Procedural 3D point cloud generation pipeline for the industrial digital twin

Anthony Yaghi[1,2], Joe Tekli[3], Marc Kamradt[1], Raphaël Couturier[2], Charbel Bou Maroun[4], Elio Hanna[4], and Angelo Yaghi[4]

[1] BMW Group, TechOffice, Petuelring 130, 80809, Germany
{anthony.yaghi,marc.kamradt}@bmw.de
[2] University of Franche-Comté, FEMTO-ST, Besançon, 25030, France
raphael.couturier@univ-fcomte.fr
[3] Lebanese American University, E.C.E. Dept, Byblos, 36, Lebanon
joe.tekli@lau.edu.lb
[4] InMind .ai R&D, Beirut, Lebanon
{charbel.boumaroun, elio.hanna, angelo.yaghi}@inmind.ai

Abstract. This paper describes a new synthetic data generation pipeline called 3DGENie designed to generate 3D point clouds to train deep learning computer vision models. 3DGENie uses procedural layout generation to produce region layout trees. It then applies 3D scene construction and asset randomization to produce scenes populated with 3D assets. Synthetic sensors are placed in the virtual environment to simulate data capture from the 3D scenes as if monitored by real-world sensors. 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. Our main application focuses on the generation of industrial car assembly lines, yet 3DGENie can be used across different applications. We conduct experiments to evaluate the generated 3D point clouds, using several deep learning semantic segmentation models. Results highlight the quality of our pipeline.

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

1 Introduction

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, compared with their 2D counterparts [17]. 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). 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 [30] as its scene building platform which leverages the latest achievements in GPU technology, and Pixar's Universal Scene Description (USD) [32] 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.

The rest of the paper is organized as follows: Section 2 briefly reviews the related works. Section 3 describes our 3DGENie pipeline. Section 4 describes the experimental evaluation, before concluding with future directions in Section 5.

2 Related Work

We briefly cover real and synthetic point cloud datasets for machine learning, and synthetic data generation pipelines.

2.1 Point Cloud Datasets

Real-World datasets: SensatUrban [18] and Semantics3D [7] are legacy real-world datasets in the area of urban and natural scenes. SemanticKIITI [11] is based on the odometry of the KITTI benchmark [1], which is derived from a LiDAR mounted on a car as it travels various types of roads. While these datasets provide high-quality 3D point clouds, their production is extremely time-consuming and requires manual labor and resources.

CAD model-based datasets: ModelNet [2] and ShapeNet [3] are large labeled collections of 3D CAD models. While CAD-based datasets allow design flexibility and extensibility, they show various limitations, chiefly: i) the data representation does not resemble the output of a real sensor like a depth camera or LiDAR, and ii) the models being collected randomly from online sources, do not guarantee high-quality data. OmniObject3D [24] scans daily objects (in contrast with industrial objects) and generates point clouds from 3D meshes rather than direct capture from LiDAR, which limits its capability of mimicking real-world sensor data.

Advanced annotation datasets: PartNet [12] introduces part-level annotations. ScanNet [6] streamlines the capture and annotation of RGB-D data for

Table 1: Comparing 3D datasets.

Dataset	# Models	# Categories	Annotations
ModelNet [2]	151,128 models	660	Classification
ShapeNetCore [3]	51,300 models	55	Classification with parts annotation
PartNet [12]	573,585 parts in 26,671 models	24	Semantic, instance, and hierarchical segmentation
ScanNet [6]	1,513 objects	20	camera poses, surface reconstructions, and instance segmentation
ScanObjectNN [14]	2,902 objects	15	Classification
OmniObject3D [24]	6,000 objects	190	Textured meshes, point clouds, images, videos
SensatUrban [18]	$>$7.6 km^2	13	Semantic segmentation
Semantics3d [7]	$>$4B points	8	Semantic segmentation
SemanticKITTI [11]	43,552 scans	28	Semantic segmentation

indoor scenes, using a depth sensor and an iPad. In a follow-up study, ScanObjectNN [14] leverages the strengths of SceneNN [4] and ScanNet [6] to provide high-quality real point clouds for indoor scenes.

To sum up, creating datasets from real point clouds is extremely challenging and time-consuming. Hence the need for faster and more efficient solutions, namely synthetic data pipelines.

