.. et al. (2017) presents a framework to streamline the capture and annotation of RGB-D data for indoor scenes, resulting in a dataset with surface re- constructions, 3D camera poses, and instance-level semantic segmentation. This dataset is created by capturing RGB-D videos using a depth sensor and an iPads, followed by deep learning-driven surface reconstruction and camera pose estimation. The dataset is then manually annotated, resulting in instance-level segmentation and 3D CAD model alignment. This annotation process is especially useful for algorithms aimed at interpreting complex indoor environments. However, the quality of the annotations is not guaranteed since it was done using online crowd sourcing. In a follow-up study, ScanObjectNN Uy M.A.. et al. (2019) leverages the strengths of SceneNN Hua B.S. . et al. (2016) and ScanNet Dai A.. et al. (2017) to provide high-quality real point clouds for indoor scenes. It offers dataset variants that simulate varying levels of classification difficulty, ranging from the vanilla variant, which presents objects in their original form, to the background and perturbed variants, which introduce complexities such as surrounding background points or inaccurately sized bounding boxes. These variants are designed to closely replicate the challenges faced in real-world applications, effectively bridging the gap between dataset benchmarks and the unpredictability of actual environments. This work underscores the ongoing need for datasets that can effectively simulate the complexity and variability of real environments. However, as mentioned previously, creating such datasets from real point clouds is extremely challenging and time consuming. Hence the need for faster and more efficient solutions, namely the creation of synthetic data pipelines.

Table 1 summarizes the properties of existing 3D point cloud datasets.

Table 1: Comparing 3D datasets.

Dataset	# Models	# Categories	Annotations
ModelNet Wu Z.. et al. (2014)	151,128 models	660	Classification
ShapeNetCore Chang A.. et al. (2015)	51,300 models	55	Classification with parts annotation
PartNet Mo K.. et al. (2019)	573,585 parts in 26,671 models	24	Semantic, instance, and hierarchical segmentation
ScanNet Dai A.. et al. (2017)	1,513 objects	20	Camera poses, surface reconstructions, and instance segmentation
ScanObjectNN Uy M.A.. et al. (2019)	2,902 objects	15	Classification
OmniObject3D Wu T.. et al. (2023)	6,000 objects	190	Textured meshes, point clouds, images, videos
SensatUrban Hu Q.. et al. (2021)	7.6 km^2	13	Semantic segmentation
Semantics3d Hackel T.. et al. (2017)	4B points	8	Semantic segmentation
SemanticKITTI Behley J.. et al. (2019)	43,552 scans	28	Semantic segmentation

2.2 Synthetic Data Generation Pipelines

LiDAR simulations for autonomous vehicles: Recent advancements in synthetic 3D data generation have mainly focused on improving realism and automation, particularly in the context of LiDAR simulation for autonomous vehicles, e.g., Wang F.. et al. (2019) Manivasagam S.. et al. (2020) Yue X.. et al. (2018). The LiDARsim Manivasagam S.. et al. (2020) project challenges the conventional use of handcrafted virtual environments by proposing a method that draws from real-world data to better replicate the complexities of real-world scenarios. The method consists of two main stages: (i) creation of assets and (ii) simulation of the sensor data. The first stage involves building a catalog of 3D static maps and dynamic object meshes by capturing environmental data by driving a fleet of vehicles in various urban environments. The second stage uses ray casting to create a physics-based rendering of the scene, which is then refined using a deep neural network trained to introduce realistic deviations and sensor noise, resulting in highly realistic LiDAR simulations. In Wang F.. et al. (2019), the authors describe an automated method for generating synthetic LiDAR data using the CARLA, an open-source autonomous driving simulator Dosovitskiy A.. et al. (2017). This method uses virtual LiDAR sensors that employ ray casting mounted on an autonomous driving car inside CARLA, so the LiDAR data can be collected automatically without human intervention. The authors showed that training deep learning models with artificial data can lead to increased accuracy and generalization. In Yue X.. et al. (2018), the authors introduce a framework for generating point clouds and images from computer game environments, namely from Grand Theft Auto V, to be used for autonomous driving applications. They utilize ray casting to simulate LiDAR scans and offer a solution for data collection in user-configured scenarios. Similar to Wang F.. et al. (2019), the lidar is mounted on a car that is autonomously driven in the game world. However, the customization implemented for the scenes in both works in Yue X.. et al. (2018)Wang F.. et al. (2019) are limited to changing the number and color of cars and the basic environment variables like the weather and background, which is not sufficient to cover a wide range of use and edge cases.

Indoor room generation and flight simulations: More recently, ControlRoom3D Schult J.. et al. (2023) introduced a method for generating 3D indoor room meshes using semantic proxy rooms, providing an efficient alternative to manual 3D environment creation for AR/VR applications. This approach facilitates the design of diverse and plausible room meshes, and improves over previous methods by allowing the generation of multiple rooms from a single proxy, albeit with limitations in variety and manual proxy definitions. STPLS3D Chen M.. et al. (2022) addresses the challenge of creating large-scale annotated point clouds that blend real and synthetic environments. The authors develop a pipeline that simulates drone flights over city layouts and generates synthetic point clouds through a process that closely mirrors real data collection methods. First, the cities are procedurally generated using CityEngine and different 3D model variations for the buildings. Second, 2D images of the city are rendered based on a recorded drone flight path. Third and finally, 3D reconstruction is done using the images to generate point clouds. This solution opens new avenues for urban planning and machine learning research, allowing to combine both real and synthetic data in order to provide more layout realism in data generation.

To sum up, most data generation pipelines focus on LiDAR simulations, e.g., Wang F.. et al. (2019) Manivasagam S.. et al. (2020) Yue X.. et al. (2018), and make use of predefined scenes or proxy layouts Yue X.. et al. (2018) Schult J.. et al. (2023) and game simulators

Table 2: Comparing 3D synthetic data generation pipelines.

Pipeline	Techniques	Simulation Engine	Data	Application
LiDARSim Manivasagam S.. et al. (2020)	Ray casting	NA	Point clouds	Autonomous Driving
Wang et al. Wang F.. et al. (2019)	Ray casting	CARLA	Point clouds	Autonomous Driving
Yeue et al. Yue X.. et al. (2018)	Ray casting	GTA V	Point clouds	Autonomous Driving
Karur et al. Karur K. et al. (2022)	Ray casting	Unity	Point clouds	Autonomous Driving
ControlRoom3D Schult J.. et al. (2023)	Proxy layouts	NA	Meshes	Room generation
STPLS3D Chen M.. et al. (2022)	Layouts and photogrammetry	NA	Point clouds	Urban planning and drones

Wang F.. et al. (2019) which can limit variety and realism in the generated data. In contrast, 3DGENie relies on procedural generation to allow for increased and controlled variety, and uses Nvidia Omniverse as a powerful platform to allow more realism and support a wider range of data and simulations.

Table 2 summarizes the properties of 3D data generation pipelines.

2.3 Procedural Layout generation

To address the variety and realism limitations of existing generation pipelines, and instead of using existing simulators, predefined scenes, or static proxies to prime the generation process, procedural layout generation algorithms are used as a means to dynamically prime the generation process following the user's preferences and application scenarios. This subsection briefly reviews existing procedural layout generation solutions.

Layout generation in video games: Layout generation is mostly used in video game design, allowing the creation of dynamic and immersive worlds that enhance the player engagement and replayability, e.g., No Man's Sky, Minecraft, Sokoban, etc. It allows to streamline the game development process and introduces a level of unpredictability and variation in gaming scenarios, by automatically generating scene layouts that transition from conceptual designs to virtual environments. In the realm of gaming, this means creating immersive and interactive environments that capture players' attention. For instance, the author in Barriga N. (2019) discusses different methods used for Procedural Content Generation in gaming to create maps, ecosystems, levels, etc. The authors in Davern S.. et al. (2023) use procedural generation to produce narrative puzzles in adventure games, allowing to create a large database of puzzle items, actions, rules, and goals, that can be used to generate new puzzles and create new item properties that fulfil the users' needs. In Zakaria Y.. et al. (2022), the authors use procedural level generators with reinforcement learning to produce Sokoban 2D grid-based puzzles. Results show better quality and higher diversity puzzles compared with existing deterministic and long-short term memory generators. Empirical results in Kumaran V.. et al. (2023) show that the proposed solution can transform natural language descriptions into playable game levels that reflect their intended design objectives.

