



UNIVERSIDADE D  
COIMBRA

Ruben Bidarra Bidault

SYNPHOREST v2.0  
SYNTHETIC PHOTOREALISTIC FOREST  
ENVIRONMENT GENERATION USING UNREAL  
ENGINE 5

VOLUME 1

Dissertation in Master's Degree in Electrical and Computer Engineering,  
supervised by Professor Doctor Paulo José Monteiro Peixoto and presented to  
the Department of Electrical and Computer Engineering of the Faculty of  
Science and Technology of the University of Coimbra.

July 2023





FCTUC FACULDADE DE CIÊNCIAS  
E TECNOLOGIA  
UNIVERSIDADE DE COIMBRA

# SynPhoRest v2.0 - Synthetic Photorealistic Forest Environment Generation using Unreal Engine 5

Ruben Bidarra Bidault

July 2023





# SynPhoRest v2.0 - Synthetic Photorealistic Forest Environment Generation using Unreal Engine 5

## **Supervisor:**

Prof. Paulo José Monteiro Peixoto

## **Jury:**

Prof. Hélder de Jesus Araújo

Prof. Paulo Jorge Carvalho Menezes

Prof. Paulo José Monteiro Peixoto

Dissertation submitted in partial fulfillment for the degree of Master of Science in  
Electrical and Computer Engineering.

July 2023



# Acknowledgements

Primeiro, quero agradecer à minha família por não entender absolutamente nada quando explicava o meu trabalho. Acredito que, ainda hoje, fingem perceber o que faço. E em especial, aos meus queridos pais, pois mais ninguém patrocinou estes longos anos de estudo. Asseguro-vos que isto tem saída.

À Só, a minha extraordinária namorada, que me acompanhou ao longo destes anos e que suportou os meus esgotamentos nervosos. Os meus discursos sem coerência levaram-na a aperfeiçoar a técnica "Sorrir e Acenar" e são os maiores responsáveis por ela frequentar um psicólogo regularmente.

Ao professor Paulo Peixoto pela disponibilidade, orientação e pelas reuniões semanais que adiou em cima da hora.

À minha psicóloga e à minha psiquiatra por me indicarem as drogas adequadas para terminar este projeto.

Aos meus amigos, em particular aos que também terminaram a tese este ano, sem eles não teria jogos de telemovel novos todas as semanas.

E finalmente, à EpicGames por disponibilizar um fórum específico ao Unreal Engine 5, sem ele nunca saberia que os meus problemas sem resolução também não têm resolução para os outros.

# Resumo

O área de aprendizagem computacional tem observado um crescimento exponencial devido aos rápidos avanços da tecnologia computacional e à crescente procura em diversas áreas. A cadência de desenvolvimento é extremamente elevada, tornando-se rapidamente um dos mercados mais lucrativos. A aprendizagem profunda, um ramo da aprendizagem computacional, utiliza redes neurais para aprender através de dados e tem obtido resultados excelentes. No entanto, estes resultados têm um preço: vastas quantidades de dados são necessários para treinar estes modelos. Os dados precisam ser recolhidos e anotados corretamente, uma tarefa manual tão monótona e extensa que apresenta uma elevada taxa de erro. Isto leva à escassez de dados, casualmente chamada de gargalo para a aprendizagem computacional. A aprendizagem profunda é amplamente aplicada à navegação autônoma, com maior foco no desenvolvimento de carros autônomos. A necessidade de grandes quantidades de dados e a disponibilidade reduzida de dados anotados resultam na falta de diversidade, onde a maioria dos dados disponíveis está associado a ambientes urbanos. A falta de dados relacionados com ambientes rurais e florestais, juntamente com as dificuldades associadas à sua etiquetagem, aumentam a necessidade de gerar dados sintéticos para incorporar no treino de modelos. Esta tese tem como objetivo preencher esta lacuna e facilitar a criação de dados focados em cenários florestais. O sistema proposto, construído usando o Unreal Engine 5, é capaz de gerar procedimentalmente ambientes florestais virtuais navegáveis e realizar a extração de dados sintéticos relevantes. Devido às capacidades do Unreal Engine 5, o sistema é capaz de alcançar um alto nível de fotorrealismo e simular com precisão áreas florestais. Através do uso de câmeras virtuais e da exploração das funcionalidades do Unreal Engine 5, o sistema é capaz de extrair dados no formato de imagens RGB, mapas de segmentação semântica e mapas de profundidade. Como um software pronto a publicar, este sistema oferece aos utilizadores a possibilidade de se movimentarem no ambiente gerado. O sistema oferece três métodos de geração de dados, duas perspectivas e a capacidade de ativar modos de visualiza-



ção de dados. Por fim, a avaliação do sistema resultou em mapas de segmentação semântica e mapas de profundidade com precisão ao nível do pixel. A avaliação de desempenho foi positiva, com o sistema a ser capaz de produzir uma centena de dados completos em pouco mais de uma hora.

**Palavras-Chave:** Dados Sintéticos; Geração Procedimental; Machine Learning; Percepção Robótica em Ambientes Florestais; Unreal Engine 5

# Abstract

The field of machine learning has experienced tremendous growth due to rapid advancements in computer technology and the increasing demand in the most various fields. The development cadence is extremely high and it is rapidly becoming one of the most profitable markets. Deep Learning, a branch of machine learning, uses a neural network to learn from data with excellent results. However, these results came at price of enormous datasets that are required for the training of these models. The data needs to be collected and correctly labelled, a manual task so monotonous and extensive that has an elevated error rate. This leads to shortage of data that is commonly called the bottleneck for Machine Learning. Deep learning is being widely applied to autonomous navigation, with self-driving cars leading the charge. The necessity of extensive amounts of data and the reduced availability of labelled datasets, translates to lack of diversity, where most available datasets are related to urban environments. The lack of data related to rural and forested environments and the difficulties associated with their labeling propel the need to generate synthetic data to incorporate in training. This thesis aims to bridge the gap and facilitate the creation of datasets focused on forest scenarios. The proposed system, built using Unreal Engine 5, is capable of procedurally generate navigable virtual forest environments and perform the extraction of accurate synthetic data. Due to Unreal Engine 5 capabilities, the system is able to achieve a high-level of photorealism and accurately simulate forested areas. Through the use of virtual cameras and exploiting Unreal Engine 5 functionalities, the system is able to extract data in the format of RGB images, semantic segmentation maps and depth maps. As a ready to ship software, this system offers users free mobility in the generated world. The system offers three methods of data generation, two perspectives and the ability to activate data view modes. Finally, the assessment of the system resulted pixel-perfect accuracy in both semantic segmentation maps and depth maps. The performance evaluation was positive, with the system capable of producing one hundred complete sets of data in just over one

hour.

**Keywords:** Synthetic Data; Procedural Generation; Machine Learning; Robotic Perception in Forest Environments; Unreal Engine 5



*“If you were a better programmer, you would program yourself a way out of this.”*

— Gilfoyle, *Silicon Valley*



# Contents

<b>Acknowledgements</b>	<b>ii</b>
<b>Resumo</b>	<b>iii</b>
<b>Abstract</b>	<b>v</b>
<b>List of Acronyms</b>	<b>xiii</b>
<b>List of Figures</b>	<b>xiv</b>
<b>List of Tables</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	3
1.1.1 Deep Learning applied to Forest Environments . . . . .	3
1.1.2 The Link Between Deep Learning and Data . . . . .	4
1.1.3 Dataset Construction . . . . .	4
1.2 Forest Data . . . . .	5
1.2.1 RGB Images . . . . .	6
1.2.2 Depth Maps . . . . .	7
1.3 Problem Synthesis . . . . .	7
1.4 SynPhoRest v2.0 - An Overview . . . . .	8
1.5 Main Contributions . . . . .	8
1.6 Document Outline . . . . .	9
<b>2 State of the Art</b>	<b>10</b>
2.1 The History of Synthetic Data . . . . .	10
2.2 Synthetic Data Generation . . . . .	11
2.2.1 Semantic Image Segmentation . . . . .	11

2.2.2	Data Generation from Fixed Virtual Environments . . . . .	12
2.2.3	Data Generation from Randomized Virtual Environments . . . . .	18
2.2.4	Data Generation from Procedural Virtual Environments . . . . .	19
2.2.5	Generation of Virtual Environments from Real Data . . . . .	22
<b>3</b>	<b>Methods</b>	<b>24</b>
3.1	Graphical Softwares . . . . .	24
3.2	Unreal Engine 5 . . . . .	25
3.2.1	Lumen . . . . .	25
3.2.2	Nanite . . . . .	26
3.2.3	Scripting . . . . .	27
3.2.4	Extension Capabilities . . . . .	27
3.3	Procedural Generation . . . . .	28
3.4	Extraction of Synthetic Data . . . . .	30
<b>4</b>	<b>Generation of Synthetic Forest</b>	
	<b>Environment</b>	<b>31</b>
4.1	Terrain . . . . .	31
4.1.1	Mesh Construction . . . . .	32
4.1.2	Object Positioning . . . . .	33
4.2	Path . . . . .	34
4.3	Materials . . . . .	35
4.4	Environment Assembling . . . . .	36
4.4.1	Chunk Management Module . . . . .	36
4.4.2	World Building Logic . . . . .	38
4.5	Graphical User Interface . . . . .	40
4.5.1	Main Menu . . . . .	40
4.5.2	Custom Mode Menu . . . . .	41
4.5.3	Pause Menu . . . . .	42
4.6	Extracted Data . . . . .	43
4.6.1	RGB Images . . . . .	43
4.6.2	Semantic Segmentation Maps . . . . .	44
4.6.3	Depth Maps . . . . .	45
4.6.4	Capture Methods . . . . .	47



<b>5</b>	<b>Results and Discussion</b>	<b>48</b>
5.1	Performance Analysis . . . . .	48
5.1.1	Generation Evaluation . . . . .	48
5.1.2	Capture Evaluation . . . . .	50
5.2	Extracted Data Analysis . . . . .	52
<b>6</b>	<b>Conclusion</b>	<b>57</b>
6.1	Future Work . . . . .	58
6.2	Personal Considerations . . . . .	59
<b>7</b>	<b>Bibliography</b>	<b>60</b>
<b>A</b>	<b>SynPhoRest v2.0</b>	