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

2.2 Synthetic Data Generation Pipelines

LiDAR simulations for autonomous vehicles: Recent advancements in synthetic 3D data generation have focused on producing LiDAR simulations for autonomous vehicles, e.g., [10] [15] [9]. LiDARsim [15] draws from real-world data to replicate real-world scenarios. In [10], [9], and [19], the pipelines use CARLA autonomous driving simulator [5] and the Unity 3D game engine. However, the customizations implemented in [10][9] are limited to changing the number and color of cars and basic environment variables like the weather and background.

Indoor room generation and flight simulations: ControlRoom3D [21] generates 3D indoor room meshes using semantic proxy rooms, albeit with limitations in variety and manual proxy definitions. STPLS3D [20] creates large-scale annotated point clouds that blend real and synthetic environments. Cities are first generated using CityEngine and different 3D model variations for the buildings. 3D reconstruction is done using the images to generate the point clouds.

To sum up, most existing data generation pipelines focus on LiDAR simulations, e.g., [10] [15] [9], and make use of predefined scenes or proxy layouts [9] [21] which can limit data variety. In other works [10], the data is generated from video games, which can negatively impact realism. 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.

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

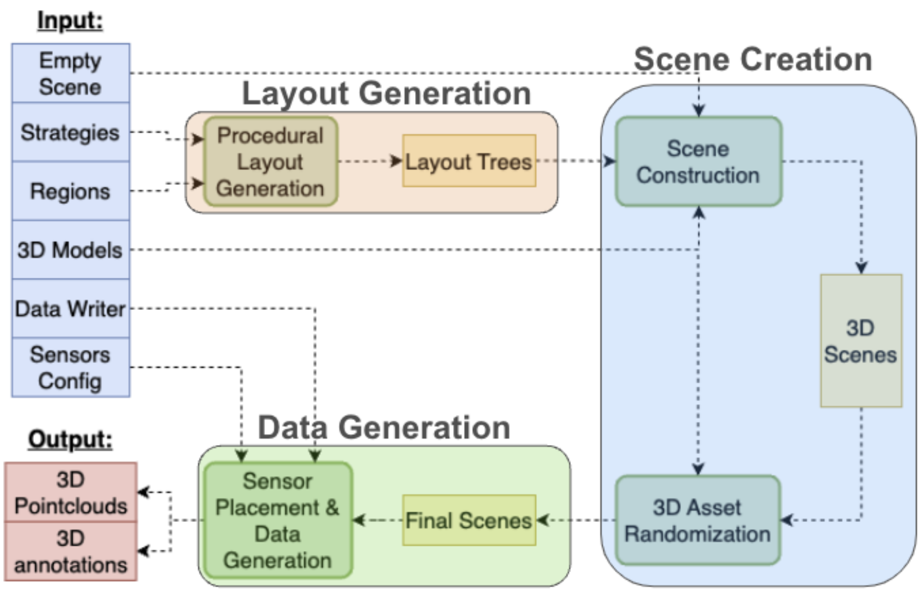


Fig. 1: 3DGENie data generation pipeline.

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 layouts Figure 2, which is a 2D description of the different regions that make up the final scene. 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. 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.

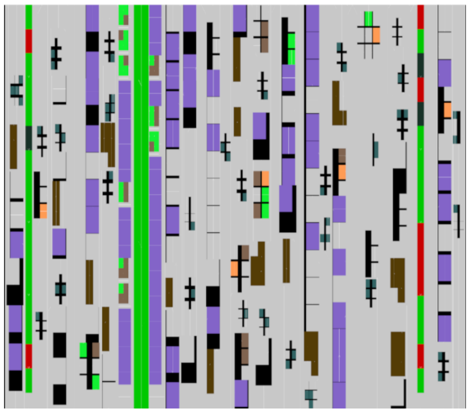


Fig. 2: Visualized layout

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 5. 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.

Property 2. Region orientation - regions are inherently oriented in 2D space with: "up", "down", "left" or "right", this plays 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) [27] and then divide it further 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.

Region Generators These are functions that take 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

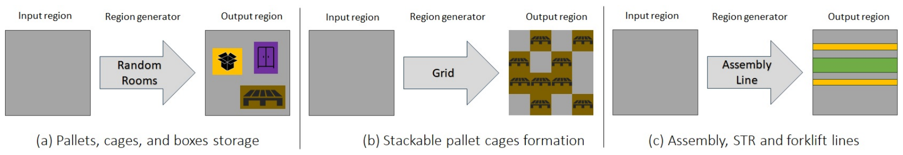


Fig. 3: Visualization of different region generators.