Enhancing virtual scene realism: In Bahrehmand A.. et al. (2017), the authors utilize procedural generation to optimize spatial layouts using quality metrics and user preferences, highlighting the potential for using procedural generation to fulfill specific user design objectives and produce highly personalized and functional virtual environments. This involves evaluating spatial configurations to optimize factors such as accessibility, visibility, and visual appeal, with the aim of ensuring that the generated virtual scenes meet both the practical requirements and aesthetic preferences of users. The authors show how procedural generation-based optimization is essential for enhancing the detail and realism of virtual

environments, making them more suitable for a range of applications including simulations, presentations, and interactive explorations. In Abdelmohsen S.. et al. (2017), the authors explore the potential of using procedural generation beyond gaming, applying it for urban planning and interior design. They utilize multiple heuristic algorithms to perform furniture placement within residential spaces, considering factors such as functionality, aesthetics, and user preferences. By automating this process, designers can efficiently generate multiple layout options, thereby facilitating informed decision-making and tailored design solutions.

Advancements in procedural layout generation show the technology's versatility and how it can be used in different fields, from virtual environment design for gaming, to urban planning and interior layout design, which is a desirable property for 3DGENie. By leveraging procedural generation 3DGENie aims to produce 3D scenes that are geometrically accurate, varied, and optimized for realism and customizability.

2.4 Generative AI

In recent years, generative AI for 3D data, including point clouds, has seen major breakthroughs Yang Z.. et al. (2022); Kong L.. et al. (2023); Yang G.. et al. (2019). Even though these methods have shown promising results in generating synthetic point clouds, nonetheless, they mainly focus on generating single objects rather than detailed scenes. In addition, the generated point clouds lack instance and semantic segmentation labels, which limit their usage in supervised computer vision tasks. Consequently, such methods are not currently used to create labelled synthetic datasets, and are outside of the scope of this work. However, combining and integrating generative AI models within existing synthetic data generation pipelines is a promising direction to be investigated in the future.

3 3DGENie Synthetic Data Generation Pipeline

We propose a new synthetic data generation pipeline called 3DGENie designed to generate controlled 3D point clouds. An overview of 3DGENie is depicted in Figure 1, and consists of three main steps: (i) layout generation, (ii) scene creation, and (iii) data generation. First, it uses procedural layout generation to produce region layout trees. Second, it applies 3D scene construction and asset randomization to produce scenes populated with 3D assets. Third, it places synthetic sensors in the virtual environment to simulate data capture from the 3D scenes as if monitored by real-world sensors.

3.1 Layout Generation

The first step of the pipeline is layout generation, which lays the foundation for the 3D scenes that will be constructed in subsequent steps. Unlike synthetic images which can be gathered in bulk from a single scene, we can only generate a single point cloud scan from a scene, which is a significant limitation for 3D synthetic data generation. To address this issue, we propose using layout generation to automatically generate thousands of layouts from simple user input. Users can combine different generation techniques to cover different requirements. For instance, in our present use case where we replicate an automotive assembly line environment, we use a mix of algorithms for procedural map generation to achieve the desired

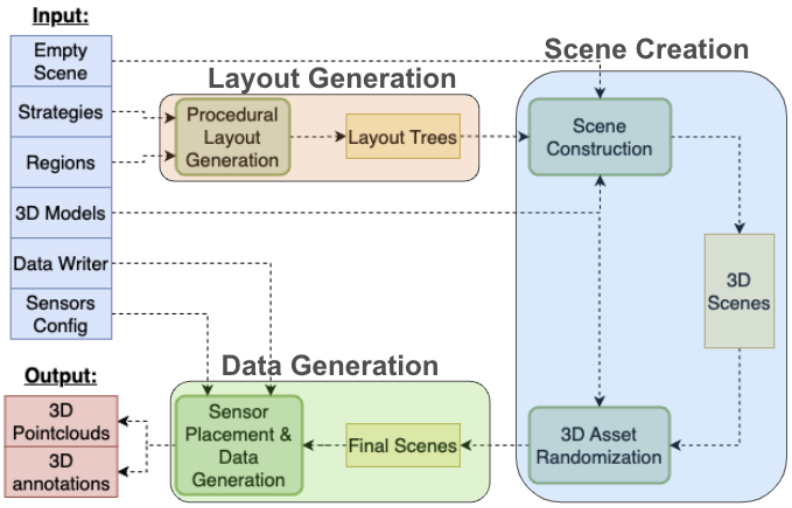


Fig. 1: 3DGENie data generation pipeline.

layouts. Most importantly, 3DGENie is extensible to using additional or alternative generation techniques, such as evolutionary, generative, or adversarial AI models, following the user's needs.

3.1.1 Layout Generation Components and Properties

We start first by introducing the main components and properties that are used in our Layout Generation process.

Component 1. Layout- It describes the different regions that make out the virtual environment, and the spatial relations between them in 2D space (Figure 2. b). We represent a layout as a list of regions, organized hierarchically in a tree where each node can have zero or multiple children. A layout acts like a blueprint for constructing the 3D scenes.

Component 2. Region- It is a rectangular area defined by its position in 2D space (x, y) and dimensions (w, h). A region has a region type and an orientation (described below), forming the building block of a scene layout and a main component of the layout generation algorithm.

Property 1. Region type- It describes the content of a region and is visualized throughout this work as the color of the region. Region types are defined by the user in the form of an input and can be linked to a specific group of 3D models. In addition, a region type can be either 'final' or 'non-final'. A region with a final region type cannot be further divided by the algorithm, whereas a non-final region can be further divided into smaller final or non-final regions.

Property 2. Region orientation- regions are inherently oriented in 2D space with the following values: "up", "down", "left" or "right", allowing to play a major role both when generating children regions and when building the final 3D scene. For example, if we generate a path for smart transportation robots (STR) Nassif J.. et al. (2024) and then divide it further

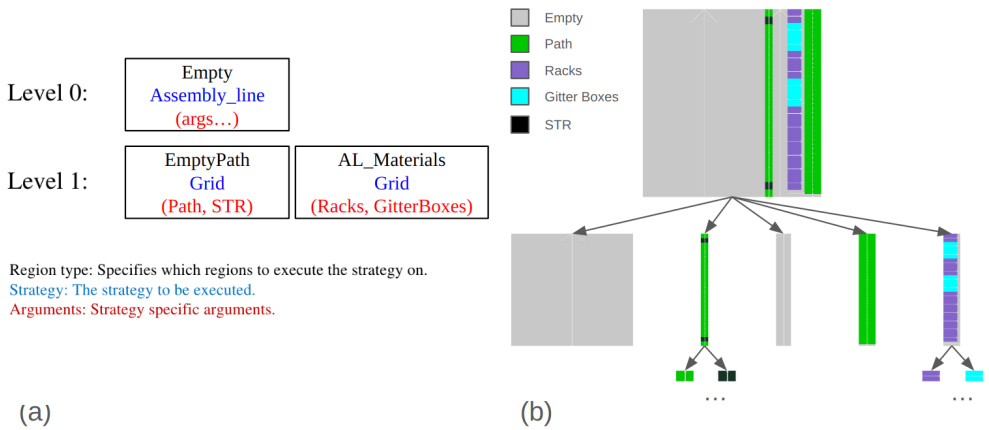


Fig. 2: Sample input list representation (a) and output layout tree (b) for the layout generation algorithm

into regions where we have an STR, it is crucial to know the orientation of the path in order to correctly orient the children's region accordingly.

3.1.2 Region Generators

These are functions that take as an input a region and divide it into a list of smaller children regions. Region generators exhibit a stochastic behavior, so if they are executed multiple times using the same parameters, the outputs would be different. The accumulation of this randomness over multiple generation steps allows to generate different layouts from a single input. In our current implementation, we consider three kinds of region generators and their use cases for our industrial applications (Figure 3). Our pipeline is extensible to more generators as needed.

Generator 1. (Figure 3. Assembly line) - it creates the area where an assembly line will go, adding a path for forklifts and STRs running parallel to the assembly line. This is usually the first generator executed to create an assembly line region spanning the entire scene. It randomizes the position as well as the orientation of the main assembly line, and also randomizes the number of paths and their spacing.