# List of Acronyms

<b>2D</b>	Two Dimensional
<b>3D</b>	Three Dimensional
<b>API</b>	Application Programming Interface
<b>CPU</b>	Central Processing Unit
<b>DL</b>	Deep Learning
<b>GPU</b>	Graphical Processing Unit
<b>GTA V</b>	Grand Theft Auto V
<b>GUI</b>	Graphical User Interface
<b>IoU</b>	Intersection over Union
<b>LOD</b>	Level of Detail
<b>ML</b>	Machine Learning
<b>RGB</b>	Red, Green and Blue
<b>RNG</b>	Random Number Generator
<b>ROS</b>	Robot Operating System
<b>UE</b>	Unreal Engine
<b>UE5</b>	Unreal Engine 5

# List of Figures

1.1	Methods of labelling: object detection (a), semantic segmentation (b), instance segmentation (c) and panoptic segmentation (d). Adapted from [14] and [15]. . . . .	6
1.2	RGB image (a) and the estimated depth map (b). Adapted from [16]. . . . .	7
2.1	Qualitative examples of labeling capability. RGB images with stereo depth map overlay (a), Cityscapes annotation (b), DeepLab [26] annotation (c), Adelaide [27] annotation (d) and Dilated10 [28] annotation (e). From [25]. . . . .	12
2.2	RGB image (a) and equivalent raw semantic segmentation map (b) obtained from GTA V virtual environment. From [29]. . . . .	13
2.3	RGB image (a) and equivalent grouped semantic segmentation map (b) obtained from GTA V virtual environment. From [29]. . . . .	14
2.4	Effect of label propagation, percentage of pre-annotated area per capture at the beginning of manual labelling of each capture. Captures arranged by processing order (a) and captures arranged by pre-annotated area (b). From [29]. . . . .	14
2.5	Differences between the final rendered scene (a) and the simplified collision meshes (b). From [34]. . . . .	16
2.6	Example of the SynthCity dataset displaying RGB values (a) and equivalent class labels (b). From [36]. . . . .	18
2.7	Examples of road networks generated using CruzWay. Complete network (a) and intersection (b). From [39]. . . . .	19
2.8	Input image and corresponding segmentation prediction by DeepLab[26]. Source of the image Syncapes (a), Richter[29] (b) and Synthia[42] (c). From [43] . . . . .	20
2.9	Examples of the environment generation. From [44]. . . . .	21

2.10	Various data types extracted by the system. RGB image (a), segmentation map (b), depth map (c) and point cloud (d). From [19]. . . . .	22
2.11	Real-world RGB image captured by the vehicle (a), autonomous annotated reconstruction (b) and virtual reconstructed scenario (c). Adapted from [49].	22
2.12	Seven of the eight RGB images captured around the vehicle and the recreation, in 3D vector space, of the labelled environment. Adapted from [49]. . . . .	23
3.1	Environment rendered with Unreal Engine 5 (a) and Unity Engine (b). Captured from [56]. . . . .	25
3.2	Comparison between Lumen (a) and lightmaps (b). Adapted from [57]. . . .	26
3.3	Model, with Nanite active, at different distances. Adapted from [58]. . . . .	26
3.4	Simplex noise generated through layering. Number of layers applied: one (a), two (b), three (c), four (d), five (e), six (f), seven (g) and eight (h). Generated using [68]. . . . .	29
3.5	Influence of one (a) and ten (b) layers Simplex Noise in a terrain mesh. . . .	29
3.6	Same scene with (a) and without (b) film grain. . . . .	30
4.1	Diagram of the object placement logic, seen from above. Green circles represent the admissible area of placement and the red circle represent the random place attributed to the object. . . . .	34
4.2	Simplified version of the path building process. Projection of the spline downwards (a), collision points obtained through the projections (b) and mesh built with the collision points (c). . . . .	35
4.3	Materials built for the terrain component (a) and path component (b). . . .	36
4.4	Visual representation of the chunk manager logic. . . . .	37
4.5	Visual representation of placement verification logic. Black circles represent already placed objects and the orange rectangle represents the path. The green and red circles represent the to be placed object and the validity of its location. Correct placement (a), adjusted placement (b) and no placement (c). .	39
4.6	Environment with the three definitive illumination conditions, day (a), dusk (b) and night (c). . . . .	40
4.7	SynPhoRest v2.0 main menu. . . . .	41
4.8	SynPhoRest v2.0 custom mode menu. . . . .	42
4.9	SynPhoRest v2.0 pause menu. . . . .	43

4.10	RGB images captured from the ground perspective (a) and the aerial perspective (b). . . . .	44
4.11	Semantic segmentation map captured from the ground perspective (a) and the aerial perspective (b). . . . .	45
4.12	Depth map captured from the ground perspective (a) and the aerial perspective (b). . . . .	46
4.13	Naming convention of each type of capture performed. . . . .	47
5.1	Comparison between common stereo depth maps (a), "Aslam" algorithm (b) and the depth maps generated by SynPhoRest v2.0 (c). Adapted from [78]. .	53
5.2	Example of semantic segmentation maps precision verification. Layering of the RGB and semantic segmentation map images, with opacity at 100% and 25%, respectively. . . . .	54
5.3	Example of semantic segmentation maps precision verification. Layering of the RGB and semantic segmentation map images, with opacity at 100% and 25%, respectively. . . . .	55
5.4	Example of semantic segmentation maps precision verification. Layering of the RGB and semantic segmentation map images, with opacity at 100% and 25%, respectively. . . . .	55
5.5	Example of semantic segmentation maps precision verification. Layering of the RGB and semantic segmentation map images, with opacity at 100% and 25%, respectively. . . . .	56
6.1	Generation error detected during testing. . . . .	59

# List of Tables

2.1	Percentage of successfully completed episodes in each condition for the three autonomous driving systems tested. From [35]. . . . .	17
2.2	Validation on synthetic data for reference version of DeepLab[26]. Adapted from [43]. . . . .	20
5.1	Values of vegetation density and respective generation times in seconds. . . .	49
5.2	Values of maximum terrain height and respective generation times in seconds.	49
5.3	Illumination conditions and respective generation times in seconds. . . . .	49
5.4	Generation with and without path inclusion and respective generation times in seconds. . . . .	50
5.5	Values of vegetation density and respective data collection times in seconds.	50
5.6	Output resolution options and respective data collection times in seconds. . .	51
5.7	Number of captures and respective data collection times. . . . .	51
5.8	Average capture time to extract one frame of a specific data type. . . . .	51
5.9	Average switch time from one data type to another. . . . .	52



# 1 Introduction

Over the past few years, the field of Machine Learning (ML) has been revolutionized thanks to the technological evolution of computers, which have become more powerful and faster in terms of computation. ML is an ever-growing study area and can be applied to a vast number of tasks, ranging from the trivial recommendation system to the difficult fully-automated surgical procedure. In fact, ML is ingrained in our day-to-day activities, being used in spam filtering, route planning, online targeted advertising and much more. With so many applications and its increasing demand, ML is becoming one of the biggest markets globally. A study conducted by Fortune Business Insights estimates that the market was valued at 21.17 billion dollars in 2022 and is expected to grow to 209.91 billion dollars by 2029 [1]. Additionally, it was estimated by The Wall Street Journal that the total amount invested in ML research, in 2021, was 57 billion dollars [2].

ML is an umbrella term for the spectrum of algorithms that are developed using the computer's capacity of learning from data. Deep Learning (DL) is a subclass of ML that is based on the inner-workings of the human brain. Specifically, it takes advantage of layered structures composed of interconnected neurons that make up the neural network. These neurons are able to take the data that is given as an input and produce a more weighted output, allowing a learning process that is more hands-off for the programmer at the expense of larger datasets and bigger computation times. Although DL is a more taxing process, it is recognized as a superior technique when compared to the conventional algorithms used in many fields. Hence, the conventional algorithms are being replaced by more precise, reliable and efficient DL algorithms. In fact, as ascertained by N. O'Mahony et al. [3], most precursor algorithms employed in computer vision have become irrelevant due to the implementation of DL based algorithms.

The development of a DL algorithm requires a substantial amount of data, directly linked to the problem at hand. This data has to be labelled to transmit the correct information to guide the learning process of the algorithm. Moreover, task complexity further increases the



amount of demanded data. To illustrate data necessity, when working with image segmentation it is common practice to use a thousand images per class [4]. Training an algorithm to detect ten different classes involves a minimum of ten thousand images, where all images are required to be labeled. The labelling process consists in masking each class present in the image, with a colored polygon, following the designated color/class code. This task is highly demanding and has the potential to halt the progress of the investigation by diverting all attention and resources.

Furthermore, in the majority of applications, obtaining a pre-labelled dataset is not an option, requiring the collection of raw data. The time-consuming process of labelling the data, coupled with the challenging task of gathering raw data, can often result in the premature termination of the research due to time-constraints or even demoralization.

Autonomous navigation is one of the areas in which ML and DL are heavily applied with constant improvements. Autonomous driving applications serve as a focal point for much of the work conducted in this field. The existence of many labelled datasets and high demand of such applications are the catalyst to the observed boost in self-driving cars development. This leaves perceiving and navigating other environments as an afterthought. However, the navigation of forested environments is an area that has become increasingly studied, with aims of developing and deploying fully automated robots to perform a range of key tasks to preserve and monitor the natural environments, as well as perform search and rescue operations.

This thesis targets the development of a system that can extract synthetic data from procedurally generated worlds for the purpose of training DL algorithms for forest applications. Currently, synthetic data cannot fully replace real-world data, however it can offer a lot of benefits. Therefore studying and developing this type of data is a crucial first step in addressing the issue of data scarcity.

This Chapter will first outline the motivation for the work that was created, stating some of the DL applications intended for forest areas, followed by the identification of pertinent data types. After that, a detailed explanation of the problem and a summary of the suggested system are provided, ending with a list of the principal benefits the developed system offers to the scientific community.