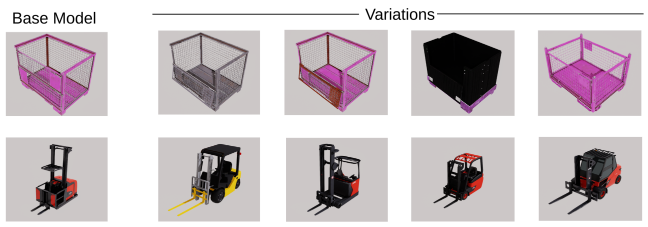


Fig. 4: Samples from the SORDI library.

for our industrial applications (Figure 3). Our pipeline is extensible to more generators as needed.

Generator 1. Assembly line (Figure 3. c) - 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. Random Rooms (Figure 3. a) - 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. Grid (Figure 3. b) - 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.

3.2 Layout Generation Algorithm

The pseudo-code for our procedural layout generation process is described in 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. This input is in the form of a JSON file written by the user once, and used to generate hundreds of scenes. Every element in the list, i.e., every level description, is represented as a key-value

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:     end if
11: end for
12: return $layouts[-1]$
    End
```

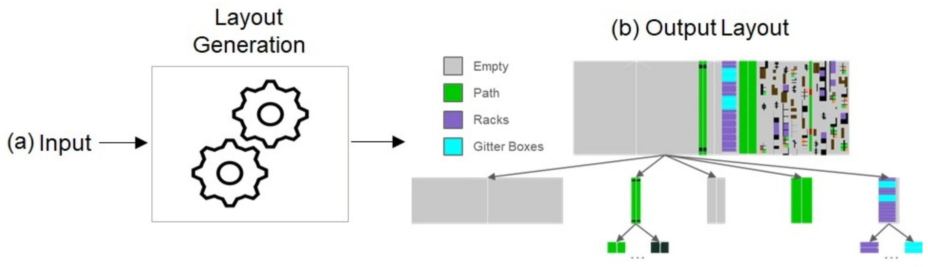


Fig. 5: Sample input list representation (a) and output layout tree (b) for the layout generation algorithm

dictionary where: the keys are the region types, and the values are the region generators. 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 (1, lines 1-2). The second step consists in generating the output tree using the previously parsed input (1, line 8). Using a breadth-first approach (2, lines 2-5), we traverse the tree level by level until the maximum depth is reached (2, line 6). At each level, we generate new regions based on the input (2, line 9). The generated regions are then used to build the tree as we traverse it (2, lines 10-20). In addition, we introduce a special merge layer operation (1, lines 4-6) to identify and merge identical and bordering regions into a more compact form regardless of the region generator used. This makes it easier to introduce and use new generators, thus improving the pipeline's extensibility.

Subsequently, the algorithm produces as output a 2D layout (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.

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
1: function $ExecuteGenerators(root, generators)$
2:     $queue \leftarrow EmptyQueue$
3:     $queue.put(root)$
4:     while $queue$ is not empty do
5:         $node \leftarrow queue.get()$
6:         if $node.depth \geq generators.size$ or $node.gen = None$ then
7:             continue
8:         else
9:             $regions \leftarrow node.gen.generate(node.region)$
10:            for $region$ in $regions$ do
11:                $gen \leftarrow None$
12:                for $node.depth + 1 < i < len(generators)$ do
13:                    if $region.type \in generators[i]$ then
14:                        $gen \leftarrow generators[i][region.type]$
15:                        break
16:                    end if
17:                end for
18:                $childNode \leftarrow RegionNode(region, gen, node.depth + 1)$
19:                $node.addChild(childNode)$
20:                $queue.put(childNode)$
21:            end for
22:        end if
23:    end while
24:    return $root$
25: end function
End
```

3.3 Scene Creation

Scene creation is step # 2 in the 3DGENie pipeline (Figure 1). It transforms the 2D layout trees into detailed and realistic virtual scenes (Figure 6. left). The main goal is to populate virtual scenes with 3D assets following the generated layout tree structure.