Generator 2. (Figure 3. Random Rooms) - given a range of room sizes (min_width, max_width, min_height, and max_height) and the number of rooms (min, max), this generator places rooms randomly inside the parent region. We use this generator to populate empty regions with different formations of pallet cages, boxes, and racks. This generator is mainly used in the initial stages of the generation process to roughly define large areas which will be divided further down the line to add more details.

Generator 3. (Figure 3. Grid)- it divides the parent regions into a grid with a user specified cell size, where each cell is converted into a region with the appropriate type and orientation. A common use case for this generator is to create the arrangements of stackable pallet cages in the area designated for them. A useful strategy we identified based on our test runs is to use a random rooms generator followed by a grid generator.

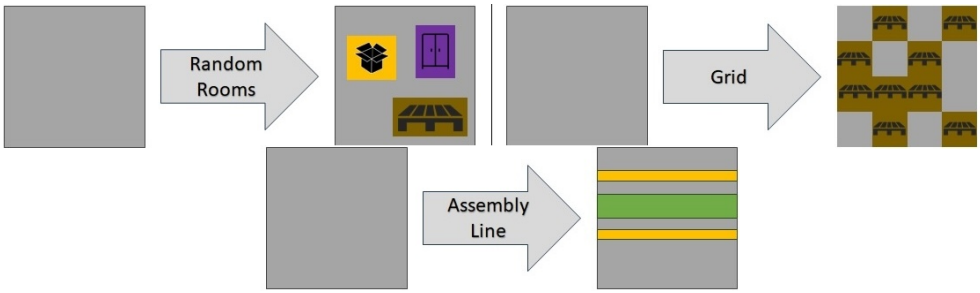


Fig. 3: Visualization of different region generators.

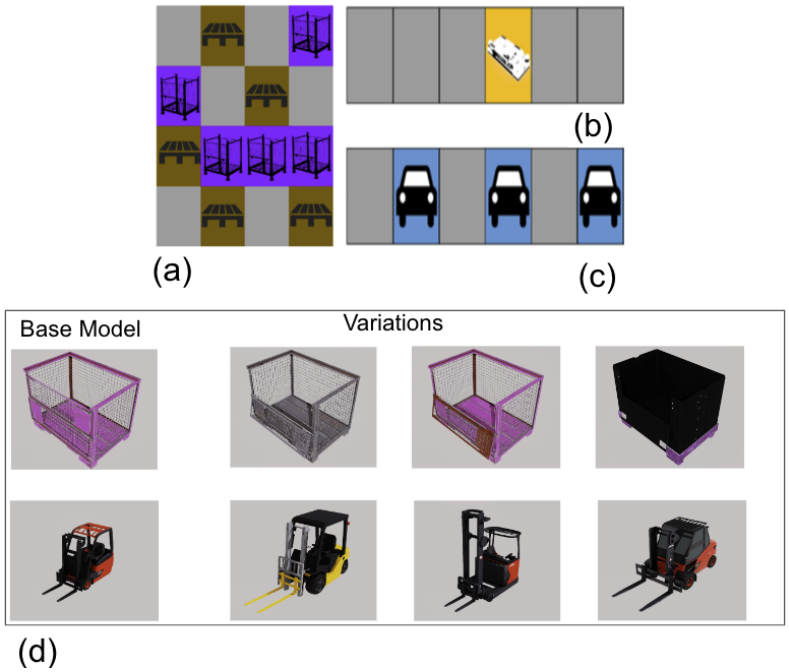


Fig. 4: (a) Use-case 1 (b) Use-case 2 (c) Use-case 3 (d) Samples from SORDI library.

While the output of different region generators can vary widely, we introduce region property providers, including region type providers and region orientation providers allowing to randomize and control region properties based on user preferences and the parent region on which the region generator is operating. For instance, to create a storage pile area arranged in a grid and containing random collections of objects (Figure 4. a), we use a grid region generator with a random region type provider. However, to create a lane in the factory plan for smart transportation robots (STR) and forklifts (Figure 4. b), or to create an assembly line where we have alternating regions of a car then an empty space (Figure 4. c), we use the grid region generator with stochastic or sequential region type providers.

Algorithm 1 LayoutGeneration

Input: $inputFile$ is a JSON file for the input strategy
Output: The root node of the generated layout tree
Begin
1: $layouts \leftarrow [EmptyLayout]$
2: $generators \leftarrow extractGenerators(inputFile)$
3: **for** $idx \in generators$ **do**
4: **if** $generators[idx] == Merge$ **then**
5: $mergedLayout \leftarrow Merge(layouts[-1])$
6: $layouts.append(mergedLayout)$
7: **else**
8: $generatedLayout \leftarrow$ **ExecuteGenerators**$(layouts[-1], generators[idx])$
9: $layouts.append(generatedLayout)$
10: **return** $layouts[-1]$
End

3.2 Layout Generation Algorithm

The pseudo-code for the procedural layout generation process is described in Algorithm 1. It accepts as input a list of elements where each element represents a level in the generation process, starting from the higher (broader) levels and going toward the lower (and more detailed) levels. Every element in the list, i.e., every level description, is represented as a key-value dictionary where: the keys are the region types, and the values are the region generators (Figure 3). The first step in the generation process is to parse the input and build an equivalent generation dictionary composed of region generators with the correct input parameters which will be applied to an empty layout (Algorithm 1, lines 1-2). Adding a new region generator can be performed in a scalable and extensible manner following two steps: adding a new generator class, and then adding this class to the region generator dictionary. The second step consists in generating the output tree using the previously parsed input (Algorithm 1, line 8). Using a breadth-first approach (Algorithm 2, lines 2-5), the tree is traversed level by level until the maximum depth is reached (Algorithm 2, line 6). At each level, generate new regions based on the input (Algorithm 2, line 9). The generated regions are then used to build the tree it is traversed (Algorithm 2, lines 10-20). In addition, a special merge layer operation is introduced (Algorithm 1, lines 4-6) to solve a problem that could arise from defining multiple region generators for the same region type at different levels of the generation process. The merge layer operation allows identifying and merging identical and bordering regions into a more compact form regardless of the region generator used (Figure 3). This makes it easier to introduce and use new generators, thus improving the pipeline's extensibility.

The algorithm produces as output a 2D layout (Algorithm 1, line 12) that will serve as the foundation for constructing 3D scenes using the scene creation (step #2) of the pipeline. The output layout consists of a tree structure where each node represents a region, and its child nodes represent the regions that result from the execution of a region generator on that node. For leaf nodes, 2D images of the appropriate size and color are generated. For a non-leaf node, the already generated images of tne node's children are combined using depth-first traversal.

Algorithm 2 ExecuteGenerators

Input: $root$ The root node of the starting tree, $generators$: Region generators
Output: The root node of the expanded layout tree
Begin
function $ExecuteGenerators(root, generators)$
    $queue \leftarrow EmptyQueue$
    $queue.put(root)$
    while $queue$ is not empty **do**
        $node \leftarrow queue.get()$
        if $node.depth \geq generators.size$ or $node.gen = None$ **then**
            continue
        else
            $regions \leftarrow node.gen.generate(node.region)$
            for $region$ in $regions$ **do**
                $gen \leftarrow None$
                for $node.depth + 1 < i < len(generators)$ **do**
                    if $region.type \in generators[i]$ **then**
                        $gen \leftarrow generators[i][region.type]$
                        break
                $childNode \leftarrow RegionNode(region, gen, node.depth + 1)$
                $node.addChild(childNode)$
                $queue.put(childNode)$
    return $root$
End

3.3 Scene Creation

Scene creation is step # 2 in the 3DGENie pipeline (Figure 1), following layout generation. It transforms the 2D layout trees into detailed and realistic virtual scenes (Figure 5. a, c). The main goal is to populate virtual scenes with 3D assets following the generated layout tree structure.

3.3.1 Scene Construction

We adopt Nvidia Omniverse Nvidia (2024a) as our scene building platform, since it leverages the latest advancements in GPU technology to allow for industry-grade scalability moving forward (in contrast with using legacy game engines used in existing solutions, cf. Section 2.2). In addition, Omniverse provides the required tools for building our extensions to parse the layout tree and assemble virtual 3D scenes. In addition, we use high-fidelity physics simulation Nvidia (2024b) and virtual sensors within IsaacSim Nvidia (2024a) during data generation (step #3 of the pipeline). We also adopt Pixar's Universal Scene Description (USD) Nvidia (2024c) for 3D graphics representation to allow for seamless interchange across platforms.