## 1.1 Motivation

At the moment, the Forest Guard is the solely responsible for maintaining and monitoring the forest environment. Even without considering the decrease in available personnel, the Forest Guard faces a significant disadvantage as they are tasked with monitoring over four billion hectares of forest [5] solely on foot. The deployment of drones eased the monitoring process, but the crucial tasks related to the preservation are still performed manually. These laborious manual tasks and the shortage in personnel increase the risk of wildfires. In 2022, Portugal had a total burned area of 1100.07  $km^2$ , of which 40.1% were forested areas [6], demonstrating a pressing need for an operational system for patrolling and maintenance of forested environments.

### 1.1.1 Deep Learning applied to Forest Environments

The development and training of an advanced robotic system to aid in the preservation and maintenance of forest requires an incredibly specific dataset. However, when searching for datasets dedicated to forested environments, it becomes apparent that the available options are limited, with existing ones often containing insufficient data and redundant information.

The existing datasets are all derivations of the Red, Green and Blue (RGB) image dataset developed by A. Giusti et al. [7]. This dataset was put collected as a result of research towards creating a controller for a quadcopter's autonomous navigation during rescue operations. The intention was to make it possible for the drone to travel along forest trails and with the use of sensors for direction detection, each image in the dataset was labelled with the precise path the quadcopter should take, performing a form of automated labelling. However, the model developed using this dataset is only able to predict which way the robot should travel could not infer information about the objects that are in its immediate vicinity.

Using Giusti's dataset in conjunction with newly collected data, L. Lind [8] investigated the use of ML in autonomous ground vehicle navigation in a forest environment. The author came to the conclusion that, despite the attained autonomy of 75% being far from reliable, multiple improvements that can be made to the model that would allow reliance on autonomous forest navigation.

Summing up the information presented, the problem is clearly the absence of data related to forested environments. The redundancy of the existing datasets and the simplistic labeling

offered further increase the necessity of new data. Additionally, A. Munappy et al. [9] stated another difficulty related to ML for forest deployment, the disparities in real-world forests and the reduced diversity of the available data.

### 1.1.2 The Link Between Deep Learning and Data

DL algorithms are known for being highly dependent on data [9]. The neural network structure present in these algorithms grants them a higher degree of independence compared to traditional ML algorithms. However, this advantage comes at the cost of requiring larger datasets and longer training times to achieve better results.

Although based on the structure of the human brain, DL models lack the ability to generalize concepts and find similarities between similar objects. These algorithms heavily rely on being presented with numerous variations of an object to develop a sufficiently general model, which leads to larger and more diverse datasets, essential to achieve the necessary and correct training.

As previously mentioned, the complexity of a problem directly influences the amount of labeled data required. In the case of autonomous navigation models, the need for comprehensive coverage of all possible scenarios and eventualities demands an extensive data collection. Furthermore, when considering autonomous navigation in non-urban areas, it becomes crucial to account for environmental disparities across multiple deployments. These disparities add another layer of complexity that must be addressed for effective autonomous navigation.

### 1.1.3 Dataset Construction

Collecting data in any context is a laborious and time-consuming task, but when it comes to data collection in a forest scenario, the challenges further escalate. In the forest environment, data collection demands extensive fieldwork, requiring hours of dedicated effort. It involves the use of sensors powered by mobile electrical sources and requires the navigation of non-optimal terrains. The demanding nature of this type of data collection is well-documented by A. Giusti et al. [7] and L. Lind [8].

Ensuring comprehensive data coverage, including all necessary cases for effective model generalization is crucial. However, achieving this goal is not always straightforward, leading to inadequate data coverage and potential issues such as overfitting or subpar performance. In such cases, data may need to be recollected to address these challenges.

Furthermore, evaluating the coverage of already collected data poses its own difficulties. Assessing the extent to which the collected data adequately represents the target domain is a complex problem to tackle. Additionally, models trained on datasets with imbalanced class representation can exhibit poor performance, as reported by Q. Hu et al. [10].

The forest environment presents even more challenges in achieving balanced coverage across all classes in the collected data. Successfully addressing these challenges becomes particularly demanding in a forest setting. The distribution values of classes such as trees and rocks vary significantly, inducing dataset imbalance problems that translate in poor performances.

As was previously stated, the use of DL algorithms is heavily intertwined with larger datasets. These datasets are required to be labelled with meaningful descriptions of the data. It is crucial for the performance of a DL model that the training is executed with precise data annotation.

Constructing a dataset requires a large amount of data that, usually, has to be labelled by hand. There are some tools that facilitate the process, such as V7 [11] and others enumerated by SyncedReview [12]. But even with such tools, the labelling process is prone to error since it is a repetitive endeavour performed by an increasingly exhausted human operator. All this variables lead to a mislabelled dataset and consequently to non-optimal output. To prove the frequency of this occurrence, an analysis made by T. Ganter [13] to Google's Open Images dataset, a thoroughly researched and improved dataset, resulted in the conclusion that 36% of the model's errors where related to mislabelled data.

Tools for dataset labelling are still being researched and developed, with the goal of improving the effectiveness and precision of annotation. However, the annotation process of images captured in forest environments, comparatively to urban scenarios, adds an additional challenge related to the unstructured nature of each scene. The difficulty of this task increases significantly when performing the annotation of point cloud data.

## 1.2 Forest Data

Autonomous navigation models use a wide variety of sensors' data in the training process and each form of data contributes differently to the algorithm's learning. Finding the relevant data types is the first step in creating a reliable synthetic data generator. The many data types that best characterise forest areas will be address in this Section.

### 1.2.1 RGB Images

The best and most common method of data collection for environmental analysis is through RGB images. Each primary colour in an RGB image is represented by a separate matrix, with each matrix value denoting a single pixel and containing the colour value of the corresponding primary colour. Various methods are used to obtain useful information from RGB image classification. These methods include object detection, semantic segmentation, instance segmentation and panoptic segmentation.

Object detection involves predicting the positions of objects within the image and annotating them with bounding boxes. This technique allows for the identification and localization of objects. Semantic segmentation focuses on masking each object present in the scene with the appropriate class color. It assigns a specific class label to each pixel, creating a comprehensive understanding of the image's composition. Instance segmentation is an extension of object detection and semantic segmentation. It differentiates between each instance of an object class by masking them with distinct colors, while disregarding other classes. Panoptic segmentation represents the most advanced and intricate method. It unifies the processes of semantic and instance segmentation, generating unique color masks for every object of every class.

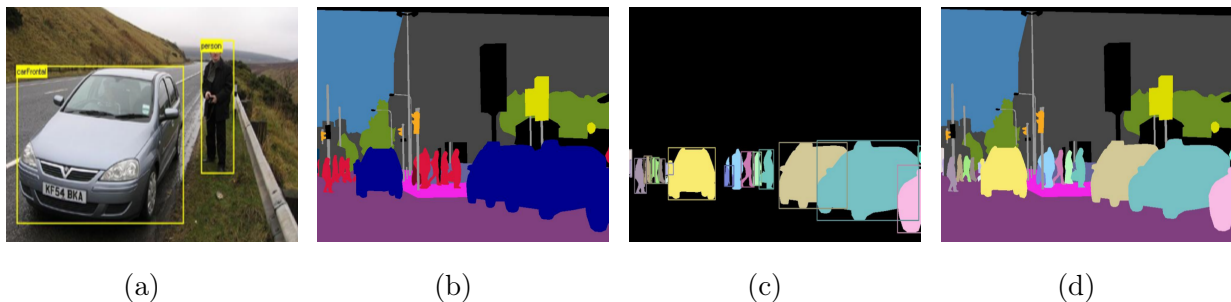


Figure 1.1: Methods of labelling: object detection (a), semantic segmentation (b), instance segmentation (c) and panoptic segmentation (d). Adapted from [14] and [15].

Considering the specific nature of the targeted environments, the need for highly detailed segmentation techniques is not essential. Therefore, the most suitable method for forest environment labeling is the semantic segmentation. It provides a sufficient level of understanding and differentiation for the objects present, aligning well with the requirements of the analysis.

## 1.2.2 Depth Maps

Depth maps are images represented by only one matrix, where each pixel represents the distance between the camera and the surface of objects within the scene. Depth maps are commonly produced using captures performed with depth cameras or stereo matching, where two RGB images with slightly different viewpoints of the same scene are used to estimate the depth.

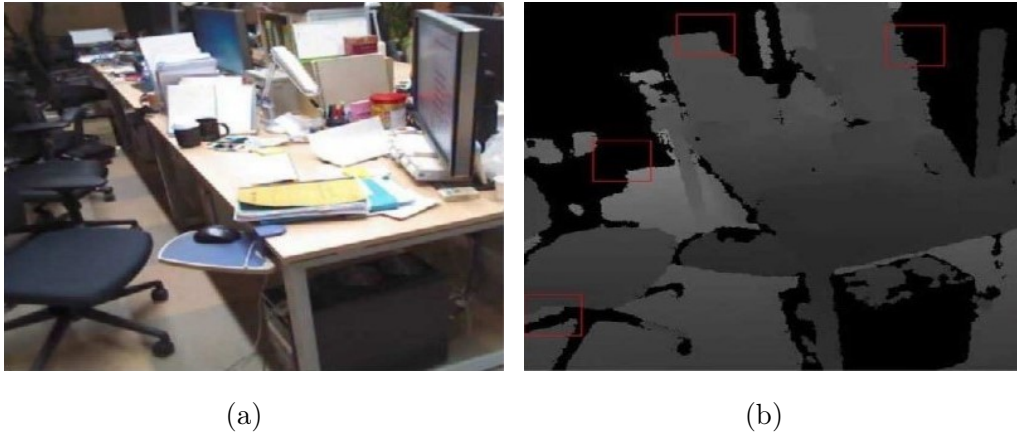


Figure 1.2: RGB image (a) and the estimated depth map (b). Adapted from [16].

The combination of depth maps with RGB images allows neural networks to infer depth information solely from RGB inputs. The integration of depth maps becomes crucial in training DL models for forest deployment, as it enhances the models' understanding of the three-dimensional environment.