Scene Construction - We adopt Nvidia Omniverse [30] 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, we use high-fidelity physics simulation [31] and virtual sensors within IsaacSim [30] and adopt Pixar's Universal Scene Description (USD) [32] to allow for seamless interchange across platforms. We use BMW Group's SORDI library Figure 4 [27], 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.

[5]

[5] Nvidia Omniverse [30] is a GPU-accelerated platform that provides realistic 3D rendering, physics simulation, and virtual sensor capabilities.

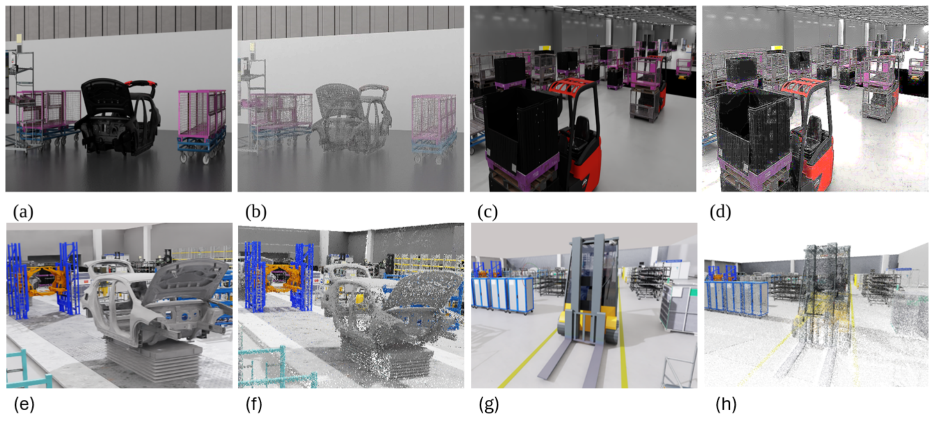


Fig. 6: (a, c, e, g) Reconstructed scene, (b, d, f, h) Generated point cloud.

Scene Randomization - Randomization is crucial in breaking patterns and biases that have a negative effect on machine learning models. By introducing randomization to an asset's placement and properties, we improve our synthetic data and support the development of robust machine learning models. It also helps simulate the unpredictable nature of real-world scenarios. In this context, we make use of IsaacSim Replicator [30] 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 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.

Sensor placement - We use the 2D layout generated in step #1 of 3DGENie to strategically position sensors within the scene. To achieve optimal placement, different strategies can be employed, using meta-heuristic or deterministic processes based on the user's needs. We investigated multiple sensor placement techniques including grid-based sampling, random placement, and greedy coverage algorithms, but chose the genetic algorithm as it experimentally produced superior coverage results with minimal fine-tuning. The algorithm is 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 regions.

Data Collection and Storage - 3DGENie generates the point cloud data and converts them into a suitable storage format. This includes not only raw sensor outputs, 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 like Semantic KITTI [11].

3.5 Requirements and Deploying 3DGENie

3DGENie requires a system with GPU capabilities to leverage Nvidia Omniverse's rendering and simulation features. The pipeline uses a microservices architecture based on Docker, it's easy to deploy and requires no external Omniverse installation. Users need access to 3D asset libraries (such as BMW's SORDI library or custom USD-formatted models) and should have basic familiarity with JSON formatting for creating input strategy files. The modular architecture of 3DGENie means that users can run the complete pipeline end-to-end or utilize individual components separately based on their needs.

4 Experimental Evaluation

4.1 Experimental Data

Real Data - We prepared a dataset of real 3D point cloud scans from car assembly lines. The scans were created using the NavVis VLX 2 [29] wearable laser scanning system, capable of generating colored and high-density point clouds. The scans were labelled manually by an industry expert, using a dedicated point cloud labeling tool that we developed in Omniverse. To maintain an acceptable input size and point density, we cropped each scene into smaller chunks, and then performed random down-sampling to 25,000 points on each. The final real dataset comprises around 4 million points and 5 classes (car, stillage, forklift, dolly, and background, cf. Table 2).