In addition, we use BMW Group's SORDI library Abou Akar C.. et al. (2024) (Figure 4. d), which includes a comprehensive collection of realistic and simulation-ready 3D assets that cover a wide range of industrial objects. Each region in the layout is associated with a set of assets, we randomly choose one of these assets when creating the region to introduce more variety. Thus, our scenes are not only detailed and realistic, but also diverse, reflecting the complexity of real-world industrial environments.

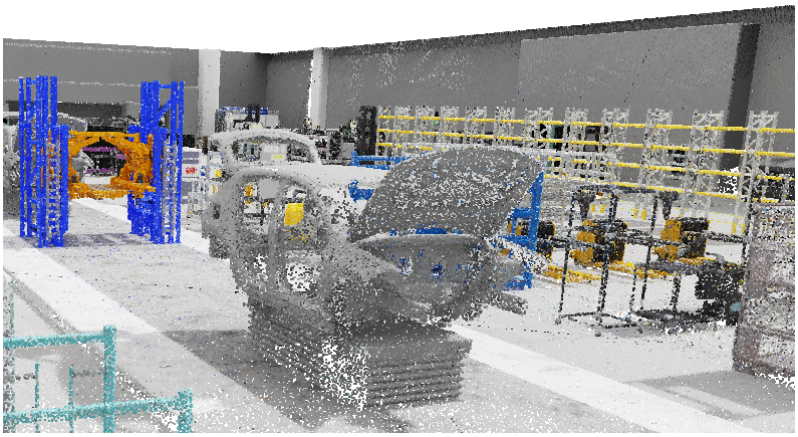
3.3.2 Scene Randomization

Randomization plays an important role in breaking patterns and biases that may arise in a scene, which will have a negative effect on machine learning models that use the generated data. By introducing randomization to an asset's placement and properties, we improve our synthetic data and support the development of robust machine learning models. Randomization can also help simulate the unpredictable nature of real-world scenarios. In this context, we make use of IsaacSim Replicator Nvidia (2024a) to introduce additional randomization by modifying various aspects of the scene, including, but not limited to, the visibility, arrangement, and colors of objects.

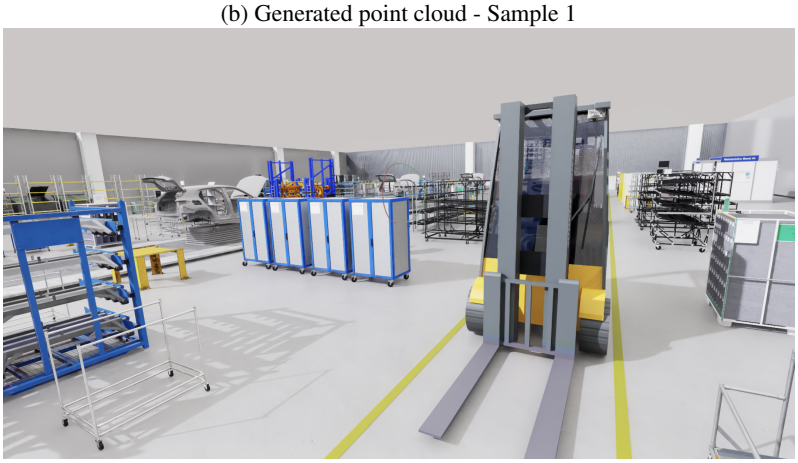
3.4 Data Generation

The third and last step of the 3DGENie pipeline is data generation, which enables the generation of not only point clouds but also photo-realistic images and other forms of data. The process of data generation is twofold: (i) sensor placement, and (ii) data collection and storage.

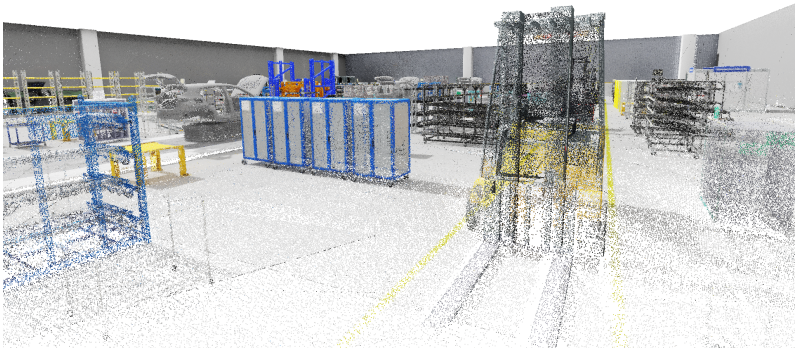
(a) Reconstructed scene - Sample 1



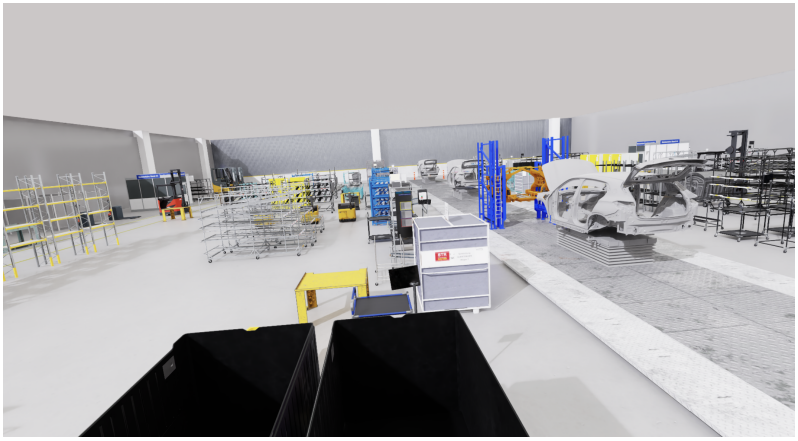
(b) Generated point cloud - Sample 1



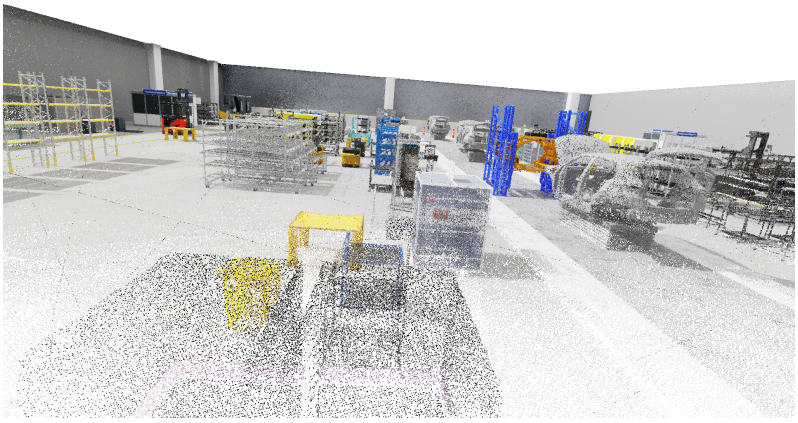
(c) Reconstructed scene - Sample 2



(d) Generated point cloud - Sample 2



(e) Reconstructed scene - Sample 3



(f) Generated point cloud - Sample 3

Fig. 5: Samples from 3DGENie: (a, c, e) Reconstructed scenes; (b, d, f) Generated point clouds.

3.4.1 Sensors placement

In our current use case focusing on generating car assembly lines, we want to generate point clouds that cover the whole scene to maximize the amount of the collected data. We use the 2D layout generated in step #1 of 3DGENie as a basis for strategically positioning sensors within the scene. To achieve optimal placement, we employ a genetic algorithm characterized by the following parameters: *sensor:* a circle with a center and a fixed radius, *x:* desired number of sensors, *chromosome:* list of x sensor centers, *fitness:* evaluated based on the union of covered pixels and their region types. It is possible to use the same scenes generated from the first two steps of the pipeline to generate synthetic data in different ways.

3.4.2 Data Collection and Storage

Consequently, 3DGENie generates the point cloud data and converts them into a suitable storage format. This includes not only raw sensor outputs like points in 3D space in the case of LiDAR, but also the annotations required to train machine learning models. 3DGENie converts the raw data produced within Omniverse into formats that are usable for training machine learning models, supporting an extensible library of formats including SORDI Abou Akar C.. et al. (2024) and Semantic KITTI Behley J.. et al. (2019). This ensures that the datasets generated by 3DGENie are compatible with existing models, frameworks, and tools.