## 1.3 Problem Synthesis

Undoubtedly, data collecting is a difficult endeavour, but it is nothing compared to the laborious labelling procedure, which is plagued by high mislabel rates. The situation worsens when large labeled datasets are required and the inclusion of several data types only aggravates the issue. The unfortunate lack of publicly available datasets and the fact that they typically only cover one data type severely limits the development of DL applications tailored for forest environments. Additionally, the wild and unorganised nature of the forest environment by itself presents special challenges, where data collection and labelling are made more difficult by a large amount of factors.

There is an urgent need to invest in the creation of datasets specifically made for forest applications in order to get over these obstacles and enable the development of dependable

DL models. Such investments will provide the groundwork for improvements in active DL researches and open the door for game-changing innovations in technology related to forests.

## 1.4 SynPhoRest v2.0 - An Overview

Leveraging the documented success of synthetic data inclusion in training of ML models for autonomous navigation by H.M. Jeon et al. [17] and L. Zherdeva et al. [18], this thesis proposes a system capable of generating accurate and labelled synthetic data captured in a virtual, procedurally generated and fully navigable forest environment. The system developed was fully built using the state-of-the-art game engine Unreal Engine 5 (UE5) and is ready for public release. As stated, the system is capable of generating synthetic data in the form of RGB images, semantic segmentation maps and depth maps of each capture performed. The world generation process presented is able to generate more than 4 billion unique maps and this number can be further increased through personalization of the available generation parameters. The generated data can be extracted through manual or automated methods and easily accessible in the launcher directory.

SynPhoRest v2.0 was developed as a new approach to the work performed by R. Nunes et al. [19] and his development of SynPhoRest v1.0.

## 1.5 Main Contributions

SynPhoRest v2.0 provides ML and DL research teams a tool to generate large and accurate synthetic datasets for training and validation of models developed for forest deployment. The generated data focuses on addressing the existing deficit in available data regarding this type of environment and delivers the semantic segmentation and depth values associated with each RGB capture. Moreover, the access to personalized world generation promotes the versatility of the extracted data. The system offers various data capture methods, including autonomous extraction for hands-off data collection and custom methods that encourages exploration and the capture of specific data.

In sum, SynPhoRest v2.0 intends to redirect attention to the creation of algorithms and models by providing research teams with data that is ready for use, removing or at the very least reducing the need for data collecting.

## 1.6 Document Outline

The present document is organized in six Chapters, beginning with the current Chapter that introduces the work developed and contextualize it, identifying the addressed problem, the proposed solution and enumerated the benefits of the project.

Chapter 2 focus on analysing documented solutions and systems relevant within the targeted filed of research, followed by Chapter 3 where the study and selection of available tools and methods is discussed.

In Chapter 4 a detailed description of the system development is presented and an analysis of the fully operational system, examining the performance and evaluating qualitatively and quantitatively the data produced, is provided in Chapter 5.

Finally, Chapter 6 presents some conclusions related to the development and evaluation of the system, in addition to the identification of possible future research directions.



## 2 State of the Art

The work developed for this thesis focuses primarily on the generation and use of synthetic data. This Chapter provides an overview of the documented work created in the field as well as a brief history of the subject. The analysis will focus on the positive aspects of each project while also pointing out its limitations and highlighting any potential weaknesses that need to be improved.

### 2.1 The History of Synthetic Data

The first documented use of synthetic data goes back to the 1940s, amidst World War II, as part of the Manhattan Project, where S. Ulam and J. von Neumann [20] developed the Monte Carlo Simulation. This rudimentary approach used the generation of random sampling, according to an estimated range of values, to determine the behaviors of complex systems or processes. This method allowed them to determine an approximation of the chain reaction of highly enriched uranium, i.e, the behavior of an atomic explosion. In the 1970s, the revolution of computer graphics and the beginning of the field of computer vision, propelled the generation of synthetic data to new heights, creating the first renderings of realistic images and scenes. Thanks to these new methods, the next few decades can be characterized as a jumping pad for the application of synthetic data. In fact, the field of robotics started using synthetic data in 1980s, enabling training, simulation and testing of algorithms in controlled environments [21].

In the following decade, D. Rubin [22] proposed the construction and publication of an anonymous database composed exclusively of artificial data based on the information collected by the U.S. Census. This database allowed the amplification of the existing smaller datasets and an increase in security related to personal information.

The coming of the new millennium marked the rebirth and expansion of machine learning, and the field began to branch in the most various directions. However, novelty came with

its drawbacks and all branches determined that the most crippling issue was the scarcity of data. To mitigate this newfound adversity, synthetic data generation became the most efficient mean of data augmentation. This method enables the use of limited datasets, boosting the available data in a convenient way and granting improvements in the models' performances. Thereafter, synthetic data generation became a hot topic and many studies aimed to not only improve its quality but also the dimension and accessibility of data. In the decade of 2010, with the increased popularity of generative adversarial networks and variational autoencoders, both models were used in data generation tools and the results rivaled real-world data [23].

Currently, synthetic data generation continues to be an active area of research. The field of machine learning is still expanding, and as new problems and applications emerge, the necessity of new and diverse data increases.

## 2.2 Synthetic Data Generation

To take advantage of the benefits of synthetic data, first one needs to find a way to generate it. Considering the generation of visual data aimed for autonomous navigation applications, there are multiple documented systems capable of generating datasets from virtual environments. The amount of data that can be generated is, however, limited because the majority are developed around pre-made scenarios. Even so, the methods used to capture and process data are transversal to the field, and the amalgamation of knowledge presented by all the cases studied provide a useful tool for referencing and comparison.

### 2.2.1 Semantic Image Segmentation

A visual dataset is composed by RGB images and their equivalent semantic segmentation maps. Semantic segmentation maps are generated by assigning a specific value to each pixel, indicating the class or object present in the associated pixel in the RGB image. This task is usually performed by hand, and in order to achieve a pixel-perfect mapping, an incredible amount of effort and attention is required. To illustrate the amount of labor inherent to the construction of a dataset, the CamVid Database, created by G.J. Brostow et al. [24], contains seven hundred and one photos that have been manually labeled for thirty-two different classes. The labelling process was performed by a small group of people and required a total amount of two hundred and thirty hours to complete.

Similarly, M. Cordts et al. [25] constructed the Cityscapes dataset, specifically designed to the challenges of semantic segmentation of an urban scene, by incorporating complex real-world urban environments. The dataset encompasses a collection of stereo video sequences captured on the streets of 50 different cities. This collection amounts to a total of five thousand fully annotated frames with pixel-level precision and an additional twenty thousand frames with coarse labelling. Amidst the various experiments performed to evaluate the Cityscapes data, one in particular is extremely relevant, the evaluation of labeling capabilities of various algorithms. The authors of Cityscapes compared, qualitatively, the performance of three state-of-the-art classification algorithms and obtained mostly positive results that can be observed in Figure 2.1.

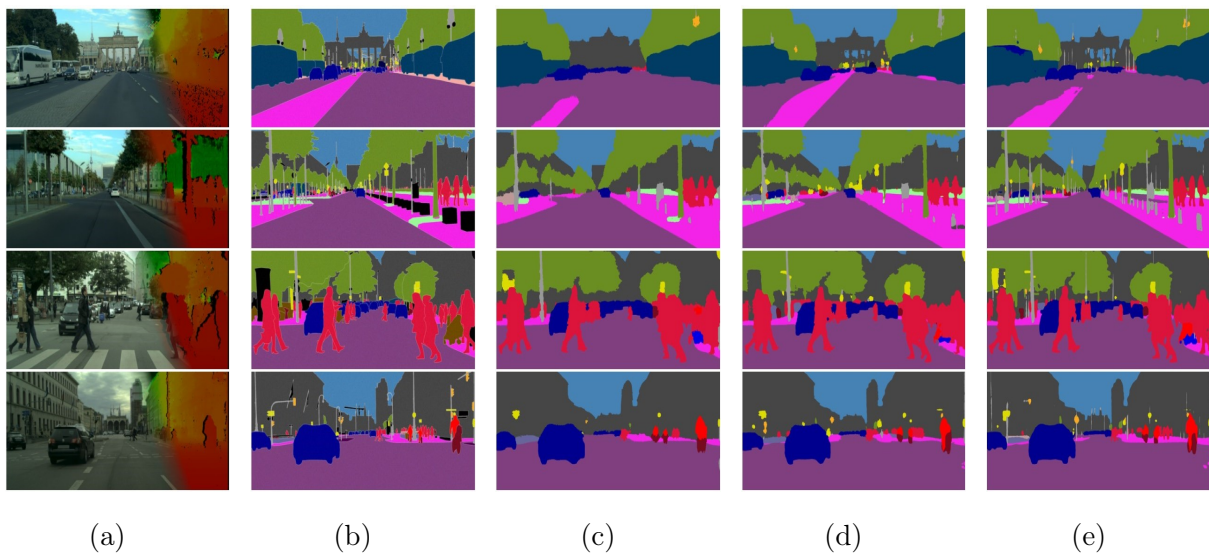


Figure 2.1: Qualitative examples of labeling capability. RGB images with stereo depth map overlay (a), Cityscapes annotation (b), DeepLab [26] annotation (c), Adelaide [27] annotation (d) and Dilated10 [28] annotation (e). From [25].

## 2.2.2 Data Generation from Fixed Virtual Environments

### Synthetic Semantic Segmentation Maps

Inspired by the power of computer vision and the exponential evolution of Three Dimensional (3D) environments, S.R.Richter et al. [29] developed a system capable of generating images and semantic segmentation maps, captured from a virtual pre-built scenario. To generate data that rivaled real-world imagery, the authors resorted to a market that is not often associated with research - the gaming industry, by utilizing and navigating the detail rich

environments of AAA<sup>1</sup> open world video-games. Renowned for its immersive open world, the game Grand Theft Auto V (GTA V), published in 2013, was chosen to be the source of the generated data. Being a full fledged AAA game none of the internal code is available, to acquire the desired data Richter’s team employed a technique called detouring. Detouring enabled the interception of data transmitted from the game’s logic to the Graphical Processing Unit (GPU), and extracting all graphics Application Programming Interface (API) calls. This method allowed for the recovery of information related to pixel values, used meshes and respective textures and shaders, automatically producing a set of RGB image and semantic segmentation map for each capture, as presented in Figure 2.2.