Synthetic data - We used 3DGENie to generate our synthetic point cloud dataset. To create the input strategies, we studied the layouts of different areas within multiple car manufacturing plants. Consequently, we generated 499 virtual scenes, each scene covered using 40 cameras configured to capture point clouds with a 512x512 resolution. We cleaned the data by removing point clouds that have few points or low percentage of labeled points. We then randomly selected a sub-sample of 56 scenes, which we found to have an acceptable training time of around 10 hours on average using an Nvidia A100 GPU. The resulting synthetic dataset comprises around 22 million points and covers 10 classes (including the 5 classes considered in the real dataset, cf. Table 2). We use 50% of the data for training and the other 50% as a test dataset.

The readers can refer to [33] for a more detailed description of the experimental data and evaluation results.

4.2 Semantic Segmentation Models

We selected three different semantic segmentation models, each known for its unique approach. **RandLA-Net [16]:** directly infers per-point semantics for

Table 2: Descriptions of real and synthetic point cloud datasets.

Class Name	# points in real dataset	# points in synthetic dataset
Background	3,474,905 (84.02%)	17,836,676 (81.746%)
Car	451,442 (10.75%)	348,676 (1.6%)
Stillage	98,772 (2.35%)	1,481,681 (6.79%)
Forklift	69,260 (1.65%)	157,985 (0.72%)
Dolly	51,887 (1.23%)	56,039 (0.26%)
Pallet	-	186,153 (0.85%)
Rack	-	1,120,530 (5.13%)
Small Load Carrier	-	52,597 (0.24%)
STR	-	3,038 (0.014%)
Cabinet	-	539,688 (2.47%)
Jack	-	39,117 (0.18%)

large-scale point clouds. Novel local feature aggregation module that progressively increases the receptive field for each 3D point, effectively preserving geometric details. **SparseConvNet [8]:** stands out for its use of sparse convolutional operations, enabling it to process sparse point clouds efficiently. **PVCNN [13]**: combines the efficiency of point-based processing with the structural advantages of volumetric convolutions. The PVCNN model is capable of achieving high accuracy at lower memory usage.

4.3 Experimental Results

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

Experiment 1: Mixing Real and Synthetic Data - In this set of experiments, we study how varying proportions of synthetic data impact semantic segmentation models. We created 10 training datasets with synthetic data increasing from 0% to 90% in 10% increments. To account for dataset size changes, we adjusted the number of training epochs. Results in Table 3 and Table 4 show that mixing synthetic with real data consistently improved performance across all models, which exhibited 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. We were able to improve the performance of all the models where most of them peaked between 40% to 60%.

Experiment 2: Train on Synthetic and Fine-tune on Real Data - We explored pretraining models on synthetic point clouds, followed by fine-tuning on real data. Models are first trained on the full synthetic dataset with varying epochs (starting from 20) until convergence, then fine-tuned on the full real dataset. All models outperformed their real-data-only baselines after pretraining on synthetic data. As shown in Table 6, performance increased with more synthetic training epochs, but declined beyond a point, suggesting overfitting to synthetic data reduced fine-tuning effectiveness.

Table 3: mAcc of the models across the dataset variants

Model	**Train 0**	**Train 10**	**Train 20**	**Train 30**	**Train 40**	**Train 50**	**Train 60**	**Train 70**	**Train 80**	**Train 90**
RandLaNet	0.608	0.628	0.630	0.658	0.660	0.653	0.668	0.672	0.680	**0.706**
SparseConvNet	0.606	0.664	0.629	0.625	0.629	0.638	**0.689**	0.672	0.654	0.545
PVCNN	0.665	0.689	0.681	0.684	**0.707**	0.687	0.699	0.678	0.668	0.629

Table 4: mIoU of the models across the dataset variants

Model	**Train 0**	**Train 10**	**Train 20**	**Train 30**	**Train 40**	**Train 50**	**Train 60**	**Train 70**	**Train 80**	**Train 90**
RandLaNet	0.553	0.576	0.582	0.603	0.611	0.612	0.623	0.623	0.628	**0.641**
SparseConvNet	0.546	0.596	0.567	0.5740	0.586	0.588	**0.613**	0.608	0.603	0.512
PVCNN	0.600	0.611	0.604	0.609	**0.623**	0.617	0.610	0.609	0.611	0.572

4.4 Autonomous Driving Scenario

Experimental Data Real Data
The experiment utilized the SemanticKITTI dataset [11], a comprehensive 3D point cloud dataset for autonomous driving collected with a Velodyne HDL-64E lidar sensor. The dataset, filmed in various environments in Karlsruhe, Germany, contains 22 sequences, with 21 for training/testing and one for benchmarking. It offers dense annotations across 28 semantic categories. For the experiment, 8000 point clouds were selected to align with the classes in a synthetic dataset. The data was split 80/20 into training and testing sets, resulting in a training dataset of 68,183,108 points and a test dataset of 1,865,540 points.