3.5 Complexity Analysis

The complexity of the 3DGENie pipeline comes down to $O(L \times R^L)$, where R is the average branching factor, i.e., the number of regions a generator produces, and L is the number of levels in the layout tree, and simplifies to $O(R^L)$. It comes down to the complexity of its main `LayoutGenerator` (Algorithm 1) and its constituent `ExecuteGenerators` (Algorithm 2). More formally:

- Let L be the number of levels,
- Let G be the average number of generators per level,
- Let R be the average branching factor, i.e., the number of regions a generator produces,
- Let D be the depth of the tree,
- Let N be the total number of nodes in the tree. With each node being expanded only once per level, the total number of nodes N is determined by the branching factor R and the number of levels L, as a geometric series which sums to:

$$N = \frac{R^L - 1}{R - 1}$$

Time Complexity for Algorithm 2 - `ExecuteGenerators`: The complexity for processing each node involves generating regions (cf. Algorithm 2, 2 line 9), finding the appropriate generator (cf. lines 10-17) and finally creating the tree nodes (cf. lines 18-20). Considering that region generation requires $O(R)$, given that finding the appropriate generator is a dictionary lookup which requires $O(1)$, and given that the creation of the node required $O(1)$, hence, the time complexity for each node becomes $O(R)$. Since there are N nodes in the tree, the total time complexity for `ExecuteGenerators` becomes:

$$O(N \times R) = O(\frac{R^L - 1}{R - 1} \times R) = O(\frac{R^{L+1} - R}{R - 1})$$

This simplifies to $O(R^L)$. Note that, based on our experiments L usually has a lesser value ($\in [2, 5]$) compared with R ($\in [10, 20]$).

Time Complexity for Algorithm 1 - `LayoutGeneration`: The algorithm first parses the input (cf. Algorithm 1, lines 1-2) then iterates over the layers (cf. line 3), applying either the `Merge` (cf. lines 4-6) operation or `ExecuteGenerators` (cf. lines 8-9). Initializing the `layouts` list and extracting generators from the `inputFile` are constant-time operations,

$O(1)$. The `for` loop iterates over each generation level, so it runs L times, where L is the number of levels. Thus, the time complexity for `LayoutGeneration` becomes:

$$O(L \times R^L)$$

4 Experimental Evaluation

We have empirically tested 3DGENie to evaluate the quality of its synthetically generated 3D point clouds. The goal of our evaluation is to verify the following hypothesis: 3DGENie can generate synthetic point clouds capable of enhancing the accuracy of semantic segmentation models in real-world tasks. We conducted multiple experiments which we organize in two groups: (i) mixing real and synthetic data, and (ii) training on synthetic and fine-tuning on real Data. We first describe our test data and the models used, and then we present our empirical results. The system implementation, experimental datasets, and test results are available online [1].

4.1 Experimental Data

4.1.1 Real Data

A dataset of real 3D point cloud scans was prepared from car assembly lines. The scans were created using the NavVis VLX 2 NavVis (2024) wearable laser scanning system, capable of generating colored and high-density point clouds. The scans were labelled manually by industry experts, using a dedicated point cloud labelling tool, developed in Omniverse, providing user friendly controls for navigation and 3D manipulation such as translation, rotation, and scaling. To maintain an acceptable point density, each scene is cropped into smaller chunks, and then random down-sampling to 25,000 points is performed on each. The final down-sampled real dataset comprises 1,224,995 points organized in 10 classes (cf. Table 3). We use 50% of the data for training and the other 50% as a test dataset.

4.1.2 Synthetic data

We used 3DGENie to generate the synthetic point cloud dataset. To create the input strategies, we studied the layout of different areas within multiple car manufacturing plants, by visiting the sites in person and capturing multiple images and videos. We also used the available 3D scans created for the car assembly lines as references. Consequently, we generated 499 virtual scenes, each scene covered using 40 cameras configured to capture point clouds with a similar density to real data. We cleaned the data by removing point clouds containing a small number of points or a small percentage of labelled points. In addition, we used the generated semantic segmentation to remove any point clouds that contain only background points. We randomly selected a sub-sample of the point clouds, requiring an acceptable training time of around 10 hours on average per model, using an Nvidia A100 GPU. The resulting synthetic dataset comprises 19,849,968 points and covers the 10 classes considered in the real dataset (cf. Table 3).

[1] Please contact the authors to request access to the data and implementation
[2] The point cloud distribution among the classes reflects the real-world distribution of the assets in the scanned factory floors.

Table 3: Real and synthetic point cloud datasets[2].

Class Name	# points in real dataset	# points in synthetic dataset
Background	9,98,260 (81.5%)	4,931,280 (25%)
Pallet	14,303 (1.17%)	254,597 (1.28%)
Rack	8,574 (0.7%)	2,797,079 (14%)
Car	8,933 (0.73%)	4,131,134 (20.8%)
Jack	14,090 (1.15%)	2,531 (0.013%)
Stillage	70,972 (5.8%)	5,096,091 (25.67%)
Storage cabinet	87,646 (7.15%)	2,270,590 (11.4%)
Dolly	1,400 (0.11%)	56,680 (0.25%)
Forklift	10,862 (0.87%)	70,289 (0.35%)
Small load carrier	9,955 (0.81%)	239,697 (1.2%)
Total	**1,224,995**	**19,849,968**

4.2 Semantic Segmentation Models

We selected three different semantic segmentation models, each known for its unique approach to processing point clouds, in order to assess the effectiveness of the generated synthetic data in improving model performance.

RandLA-Net Hu et al (2019): introduces an architecture that directly infers per-point semantics for large-scale point clouds. It uses random point sampling, which is computationally efficient, but can discard key features by accident. To overcome this, the model introduces a novel local feature aggregation module that progressively increases the receptive field for each 3D point, effectively preserving geometric details.

SparseConvNet Graham B.. et al. (2018): stands out for its use of sparse convolutional operations, enabling it to process sparse point clouds efficiently. The authors introduced a novel sparse convolutional operation tailored specifically to process sparse data. SparseConvNets offers the same performance as other state-of-the-art convolutional models at a lower computational cost, making it a good candidate for our experiments that use large synthetic datasets.

PVCNN Liu Z.. et al. (2019): combines the efficiency of point-based processing with the structural advantages of volumetric convolutions, and offers a balanced approach to semantic segmentation. The PVCNN model is capable of achieving high accuracy at lower memory usage, making it computationally and memory efficient.

By using models with vastly different architectures, we aim to prove the value of the generated synthetic point clouds across a wide range of applications. In addition, we analyse and compare the performance of those models in a real-world industrial application, providing new insights for future research.

4.3 Semantic Segmentation Models

We used three different semantic segmentation models, each known for its unique approach to processing point clouds, in order to assess the effectiveness of the generated synthetic data in improving model performance. **RandLA-Net** Hu et al (2019): introduces an architecture that infers per-point semantics for large-scale point clouds. It uses random point sampling, which is computationally efficient, but can discard key features by accident. To overcome this, the model introduces a novel local feature aggregation module that progressively increases the

receptive field for each 3D point, effectively preserving geometric details. **SparseConvNet** Graham B.. et al. (2018): stands out for its use of sparse convolutional operations, enabling it to process sparse point clouds efficiently. The authors introduced a novel sparse convolutional operation tailored specifically to process sparse data. SparseConvNets offers the same performance as other state-of-the-art convolutional models at a lower computational cost, making it a good candidate for the experiments that use large synthetic datasets. **PVCNN** Liu Z.. et al. (2019): combines the efficiency of point-based processing with the structural advantages of volumetric convolutions, and offers a balanced approach to semantic segmentation. The PVCNN model is capable of achieving high accuracy at lower memory usage, making it computationally and memory efficient.

By using models with different architectures, the experiments aim to prove the value of the generated synthetic point clouds across a wide range of applications. In addition, we assess their performance in a real-world industrial application, providing new insights for future research.

4.4 Experimental Results

We conducted two sets of experiments: (i) mixing real and synthetic data, and (ii) training on synthetic data and fine-tuning on real data.