Figure 2.2: RGB image (a) and equivalent raw semantic segmentation map (b) obtained from GTA V virtual environment. From [29].

However, the semantic segmentation map creates a new division for every color change present. Two additional task need to be performed, the grouping of segments that represent the same classes and the labelling attribution. The final result can be observed in Figure 2.3. To expedite the labeling process, the label attributed to one object was propagated to all objects that shared either mesh, texture or shader. This meant that every fully-labelled image would reduce the labour required for the next one, following the plotted behavior present in Figure 2.4.

All this implementations allowed S.R. Richter et al. to create a dataset composed of twenty-five thousand images and their respective semantic segmentation maps with pixel-level precision. The collection and processing of this dataset amounted to a total of forty-

---

<sup>1</sup>AAA (triple-A). Video-games with a high quality and production value, developed by established and reputable game development studios. These games frequently have high production standards, cutting-edge graphics, intricate gameplay mechanics, engaging storytelling, and a wide variety of content. This term was initially used by the industry as a rating system but later evolved into a descriptor for games and their associated expectations.

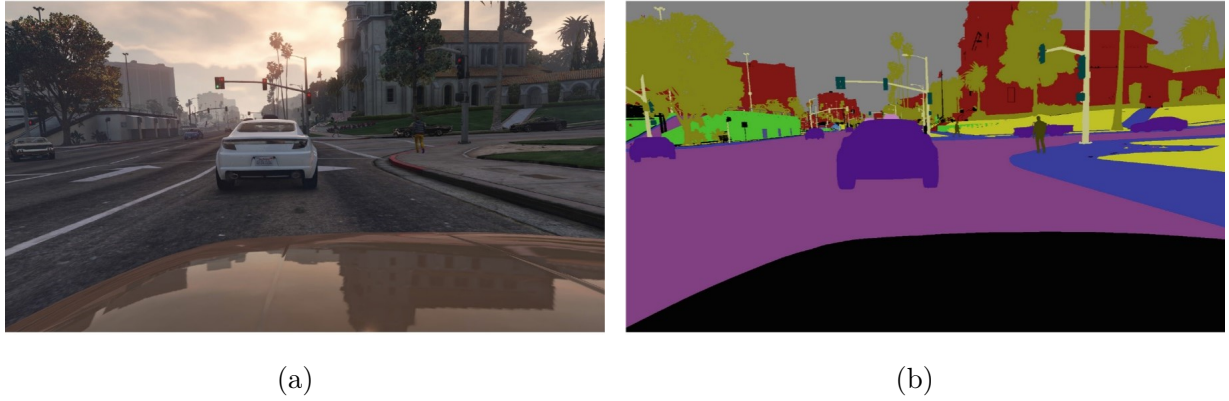


Figure 2.3: RGB image (a) and equivalent grouped semantic segmentation map (b) obtained from GTA V virtual environment. From [29].

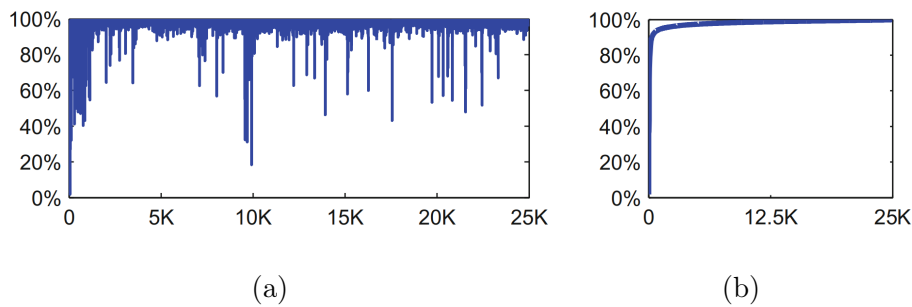


Figure 2.4: Effect of label propagation, percentage of pre-annotated area per capture at the beginning of manual labelling of each capture. Captures arranged by processing order (a) and captures arranged by pre-annotated area (b). From [29].

nine hours. To validate the benefits of synthetic data, this team compared the results of two models with the same algorithm but trained with different datasets. One used the complete CamVid dataset [24], the second used a combination of data generated from GTA V and data from the CamVid dataset, following a ratio of 2:1. The model trained with the mixed dataset outscored the CamVid model by 2.8 percentage points, proving the effectiveness of synthetic data in model performance, while reducing the time of collection and labelling.

### Synthetic Point Clouds

Also taking advantage of GTA V powerful graphics, X. Yue et al. [30] developed a system capable of generating annotated point clouds. By utilizing the DeepGTAV [31] plugin, a game mod that transforms the game into a vision-based self-driving car research environment, the team was able to deploy an in-game self-driving car, controlled by the game’s artificial intelligence, complete with a virtual LIDAR scanner and aRGB camera. Both these components were overlapping and positioned on top of the car, in order to extract

color coordinated point clouds, with each color representing a different class of object.

To verify and evaluate the quality of the extracted data, the authors used the DL model SqueezeSeg [32] trained with a modified version of the KITTI dataset [33], where the bounding boxes provided by the dataset were converted to labels for the contained points. Four tests were performed and used the Intersection over Union (IoU) metric for comparison. LIDAR scanners have an intensity value associated with each point. As reported by the authors, this value was too difficult to generate in the virtual environment and, as such, it was discarded. The datasets used for training and the results obtained were as follows:

- KITTI dataset(including intensity) resulted in an IoU of 64.6%
- KITTI dataset(excluding intensity) resulted in an IoU of 57.1%
- Fully synthetic dataset resulted in an IoU of 29.0%
- Combination of KITTI dataset(excluding intensity) and synthetic data resulted in an IoU of 66.0%

The results demonstrate that the synthetic data collected alone was not a viable option as a training dataset. However, the combination of both types of data enabled an increased performance relative to the test performed only with the KITTI dataset. This also confirmed that the intensity value captured by the LIDAR sensor can be discarded. It's important to note that the authors did not specify the ratio of data contained in the combined dataset. The absence of this information makes it impossible to assess the effectiveness and quality of the data that this system has extracted. Additionally, an important topic was omitted in this experiment, the noise generated by the sensor captures. In a physical deployment of a LIDAR sensor, all data gathered has some type of noise associated. To realistically simulate this data collection, some sort of noise function needs to be implemented.

Using the same virtual environment and the DeepGTAV plugin [31], B. Hurl et al. [34] noted that the virtual LIDAR sensor could only perform readings at a small portion of the original counterpart's range. Furthermore, the simulated sensor relies on the in-game collision volumes associated with each object. This volumes, in order to increase performance, are simplified and reduced to the basic geometric shapes, as seen in Figure 2.5.

The authors opted to discontinue the in-game LIDAR simulation and instead utilized the GPU's depth buffer data, used on the rendering of the scenes, which bypassed the two aforementioned limitations. This information provided a two-and-a-half-dimensional representation of the scenario, that combined with the in-game parameters and the scenes

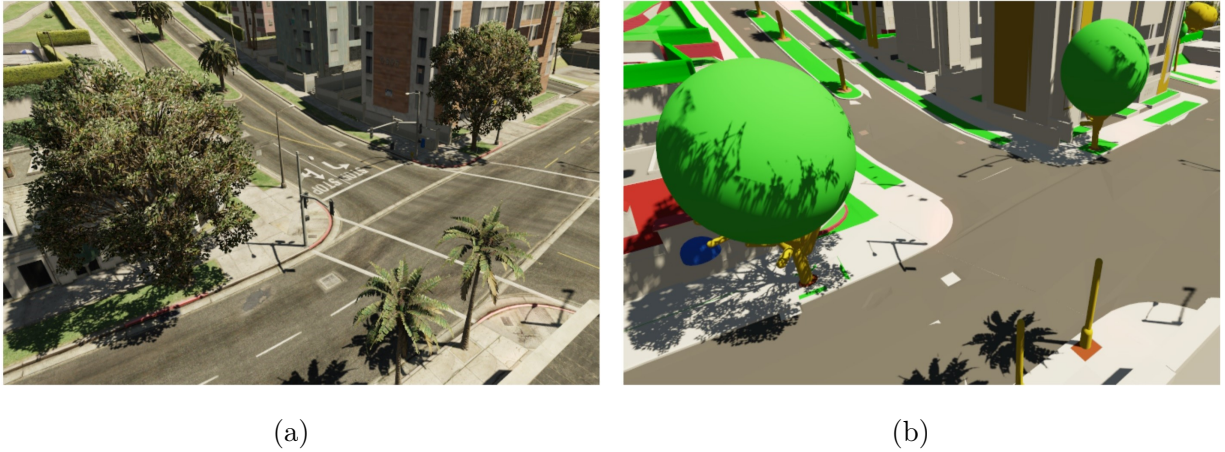


Figure 2.5: Differences between the final rendered scene (a) and the simplified collision meshes (b). From [34].

RGB images allowed the creation of a 3D point cloud description of the environment. This description was then processed using a simulated LIDAR scanning pattern and resulted in the collection of a labelled point cloud. Finally, the sensor noise was computed using Gaussian Noise with 6 millimeters of standard deviation and applied to the depth values of each point.

A pre-trained algorithm designed for pedestrian detection was used to validate this synthetic data generating method. By mixing data from four thousand images from the KITTI dataset [33], with the forty thousand generated frames and applying different data ratios to each iteration of training, B. Hurl et al. concluded that the optimal ratio of data was 10:1, which, using the average precision metric and multiple categories of detection difficulty, resulted in 59.47% in the easy category, 55.57% in the moderate and 49.20% in the hard. Comparatively, the datasets that contained exclusively images with the same origin were registered to perform worst, with the KITTI exclusive dataset attaining the performance values of 57.40%, 51.31% and 44.70% for the categories of easy, moderate and hard detection level, and the fully PreSIL dataset getting 14.13%, 11.85% and 11.62%, respectively. The improvements observed prove, once more, that synthetic data is beneficial to the training of DL algorithms, as a complement to existing datasets and significantly reduces the time required for data collection.