Synthetic Data
The synthetic dataset was created using 3DGENie to replicate real-world driving environments. A total of 625 virtual scenes were generated, featuring diverse environmental conditions. A Velodyne VLS-128 lidar sensor was used in simulations to mimic the density and distribution of real data, employing idealized ray tracing and normalized intensity processing to ensure high accuracy. That is a different type of sensor than the one used for the industrial dataset, and highlights 3DGENie flexible and modular nature. The final dataset contains over 741 million points spread across 6 semantic classes that match those in SemanticKITTI [11], facilitating realistic and precise data analysis.

Experiment: Mixing Real and Synthetic Data Based on the results of scenario 1, and seeing how data mixing gives better results than fine-tuning, we create 10 datasets. By using the training set from our processed SemanticKITTI data [11], and mixing it with an increasing amount of synthetic data we create the datasets shown in Table 7.

Table 5: mAcc of the models after fine-tuning.

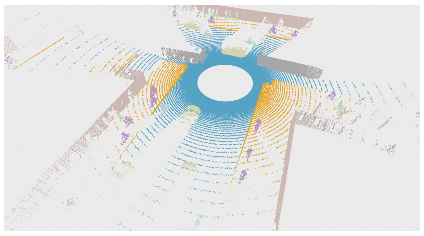
Model	**20 eps**	**40 eps**	**60 eps**	**80 eps**	**100 eps**	**Converge**
RandLaNet	0.6664	0.6604	**0.6706**	0.6704	0.655	0.609
Sparse ConvNet	0.6154	**0.6892**	0.6722	0.6682	0.6636	0.6652
PVCNN	0.6558	0.6342	0.6702	0.6466	0.63	**0.6898**

Table 6: mIoU of the models after fine-tuning.

Model	**20 eps**	**40 eps**	**60 eps**	**80 eps**	**100 eps**	**Converge**
RandLaNet	0.626	0.6214	0.6262	**0.6298**	0.6082	0.5616
Sparse ConvNet	0.5614	**0.6294**	0.6092	0.6245	0.6158	0.594
PVCNN	0.6024	0.584	**0.6234**	0.5784	0.578	0.6072

Results: In Experiment 1 of the autonomous driving scenario, synthetic data was incrementally added to real data, improving model performance as measured by mAccuracy and mIoU across three models. RandLaNet achieved its highest mIoU (0.719) and mAccuracy (0.811) with 40% synthetic data. SparseConvUNet also reached peak performance at the same ratio, with a mIoU of 0.640 and mAccuracy of 0.727. PVCNN showed best results at 20% synthetic data with a mIoU of 0.567 and mAccuracy of 0.675, but performance declined beyond this point. The study demonstrates the benefit of synthetic data, though at higher ratios (80–90%), performance either plateaued or diminished, likely due to the domain gap from an excess of synthetic data.

Comparison with existing data generation pipelines: We compared 3DGENie data generation tool with SynLIDAR, a pipeline built on Unreal Engine 4. Results show 3DGENie achieves better mean accuracy (mAcc), while SynLIDAR scores higher in Intersection over Union (IoU) metrics for specific


(a) 3DGENie synthetic sample

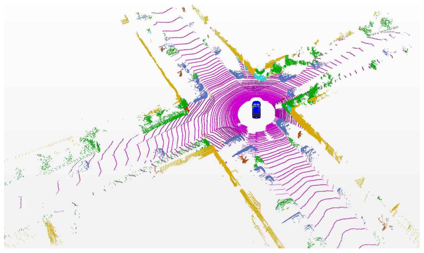

(b) SemanticKITTI real sample

Fig. 7: PC samples for autonomous driving.

Table 7: Datasets used for autonomous driving scenario.