4.4.1 Experiment 1: Mixing Real and Synthetic Data

- The first set of experiments investigate how adding synthetic data, at varying proportions, affects the performance of different semantic segmentation models. It aims to evaluate the impact of using synthetic point clouds in the context of limited real-world datasets. To do so, 10 training datasets are created by varying the proportion of synthetic-to-real data: starting from 0% synthetic (purely real) and increasing by 10% increments up to 90% synthetic. To compensate for the different dataset sizes due to the incremental addition of synthetic data, the number of training epochs is adjusted to ensure model convergence. A testing dataset with 0% synthetic data is used to test the models performance. Each model is trained 2 different times with on each training dataset, including a total of 60 training runs. Model weights and sample point clouds with inferred semantics are saved during frequent checkpoints, where the best checkpoint from each run is used for testing. The below evaluation metrics are used, averaged across the different training runs:

- Acc (Accuracy) per class: ratio of correctly predicted points for each class over the total number of points in the class, highlighting class-wise predictive success.
- IoU (Intersection over Union) per class: where A is the set predicted points, and B is the set of ground truth points for a specific class.
- mAcc: mean accuracy across all classes.
- mIoU: mean IoU across all classes.

Results: The mAcc and mIoU results in Table 4 and Fig. 6 clearly show that mixing real and synthetic data improved the performance of all models when compared with training on real data only. All models exhibit the same behavior: an increase in performance over a range of the synthetic data ratio and a sharp decline in performance if we keep adding more synthetic data. For RandLaNet, the range goes from 40% to 60%, with 40% showing the best

Table 4: mAcc and mIoU of the models across the datasets variant mixing real and synthetic data.

Ratio	mAccuracy			mIoU		
	RandLaNet	PVCNN	SparseConvUNet	RandLaNet	PVCNN	SparseConvUNet
0%	0.20	0.33	0.33	0.15	0.25	0.23
10%	0.19	0.31	**0.38**	0.14	0.24	**0.30**
20%	0.20	0.33	**0.35**	**0.16**	0.25	**0.23**
30%	0.16	0.32	**0.34**	0.11	0.24	0.22
40%	**0.25**	**0.36**	0.26	**0.20**	**0.27**	0.16
50%	0.20	0.30	0.17	0.13	0.23	0.11
60%	**0.24**	0.18	0.31	**0.17**	0.14	0.19
70%	0.10	0.20	0.12	0.08	0.15	0.07
80%	0.17	0.19	0.10	0.15	0.13	0.07
90%	0.02	0.11	0.04	0.02	0.07	0.03

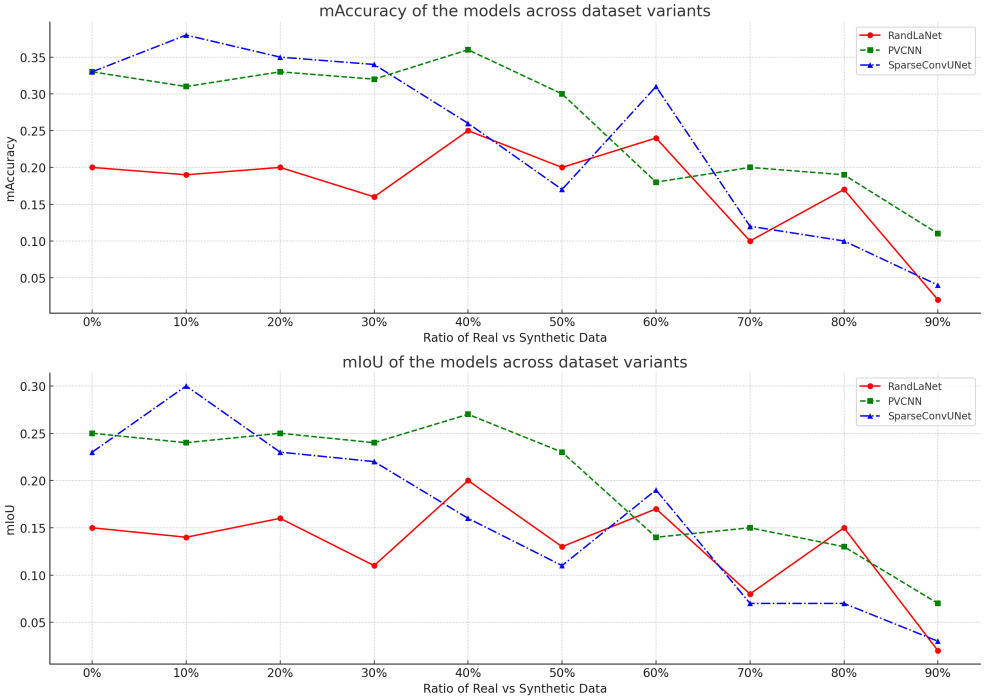


Fig. 6: Graphical results of Experiment 1.

performance. PVCNN performs best with 40% synthetic data. SparseConvUnet benefits from adding 10% to 30% of synthetic data, with the best performace being at 10%.

Table 5: mAcc and mIoU of the models after following a pre-train and fine tune process.

Epochs	mAccuracy			mIoU		
	RandLaNet	PVCNN	SparseConvUNet	RandLaNet	PVCNN	SparseConvUNet
0	0.02	0.09	0.05	0.01	0.04	0.04
20	0.10	0.15	**0.16**	0.09	0.14	**0.07**
40	0.06	0.14	0.14	0.04	0.13	0.06
60	0.08	0.15	0.13	0.04	0.13	0.06
80	0.06	0.18	0.14	0.04	**0.16**	0.06
100	0.08	0.15	0.12	0.05	0.13	0.05
200	0.15	0.10	0.10	**0.11**	0.09	0.04
300	**0.22**	**0.24**	0.04	0.03	0.05	0.02

4.4.2 Experiment 2: Train on Synthetic and Fine-tune on Real Data

Here, another strategy for using synthetic point clouds is examined, by first training on purely synthetic datasets then fine-tuning on a small real dataset. The aim is to evaluate the impact of training on synthetic data only, versus training purely on real data. The number of training epochs is varied during the first stage, starting from a low number of 20 epochs, and gradually increasing it until the models converge. Here, all of the synthetic dataset is used for training. The models are fine tuned by training them on a dataset of real point clouds only, until they converge. Similarly, all of the real dataset is used for fine-tuning. Each model is trained 5 times on the synthetic dataset, with different numbers of epochs. The latest checkpoint from each training run is used as a starting point for fine-tuning.

Results: All three models achieved better results compared with training on real data only, as can be seen in Table 5 and Fig. 7. RandLaNet Hu et al (2019) witnessed an increase of 13% for mAcc and 10% for mIoU. The performance of SparseConvNet Graham B.. et al. (2018) improved by 11% for mAcc and 3% for mIoU.As for PVCNN Liu Z.. et al. (2019), we achieve an increase of 9% and 12% in mAcc and mIoU respectively. Performance steadily increased as we increased the number of training epochs in the first stage, then it reached a tipping point where training the models further led a to a decrease in performance following the fine-tuning phase. This behavior is not clear if we look only at the mAcc because the high percentage of background points is dominating, which is why IoU is usually a better metric to use for evaluating segmentation tasks. By examining the results, we realized that increasing the number of training epochs allowed the models to converge to a more specific and local solution suitable for the synthetic data, which explains why fine-tuning in the second stage became less effective here.

4.4.3 Extreme Label Scarcity

Here we repeated Experiments 1 and 2 in a more extreme scenario. We considered a realistic logistics scenario with the following 3 classes: *pallet*, *rack*, and *small load carrier*. In doing so, we reduce the number of labeled data to 2.68% of the total real points. We then repeated Experiment 1 to test if the synthetic data generated by 3DGENie can still prove useful. The results in Table 6 and Fig. 8 show that all models produce a gain of 5% to 10% over only real data. The best performance for all the models is achieved when using all the synthetic data, where this effect kicks in and becomes clearly noticed around the 80% ratio.