Taking a different approach and building a completely original world via Unreal Engine (UE) version 4.24, A. Dosovitskiy et al. [35] developed the CARLA simulator, an open-source simulator for autonomous driving research. Despite of being primarily designed as a safe environment for algorithm testing, this system is also capable of producing data in

the forms of RGB images, depth maps and semantic segmentation maps. The CARLA simulator operates with two manually pre-build towns of urban and suburban areas, with an integrated system of traffic patterns and dynamical scenarios, emphasizing CARLA’s capability to produce realistic and varied virtual urban environments. The authors of this paper did not release any system’s specificities and do not extract synthetic data for algorithm training. Nevertheless, some tests evaluating the simulation capabilities of the system were performed. Three approaches to autonomous driving were tested with various environment conditions. The approaches employed in testing were:

- Modular pipeline: decomposition of the driving task among the subsystems of perception, planning and continuous control.
- Conditional Imitation Learning: a form of imitation learning that uses high-level commands in addition to perceptual input. Trained with fourteen hours of real world driving data.
- Deep Reinforcement Learning: algorithm trained with a reward based system. The network was trained in parallel agent threads, totaling 10 million simulation steps.

The approaches were then evaluated for navigational tasks: straight driving, one turn destination and completion of distance. The first two tasks were performed without dynamic obstacles and the third was performed for both scenarios. The results of the various approaches performing each task are present in Table 2.1.

Task	Traning Conditions			New Town			New Weather			All New		
	MP	CIL	DRL	MP	CIL	DRL	MP	CIL	DRL	MP	CIL	DRL
Straight	98	95	89	92	97	74	100	98	86	50	80	68
One Turn	82	89	34	61	59	12	95	90	16	50	48	20
Static Navigation	80	86	14	24	40	3	94	84	2	47	44	6
DynamicNavigation	77	83	7	24	38	2	89	82	2	44	42	4

Table 2.1: Percentage of successfully completed episodes in each condition for the three autonomous driving systems tested. From [35].

The experiments performed by A. Dosovitskiy et al. do not evaluate the quality of the synthetic data produced by the system, yet the tests verify that the synthetic environment provides a valid approximation to realistic scenarios.



### 2.2.3 Data Generation from Randomized Virtual Environments

A point-wise annotated point cloud dataset called SynthCity was constructed by D. Griffiths and J. Boehm [36] using 3D models of cities and other densely inhabited urban regions. By using Blender [37], an open-source 3D modeling software, the authors were able to manipulate the virtual environment. Employing duplication and randomized values, the construction of an extended virtual world composed of disconnected urbanized clusters was achieved. To interconnect these clusters, the placement of roads and other urbanistic components was performed manually. The completed environment was scanned using the plugin Blender Sensor Simulation [38] in order to simulate a LIDAR device. To assemble the dataset, seventy-five thousand frames were rendered. Each iteration of scanning was three hundred and thirty seconds long and the estimated completion time was approximately 287 days.

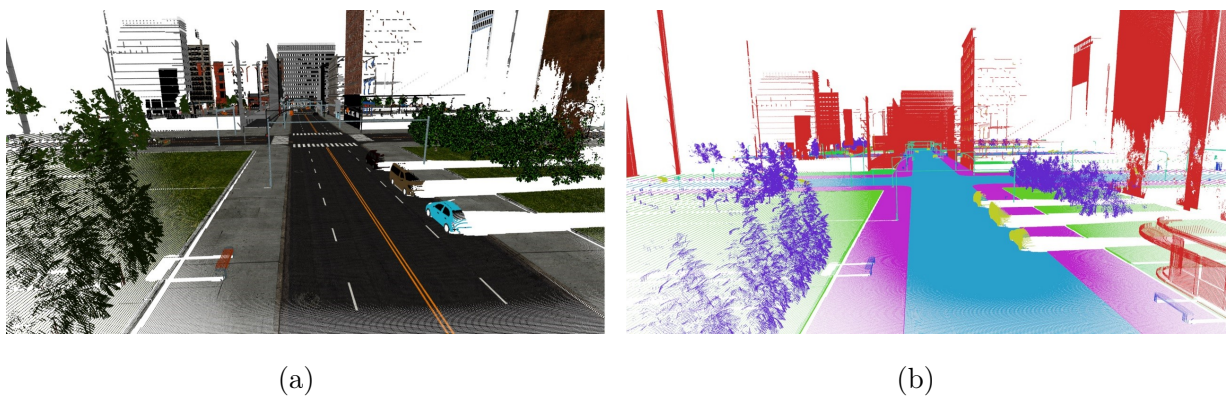


Figure 2.6: Example of the SynthCity dataset displaying RGB values (a) and equivalent class labels (b). From [36].

To expedite the computation time, the authors resorted to the services provided by Amazon Web Services, dividing the work among 22 computers with 8 virtual Central Processing Unit (CPU)s and 61GB of RAM apiece, reducing the total processing time to thirteen days. The extracted data was computed to obtain the absolute world position of each point. In the interest of having a workable dataset, a series of computations applying rotation and transform matrices redefined the position of each point, with coordinates relative to the sensor’s position and even including the addition of Gaussian Noise, with a standard deviation value of 0.005.

## 2.2.4 Data Generation from Procedural Virtual Environments

I. Paranjape et al. [39] created the simulation environment CruzWay, a procedural generation system created with UE4 and using two external tools called TownSim [40] and IntGen [41]. The system creates scenarios that closely resemble a town-sized road network, using modular components. Additionally, it integrates dynamic obstacles controlled through behavior trees that navigate the road network. Being a system destined for simulation and testing trained algorithms, it does not provide valuable knowledge on itself. Nonetheless, this system serves as a proof of concept for the integration of procedural generation systems in the field of autonomous navigation.

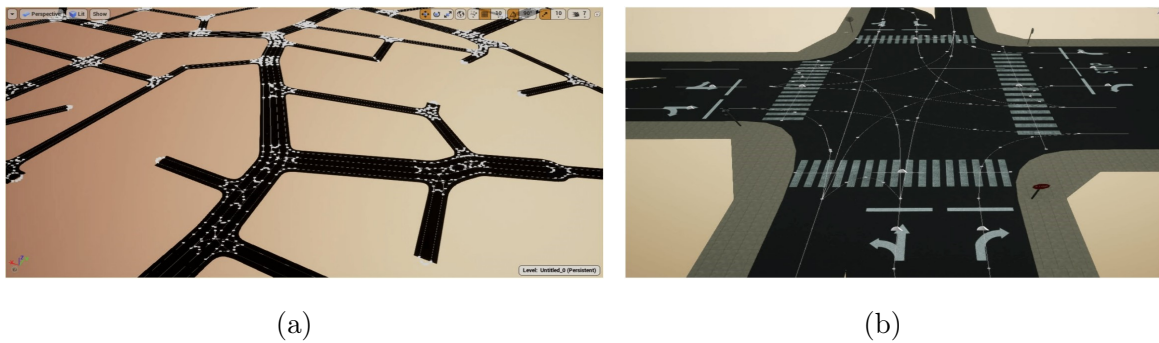


Figure 2.7: Examples of road networks generated using CruzWay. Complete network (a) and intersection (b). From [39].

Using the modular methodology, G. Ros et al. [42] created the Synthia dataset. Utilizing a virtual world consisting of modular components and populated with realistic models of vehicles and pedestrians, the authors were able to generate diverse and realistic synthetic images with pixel-level annotations. An extensive number of tests were performed and the results obtained reinforce the benefit of synthetic data on DL applications.

Adopting an alternative approach to synthetic data generation, M. Wrenninge et al. [43] developed a proprietary system that generates artificial imagery capable of accurately representing the realism of street scenes. The dataset called Synscapes includes a wide range of components typical of urban street environments. This dataset was procedurally generated and includes a variety of scenarios from which a set of twenty-five thousand images was rendered. Each scenario is unique and generated based on various parameters, including geometric and material composition, illumination, among others. The dataset is composed of RGB images, semantic segmentation maps and some meta-data relevant for the field of computer vision. The generated data was tested for both training and testing of autonomous navigation algorithms. The results of the validation experiments presented in Figure 2.8 and

refer to the evaluation of Cityscapes pre-trained algorithm [25] to various synthetic images from different sources. The Table 2.2 presents the metrics results relative to the previous experiment.

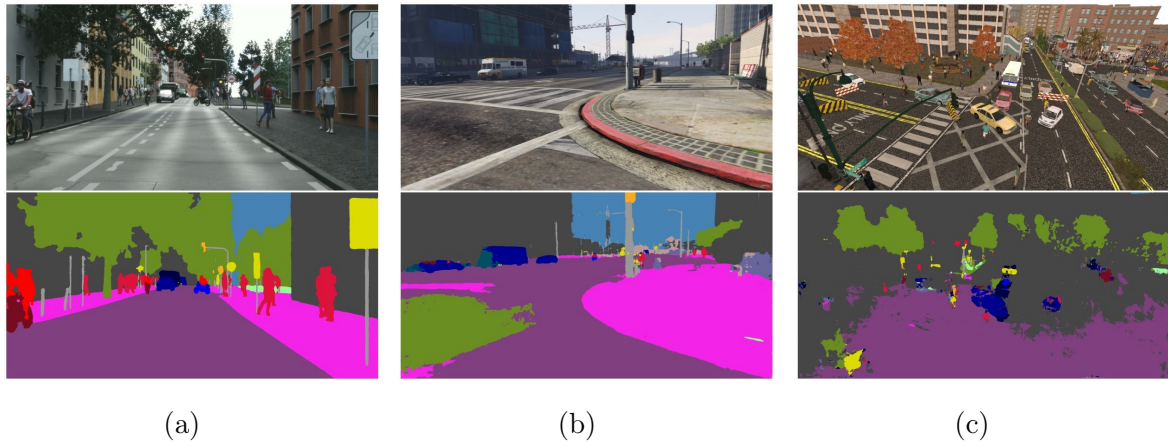


Figure 2.8: Input image and corresponding segmentation prediction by DeepLab[26]. Source of the image Synscapes (a), Richter[29] (b) and Synthia[42] (c). From [43]

Validation Dataset	Cityscapes	Synscapes	Richter	Synthia
Mean IoU	78.79	63.64	45.84	23.89

Table 2.2: Validation on synthetic data for reference version of DeepLab[26]. Adapted from [43].