Ratio	**# Real Points**	**# Synthetic Points**	**Total # Points**
0%	68,183,108	-	68,183,108
20%	68,183,108	20,139,194	88,322,302
40%	68,183,108	55,173,277	123,356,385
60%	68,183,108	123,351,787	191,534,895
80%	68,183,108	334,300,790	402,483,898
90%	68,183,108	741,049,322	809,232,430

Table 8: mAcc and mIoU of the models across the datasets variant for the autonomous driving scenario.

Ratio	**mAccuracy**			**mIoU**		
	RandLaNet	**PVCNN**	**SparseConvUNet**	**RandLaNet**	**PVCNN**	**SparseConvUNet**
0%	0.77	0.66	0.69	0.71	0.561	0.61
20%	0.809	0.675	0.706	0.707	**0.567**	0.625
40%	0.811	0.666	**0.727**	**0.719**	0.561	**0.640**
60%	0.815	0.669	0.695	0.714	0.521	0.613
80%	0.793	**0.684**	0.672	0.685	0.535	0.587
90%	**0.820**	0.660	0.699	0.691	0.529	0.627

networks. Despite close overall performance, 3DGENie offers a significant advantage in efficiency, as it allows rapid generation of new scenes in minutes and datasets in hours, whereas SynLIDAR requires manual, time-consuming scene construction. The findings demonstrate that 3DGENie maintains data quality while offering greater flexibility and quicker data generation capabilities than manual methods.

Table 9: mAcc across different model's for 3DGENie and SynLIDAR

	RandLaNet		**PVCNN**		**SparseConvUNet**	
	3DGENie	**SynLIDAR**	**3DGENie**	**SynLIDAR**	**3DGENie**	**SynLIDAR**
20.00%	0.80955	0.79200	0.67493	0.63618	0.70670	0.68247
40.00%	0.81134	0.80477	0.66615	0.65297	**0.72671**	0.69633
60.00%	**0.81524**	0.80859	0.66957	0.64972	0.69453	0.69505
80.00%	0.79336	0.81203	**0.68419**	0.64352	0.67289	0.70142
90.00%	0.81991	0.79747	0.65941	0.63738	0.69999	0.69609

5 Conclusion

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

Table 10: mIoU across different model's for 3DGENie and SynLIDAR

	RandLaNet		**PVCNN**		**SparseConvUNet**	
	3DGENie	**SynLIDAR**	**3DGENie**	**SynLIDAR**	**3DGENie**	**SynLIDAR**
20.00%	0.70736	0.7347	0.56732	0.55605	0.62514	0.61195
40.00%	0.71895	0.74360	0.56093	**0.57339**	**0.6404**	0.62330
60.00%	0.71463	**0.76069**	0.52139	0.54637	0.61341	0.62831
80.00%	0.68491	0.76152	0.53488	0.56154	0.60663	0.63613
90.00%	0.69108	0.74085	0.52901	0.55505	0.62754	0.62811

and applies 3D scene construction and asset randomization to produce scenes with 3D assets. We conducted various experiments to evaluate the performance of multiple computer vision models. Results consistently showed improved performance across all models. 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 [25], object recognition [23] and 6D pose estimation[26]. We are also building on 3DGENie to generate a range of synthetic data types, including new LiDAR simulations [15] [9], and RGB-D sensors for applications requiring color and depth information [6] (e.g., autonomous vehicle navigation [28], and virtual reality applications [22]). We also envision exploring the integration of recent Large Language Models (LLMs) or Vision-Language Models (VLMs) as layout generators, enabling natural language input to guide layout creation as future work.