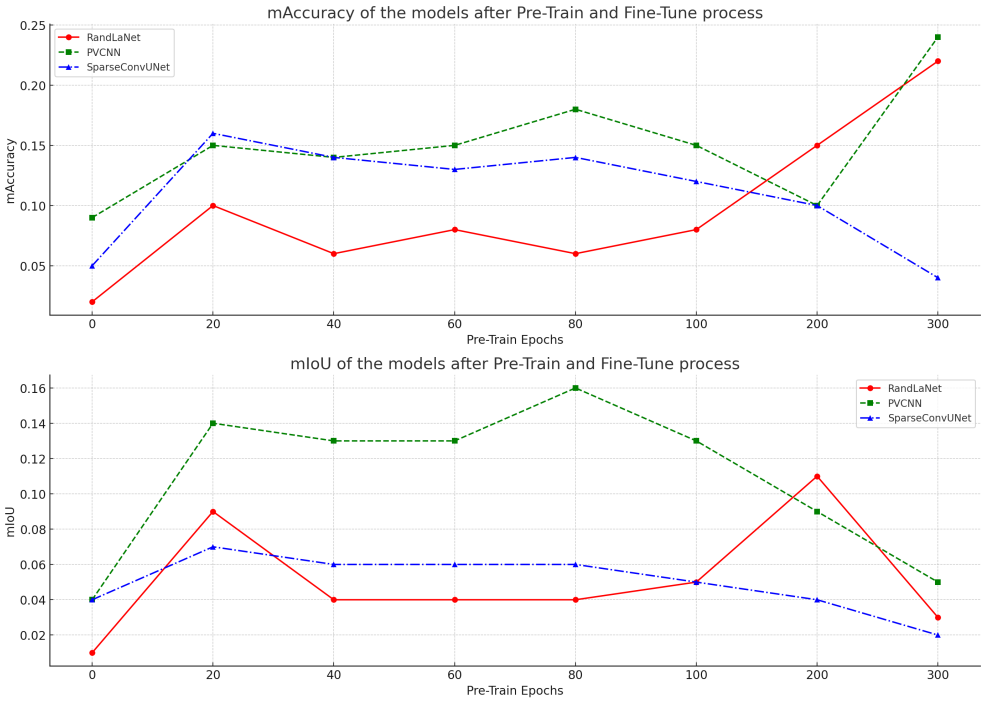


Fig. 7: Graphical results of Experiment 2.

Table 6: mAcc and mIoU under extreme label scarcity.

Ratio	mAcc Gain (%)			mIoU Gain (%)		
	RandLaNet	PVCNN	SparseConvUNet	RandLaNet	PVCNN	SparseConvUNet
10%	-0,03	-1,67	-0,18	2,01	-2,33	-0,16
20%	0,11	-0,42	-1,39	1,89	-0,56	-1,44
30%	-1,08	-3,31	-3,54	1,39	-3,19	-3,79
40%	-1,41	-4,41	-1,92	0,87	-10,7	-2,31
50%	-0,14	-1,66	-3,10	1,49	-1,96	-3,31
60%	-0,24	-6,31	-3,07	1,39	-6,68	-3,24
70%	21,77	-1,01	-2,08	12,45	-1,77	-2,17
80%	-0,15	23,26	3,24	1,72	-0,27	2,38
90%	0,40	7,97	7,21	1,72	-2,57	5,02
100%	**24,53**	**34,35**	**18,90**	**21,31**	**8,49**	**16,03**

4.4.4 Complexity Analysis Experiment

We previously derived the time complexity of our approach by analyzing the different algorithms. In this section, we conduct the required experiments to prove that our previous analysis is correct. A dummy region generator is developed whose time complexity is proportional to its branching factor, we then conduct multiple runs by fixing L and varying R or the other way around. We keep track of the CPU time for each run and report the results in Table

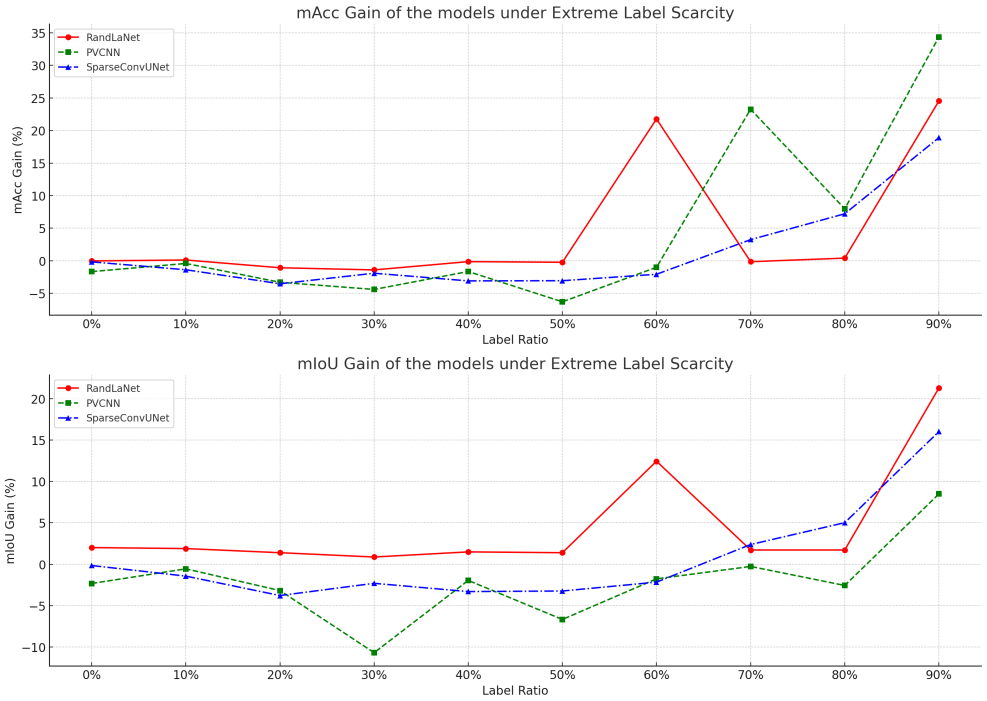


Fig. 8: Graphical results for the extreme label scarcity experiment.

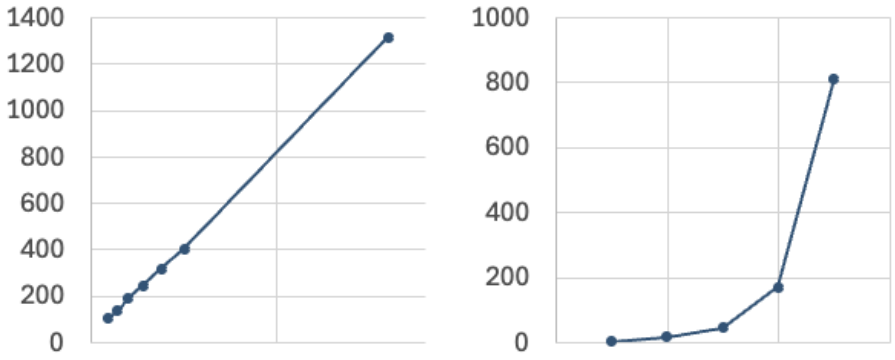
7 and Fig. 9. As can be seen, especially from the graph in Fig. 9, the execution time is linearly proportional to R and exponentially proportional to L, confirming our previous finding that the complexity of our algorithm is $O(L \times R^L)$.

L = 4			R = 5	
R	R^L	Time (ms)	L	Time (ms)
10	10000,00	103	3	1
11	14641,00	135	4	16
12	20736,00	188	5	43
13	28561,00	242	6	168
14	38416,00	316	7	810
15	50625,00	400		
20	160000,00	1313		

Table 7: Results of complexity analysis.

5 Conclusion

This paper introduces 3DGENie, a pipeline for synthetic 3D point cloud data generation. It uses procedural generation to produce region layout trees, and applies 3D scene construction



(a) R^L (b) L

Fig. 9: 3DGENie execution time in function of R and L.

and asset randomization to produce scenes with 3D assets. Synthetic sensors are used to simulate data capture from the 3D scenes as if monitored by real-world sensors. 3DGENie uses Nvidia Omniverse Nvidia (2024a) as its scene building platform and Pixar's Universal Scene Description (USD) Nvidia (2024c) for 3D graphics representation to allow for seamless interchange across platforms. We conducted various experiments to evaluate the performance of multiple computer vision models performing both training and fine-tuning using 3DGENie's synthetic data. Results consistently showed improved performance across all models. More importantly, our empirical study was conducted in a real-world car manufacturing setting, proving the value of synthetic point clouds for industrial applications.