Applying the procedural methodology to forested environments, C. Newlands et al. [44] developed a cross-platform system capable of generating and rendering realistic forest scenarios. The environment generation follows multiple parameters that define the geometric structure of the map. The terrain generation utilizes Two Dimensional (2D) fractal simplex noise in order to create a realistic heightmap and is constructed with voxel technology. A particularity observed in this system is the tree generation. The authors implemented specialized L-systems, based on the algorithm created by A.Lindenmayer [45], to generate artificial trees that are distributed throughout the voxel based terrain following an ecosystem simulation logic. The end result is a fully navigable virtual environment designed for exploration and testing of autonomous navigation algorithms. The system accomplishes the generation of a completely navigable virtual environment, designed for the research and testing of autonomous navigation algorithms. It is worth noting that the system was developed for simulation exclusively, therefore it does not generate synthetic data. However, it does demonstrate certain procedural generation techniques that may be useful for the

development of a more complete system. Additionally, the authors intended to develop a cross-platform system. To achieve this, the system was developed using OpenGL, compromising the environment’s photorealism. The system is open-source and was made publicly available through the creator’s GitHub repository [46].

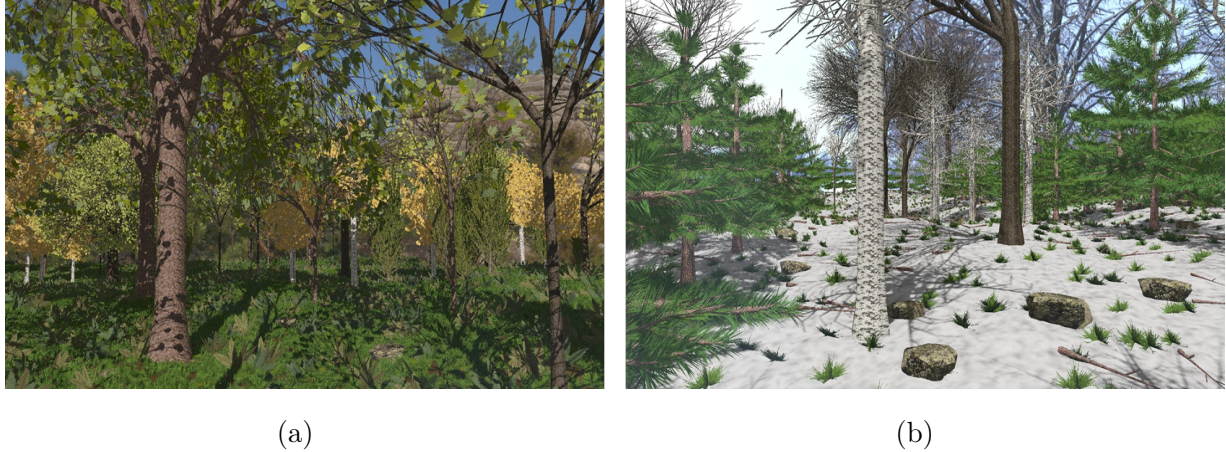


Figure 2.9: Examples of the environment generation. From [44].

Employing the same generation strategy, R. Nunes et al. [19] developed SynPhoRest v1.0, a system capable of constructing complete synthetic datasets. This system extracts RGB images and the corresponding semantic segmentation and depth maps, as well as the point cloud annotation. The virtual environment is infinite and managed using chunk logic. The scenario itself is generated by defining some parameter constrains and the topography is determined by the use of Perlin noise. The environment generation includes a trail generation and a controlled object placement. The authors developed the system using Unity [47], a proprietary game-engine by Unity Technologies, granting the best approximation to photorealism available.

The synthetic semantic segmentation data generated by the system was evaluated using Cityscapes pre-trained algorithm [25]. Despite using an algorithm that was trained on data from the urban area, the authors reported positive results, concluding the success of the system. Despite the system’s success, it has not yet been made available to the public.

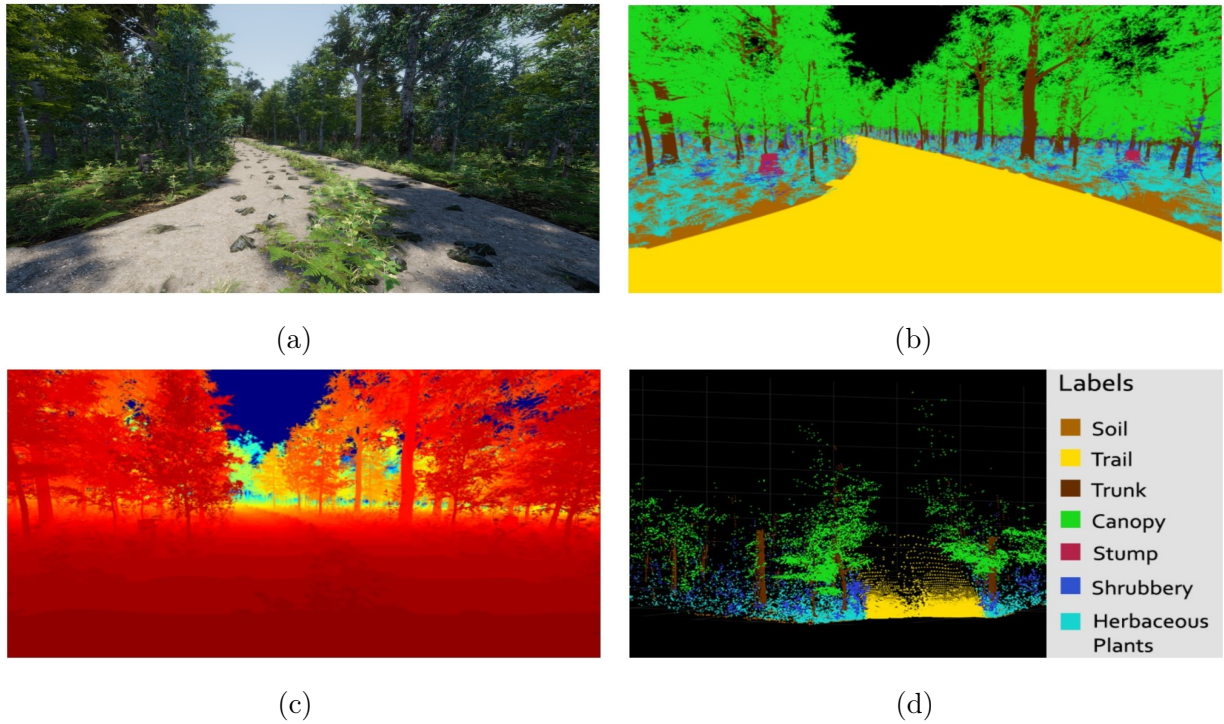


Figure 2.10: Various data types extracted by the system. RGB image (a), segmentation map (b), depth map (c) and point cloud (d). From [19].

## 2.2.5 Generation of Virtual Environments from Real Data

In 2021, the well-known manufacturer of autonomous vehicles, Tesla [48], hosted an event focusing on artificial intelligence. This marketing event was broadcast live on YouTube and demonstrated some relevant scientific results.

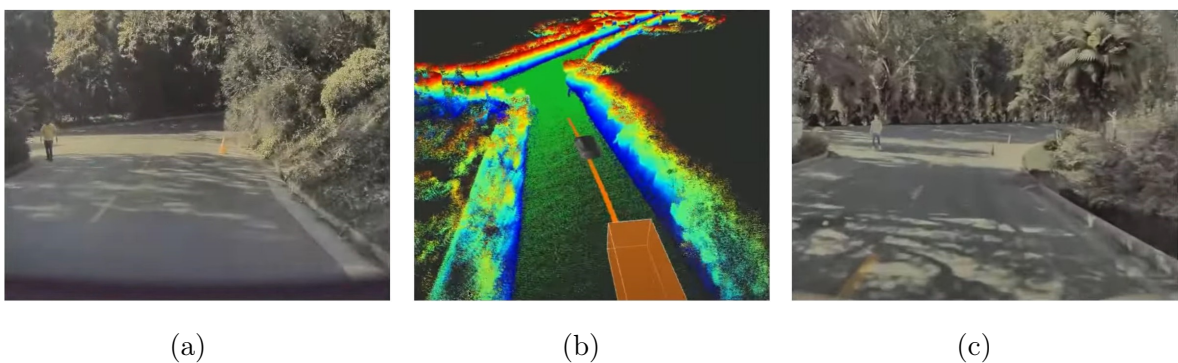


Figure 2.11: Real-world RGB image captured by the vehicle (a), autonomous annotated reconstruction (b) and virtual reconstructed scenario (c). Adapted from [49].

Unexpected and uncommon circumstances are one of the numerous difficulties that autonomous driving must overcome. Due to an absence of training data, developing models to handle this specific circumstances might be difficult. Trying to solve this issue, Tesla uses

procedural generation to create virtual settings based on real-world data collected by the Tesla fleet. Firstly, every vehicle is equipped with eight RGB cameras, that capture the surrounding environment. These eight perspectives are converted in real time into 3D vector space, employing the RegNet algorithm [50] to label every object, in parallel with the Transformer algorithm [51] that merges all eight images and assigns the correct measurements to the objects. This process allows the rendering of a labelled environment that is displayed in the center console. When a Tesla vehicle encounters a condition which it can not handle, it transmits environmental data to Tesla’s network. This allows the recreation of this particular occurrence and the generation of countless variations, producing an abundance of data that can be used to train the models and provide updates to the entire Tesla fleet. The broadcasted event is still available on Tesla’s channel [49].