References

1. Geiger A.. et al., .: Are we ready for autonomous driving? the kitti vision benchmark suite. In: (CVPR'12). pp. 3354–3361 (2012) 2
2. Wu Z.. et al., .: 3d shapenets: A deep representation for volumetric shapes (2014) 2, 3
3. Chang A.. et al., .: ShapeNet: An Information-Rich 3D Model Repository. CoRR abs/1512.03012 (2015) 2, 3
4. Hua B.S. . et al., .: Scenenn: A scene meshes dataset with annotations. In: Inter. Conf. on 3D Vision (3DV'16). pp. 92–101 (2016) 3
5. Dosovitskiy A.. et al., .: CARLA: An open urban driving simulator. In: Annual Conference on Robot Learning. pp. 1–16 (2017) 3
6. Dai A.. et al., .: Scannet: Richly-annotated 3d reconstructions of indoor scenes. In: Comp. Vis. and Patt. Recogn. (CVPR'17). pp. 2432–2443 (2017) 2, 3, 15
7. Hackel T.. et al., .: Semantic3d.net: A new large-scale point cloud classification benchmark. In: ISPRS Annals of the Photogrammetry. pp. 91–98 (2017) 2, 3
8. Graham B.. et al., .: 3d semantic segmentation with submanifold sparse convolutional networks. (CVPR'18) pp. 9224–9232 (2018) 11
9. Yue X.. et al., .: A lidar point cloud generator: from a virtual world to autonomous driving. Inter. Conf. on Multimedia Retrieval (ICMR'18) pp. 458–464 (2018) 3, 15
10. Wang F.. et al., .: Automatic generation of synthetic lidar point clouds for 3-d data analysis. IEEE Trans. on Instrum. and Meas. **68**(7), 2671–2673 (2019) 3

11. Behley J.. et al., .: A dataset for semantic segmentation of point cloud sequences. CoRR **abs/1904.01416** (2019) 2, 3, 10, 12
12. Mo K.. et al., .: 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 (2019) 2, 3
13. Liu Z.. et al., .: Point-voxel cnn for efficient 3d deep learning. Advances in Neural Information Processing Systems **32** (2019) 11
14. Uy M.A.. et al., .: 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 (2019) 3
15. Manivasagam S.. et al., .: Lidarsim: Realistic lidar simulation by leveraging the real world (2020) 3, 15
16. Hu Q.. et al., .: Randla-net: Efficient semantic segmentation of large-scale point clouds (2020) 10
17. Xiao A.. et al., .: Synlidar: Learning from synthetic lidar sequential point cloud for semantic segmentation. CoRR **abs/2107.05399** (2021) 1
18. Hu Q.. et al., .: 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 (2021) 2, 3
19. Karur K. et al., .: End-to-end synthetic lidar point cloud data generation and deep learning validation. Tech. rep., SAE Technical Paper (2022) 3
20. Chen M.. et al., .: Stpls3d: A large-scale synthetic and real aerial photogrammetry 3d point cloud dataset. In: British Mach. Vis. Conf. (BMVC'11). p. 429 (2022) 3
21. Schult J.. et al., .: Controlroom3d: Room generation using semantic proxy rooms (2023) 3
22. Lu Y.. et al., .: Machine learning for synthetic data generation: A review. CoRR abs/2302.04062 (2023) 15
23. Lu G.. et al., .: A novel method for improving point cloud accuracy in automotive radar object recognition. IEEE Access pp. 78538–78548 (2023) 15
24. Wu T.. et al., .: Omniobject3d: Large-vocabulary 3d object dataset for realistic perception, reconstruction and generation. Comp. Vis. and Patt. Recogn. (CVPR'23) pp. 803–814 (2023) 2, 3
25. Vu T.. et al., .: Scalable softgroup for 3d instance segmentation on point clouds. IEEE Trans. on Pattern Analysis and Machine Intell. **46**(4), 1981–1995 (2023) 15
26. Zou L.. et al., .: Learning geometric consistency and discrepancy for category-level 6d object pose estimation from point clouds. Patt. Recogn. **145**, 109896 (2024) 15
27. Nassif J.. et al., .: Synthetic Data: Revolutionizing the Industrial Metaverse. Springer Nature, 978-3-031-47560-3 (2024) 5, 8
28. Song Z.. et al., .: Synthetic datasets for autonomous driving: A survey. IEEE Transactions on Intelligent Vehicles **9**(1), 1847–1864 (Jan 2024) 15
29. NavVis: Navvis vlx 2. `https://www.navvis.com/vlx-2` (2024), access March'24 10
30. Nvidia: Isaac sim. `https://developer.nvidia.com/isaac-sim` (2024), access March'24 2, 8, 9
31. Nvidia: Physx sdk. `https://developer.nvidia.com/physx-sdk` (2024), access March'24 8
32. Nvidia: Pixar universal scene description. `https://developer.nvidia.com/usd` (2024), access March'24 2, 8
33. Yaghi, A., Tekli, J., Kamradt, M., Couturier, R.: 3dgenie: Synthetic point clouds for semantic segmentation in realistic virtual environments. Multimedia Tools and Applications pp. 1–33 (2025) 10