We are currently extending 3DGENie to support additional forms of annotations to perform instance segmentation Vu T.. et al. (2023), object recognition Lu G.. et al. (2023), and 6D pose estimationZou L.. et al. (2024). We are also building on 3DGENie to generate a range of synthetic data types, including simulating LiDAR for capturing detailed geometry Manivasagam S.. et al. (2020) Yue X.. et al. (2018), and RGB-D sensors for applications requiring color and depth information Dai A.. et al. (2017). We also aim to develop a library of randomization techniques to enhance the realism of the generated data, including textures, occlusions, light conditions, and material degradation Nassif J.. et al. (2024) Abou Akar C.. et al. (2024), etc., where realism remains a primary requirement in industrial applications.

Declarations

Author Individual Contributions Statement: We confirm the authors' contributions in this work and their distribution of the tasks as follows: i) Anthony Yaghi (Ph.D. candidate): Conceptualization, Formal Analysis, Data Curation, Validation, Software, Implementation, Writing-Original Draft, ii) Joe Tekli (Ph.D. co-supervisor): Methodology, Formal Analysis, Validation, Writing-Original Draft, Writing-Review and Editing, Visualization, Supervision, iii) Marc Kamradt (industry supervisor): Conceptualization, Formal Analysis, Software, Validation, Supervision, and iv) Raphael Couturier (Ph.D. supervisor): Methodology, Formal Analysis, Writing-Review and Editing, Supervision, Project administration.

References

Geiger A.. et al. (2012) Are we ready for autonomous driving? the kitti vision benchmark suite. In: Comp. Vis. and Patt. Recogn. (CVPR'12), pp 3354–3361

Wu Z.. et al. (2014) 3d shapenets: A deep representation for volumetric shapes

Chang A.. et al. (2015) ShapeNet: An Information-Rich 3D Model Repository. CoRR abs/151203012

Hua B.S. . et al. (2016) Scenenn: A scene meshes dataset with annotations. In: Inter. Conf. on 3D Vision (3DV'16), pp 92–101

Dosovitskiy A.. et al. (2017) CARLA: An open urban driving simulator. In: Annual Conference on Robot Learning, pp 1–16

Abdelmohsen S.. et al. (2017) A heuristic approach for the automated generation of furniture layout schemes in residential spaces. In: Design Computing and Cognition'16, pp 459–475

Bahrehmand A.. et al. (2017) Optimizing layout using spatial quality metrics and user preferences. Graphical Models 93:25–38

Dai A.. et al. (2017) Scannet: Richly-annotated 3d reconstructions of indoor scenes. In: Comp. Vis. and Patt. Recogn. (CVPR'17), pp 2432–2443

Hackel T.. et al. (2017) Semantic3d.net: A new large-scale point cloud classification benchmark. In: ISPRS Annals of the Photogrammetry, pp 91–98

Graham B.. et al. (2018) 3d semantic segmentation with submanifold sparse convolutional networks. Comp Vis and Patt Recogn (CVPR'18) pp 9224–9232

Yue X.. et al. (2018) A lidar point cloud generator: from a virtual world to autonomous driving. Inter Conf on Multimedia Retrieval (ICMR'18) pp 458–464

Wang F.. et al. (2019) Automatic generation of synthetic lidar point clouds for 3-d data analysis. IEEE Trans on Instrum and Meas 68(7):2671–2673

Behley J.. et al. (2019) A dataset for semantic segmentation of point cloud sequences. CoRR abs/1904.01416

Mo K.. et al. (2019) PartNet: A large-scale benchmark for fine-grained and hierarchical part-level 3D object understanding. In: Comp. Vis. and Patt. Recogn. (CVPR'19), pp 909–918

Liu Z.. et al. (2019) Point-voxel cnn for efficient 3d deep learning. Advances in Neural Information Processing Systems 32

Yang G.. et al. (2019) Pointflow: 3d point cloud generation with continuous normalizing flows. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp

4540–4549, https://doi.org/10.1109/ICCV.2019.00464

Uy M.A.. et al. (2019) Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In: Inter. Conf. on Comp. Vis. (ICCV'19), pp 1588–1597

Morbidoni C.. et al. (2020) Learning from synthetic point cloud data for historical buildings semantic segmentation. J Comput Cult Herit 13(4):1–16

Manivasagam S.. et al. (2020) Lidarsim: Realistic lidar simulation by leveraging the real world

Xiao A.. et al. (2021) Synlidar: Learning from synthetic lidar sequential point cloud for semantic segmentation. CoRR abs/2107.05399. URL https://arxiv.org/abs/2107.05399

Chen R.. et al. (2021) Synthetic data in machine learning for medicine and healthcare. Nature Biomedical Engineering 5(6):493–497

Hu Q.. et al. (2021) Towards semantic segmentation of urban-scale 3d point clouds: A dataset, benchmarks and challenges. In: Comp. Vis. and Patt. Recogn. (CVRP'21), pp 4977–4987

Yang Z.. et al. (2022) Conditional gan for point cloud generation. In: Proceedings of the Asian Conference on Computer Vision, pp 3189–3205

Karur K. et al. (2022) End-to-end synthetic lidar point cloud data generation and deep learning validation. Tech. rep., SAE Technical Paper, https://doi.org/https://doi.org/10.4271/2022-01-0164

Zakaria Y.. et al. (2022) Procedural level generation for sokoban via deep learning: An experimental study. IEEE Transactions on Games 15(1):108–120

Chen M.. et al. (2022) Stpls3d: A large-scale synthetic and real aerial photogrammetry 3d point cloud dataset. In: British Mach. Vis. Conf. (BMVC'11), p 429

Schult J.. et al. (2023) Controlroom3d: Room generation using semantic proxy rooms. 2312.05208

Yunas M.. et al. (2023) Early diagnosis and personalised treatment focusing on synthetic data modelling: Novel visual learning approach in healthcare. Computers in Biology and Medicine 164:107295

Kumaran V.. et al. (2023) End-to-end procedural level generation in educational games with natural language instruction. In: IEEE Conference on Games (CoG'23), pp 1–8

Kong L.. et al. (2023) Generative models for 3d point clouds. arXiv preprint arXiv:230213408

Lu Y.. et al. (2023) Machine learning for synthetic data generation: A review. CoRR abs/230204062

Lu G.. et al. (2023) A novel method for improving point cloud accuracy in automotive radar object recognition. IEEE Access pp 78538–78548

Wu T.. et al. (2023) Omniobject3d: Large-vocabulary 3d object dataset for realistic perception, reconstruction and generation. Comp Vis and Patt Recogn (CVPR'23) pp 803–814

Vu T.. et al. (2023) Scalable softgroup for 3d instance segmentation on point clouds. IEEE Trans on Pattern Analysis and Machine Intell 46(4):1981–1995

Davern S.. et al. (2023) Towards procedural generation of narrative puzzles for open world games. In: Inter. Conf. on Interactive Digital Storytelling, pp 146–154

Zou L.. et al. (2024) Learning geometric consistency and discrepancy for category-level 6d object pose estimation from point clouds. Patt Recogn 145:109896

Abou Akar C.. et al. (2024) Sordi. ai: large-scale synthetic object recognition dataset generation for industries. Multimedia Tools and Applications pp 1–42

Nassif J.. et al. (2024) Synthetic Data: Revolutionizing the Industrial Metaverse. Springer Nature, 978-3-031-47560-3

Song Z.. et al. (2024) Synthetic datasets for autonomous driving: A survey. IEEE Transactions on Intelligent Vehicles 9(1):1847–1864. https://doi.org/10.1109/tiv.2023.3331024, URL http://dx.doi.org/10.1109/TIV.2023.3331024

Barriga N. (2019) A short introduction to procedural content generation algorithms for videogames. Inter J on Artificial Intelligence Tools 28(02):1930001

Hu Q, Yang B, Xie L, et al (2019) Randla-net: Efficient semantic segmentation of large-scale point clouds. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) pp 11105–11114. URL https://api.semanticscholar.org/CorpusID:208290898

NavVis (2024) Navvis vlx 2. https://www.navvis.com/vlx-2, access March'24

Nvidia (2024a) Isaac sim. https://developer.nvidia.com/isaac-sim, access March'24

Nvidia (2024b) Physx sdk. https://developer.nvidia.com/physx-sdk, access March'24

Nvidia (2024c) Pixar universal scene description. https://developer.nvidia.com/usd, access March'24