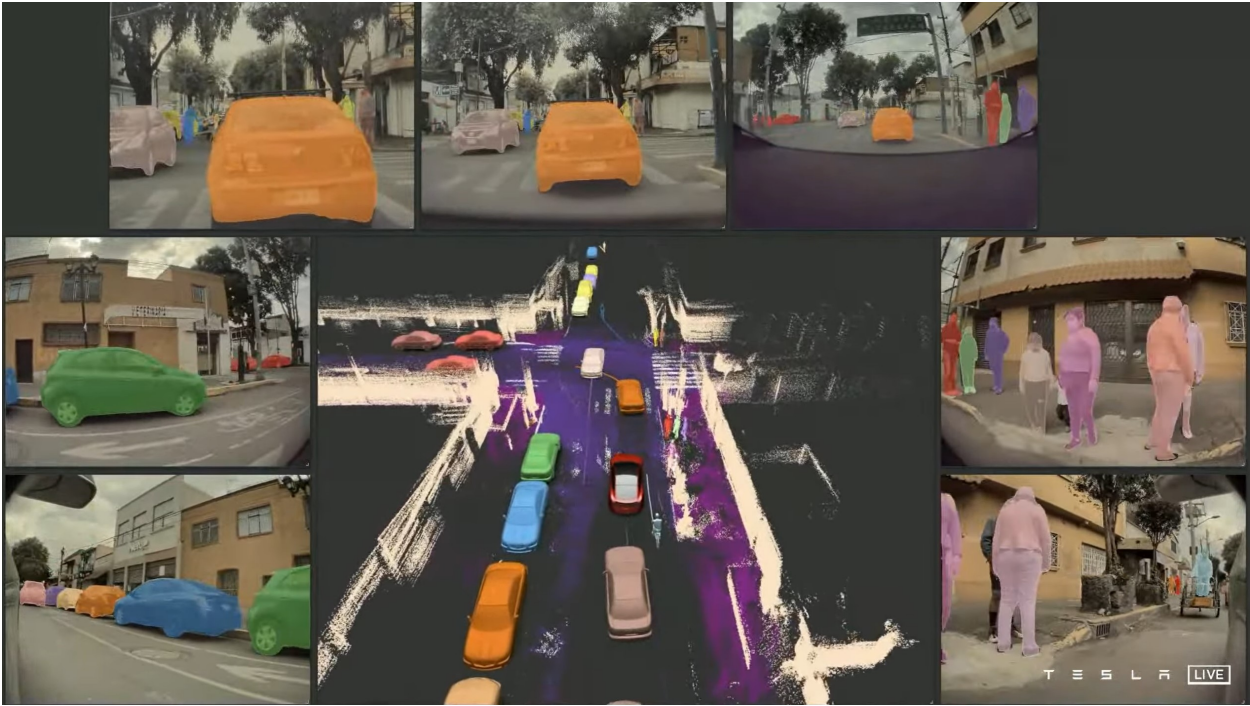


Figure 2.12: Seven of the eight RGB images captured around the vehicle and the recreation, in 3D vector space, of the labelled environment. Adapted from [49].

## 3 Methods

This Chapter presents the many tools and methods that have been fundamental to the development of this project and provides arguments for their choice over the alternatives.

### 3.1 Graphical Softwares

The required photorealism to simulate a genuine forest environment demands the use of an advanced graphical software. In Section 2.2.2, the majority of methods used video-game worlds as research subjects and praised their realism and urban-like qualities. The aforementioned environments were created using game engines, a type of advanced graphics' software that can produce the appropriate photorealistic setting. After determining the kind of software to utilize, one must be chosen among the various available options. There are numerous game engines available but not all are designed to primarily simulate realistic environments, as are the examples of GameMaker Studio [52] and Godot Engine [53] which focus in 2D game development. After extensive research, the list of prospective game engines was reduced to the following three options: Frostbite Engine [54], Unreal Engine 5 [55] and Unity Engine [47].

Frostbite Engine was discarded for being a proprietary software. The last two engines are available for free. A representation of the same environment was utilised to compare the rendering abilities of the engines in order to decide which software was the best choice. Figure 3.1 shows the renderings of each engine. When comparing the two renders, the illumination stands out as the most notable aspect; UE5 offers a superior global illumination model in comparison to the diffused light produced by Unity Engine, making it the best option.

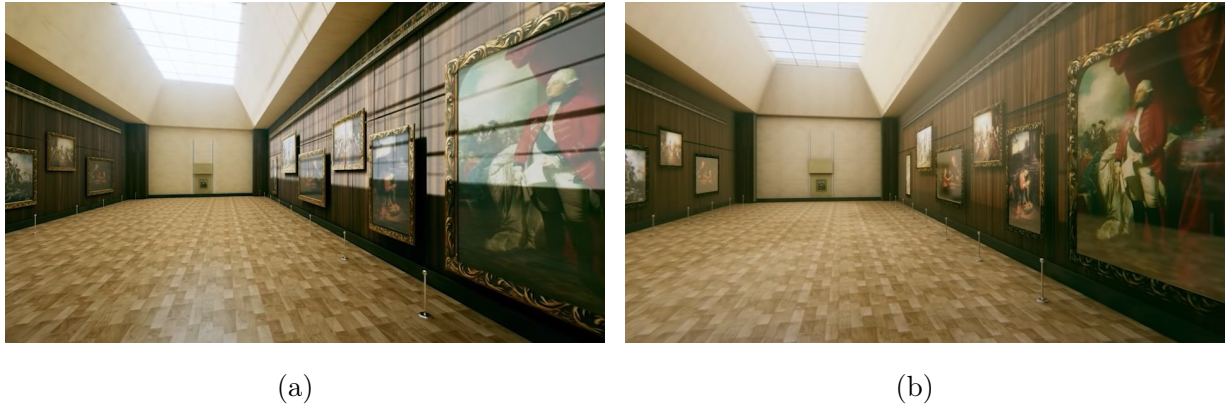


Figure 3.1: Environment rendered with Unreal Engine 5 (a) and Unity Engine (b). Captured from [56].

## 3.2 Unreal Engine 5

UE5, the game engine chosen for the development of SynPhoRest v2.0 system, offers many features that distinguishes it from other game engines. The development of the system was fully made on UE version 5.1, but for simplicity, will be referred as UE5. This Section will focus on the features of the engine and some other more common characteristics that were essential for the project’s development.

### 3.2.1 Lumen

As stated in the previous Section, the illumination demonstrated in UE5 renders allows for the representation of realistic light interaction. This can only be achieved by the use of the exclusive technology called Lumen - a real-time global illumination system, first released with the current version of the engine. Lumen replicates indirect lighting and light bounces in a scene as a dynamic lighting solution, eliminating the necessity for pre-computed lightmaps or static lighting.

This feature has a major effect on real-time rendering since it allows the creation of lighting effects that are more precise and realistic, thereby improving the visual fidelity and immersion of virtual environments. Lumen adds a new degree of fluidity and realism to interactive experiences by responding dynamically to changes in the scene, such as moving light sources, changes in the time of day or modifications to the environment. Furthermore, Lumen’s capability of delivering precise indirect illumination in intricate and dynamic scenarios, without degrading performance or needing extensive calculations, is an amazing development in real-time rendering technology.





Figure 3.2: Comparison between Lumen (a) and lightmaps (b). Adapted from [57].

### 3.2.2 Nanite

Also released for the first time with the current version of the engine, Nanite is an innovative virtualized micropolygon method that reinvents real-time depiction of complex geometry. Nanite eliminates the need for Level of Detail (LOD) models and enables the creation of highly detailed assets, sometimes containing millions or even billions of polygons. It dynamically handles vast amounts of geometric detail by streaming and rendering only the necessary information in real-time. Nanite selectively loads and processes the required details based on visibility and distance, optimizing performance and memory usage. This development is essential because it completely alters how programmers can construct complicated scenarios without sacrificing performance.



Figure 3.3: Model, with Nanite active, at different distances. Adapted from [58].

### 3.2.3 Scripting

UE5 enables programmers to select between two scripting methods for logic and functionality creation: Blueprint visual scripting and traditional code scripting. Although both can be utilized to achieve the same outcomes, they do have some distinct advantages and disadvantages in addition to having different experiences.

Traditional code scripting refers to the use of programming languages, and in the UE5 context specifically C++, to create game logic and functionality within the environment development. This approach requires advanced programming knowledge and complete understanding of concepts and syntax, but gives proficient programmers the freedom and control they need to construct sophisticated algorithms, data structures and other custom systems. Using code scripting, programmers can enhance performance, fine-tune execution, and include external libraries or systems. With advanced programming abilities, developers can add new features and expand the engine's functionality, improving the game's overall usability and behavior. Code scripting is preferred in collaboration environments, however it requires effective and comprehensive communication between programmers and designers/artists.

Blueprint visual scripting enables developers to create in-game logic and functionality without the need for traditional coding. It utilizes a node-based interface where pre-built nodes representing various functions, actions, conditions and variables can be connected to define the in-game behavior. Blueprint scripting offers a more accessible and intuitive approach while still requiring previous programming knowledge. The early stages of development are facilitated by blueprint programming, which encourages quick iteration and experimentation. The visual representation obtained by this scripting method greatly benefits collaborative workflow. The most appealing feature of this method may also be its biggest hindrance: its visual nature may induce performance overhead.

It is worth noting that the scripting methods are not mutually exclusive and can be used together to compensate the shortcomings of one another. A blueprint node that is not included in the default UE5 library, can be coded using C++ and integrated in the logic. In fact, the proposed system integrates both forms in order to achieve the desired results.

### 3.2.4 Extension Capabilities

UE5 has access to EpicGames Marketplace, a digital platform where developers can discover, purchase and sell a wide range of assets including 3D models, animations, sound effects, plugins and even complete projects. This greatly expands the engine capabilities and

content available. During SynPhoRest v2.0 development, two types of these products were actively searched and tested: the plugins and the asset libraries.

Plugin integration enables the seamless integration of third-party tools and the expansion in functionality of the engine by providing specialized features not included in the default installation. This integration empowers developers to leverage external expertise, accelerate development timelines and add features without the need to build them from scratch. Asset libraries offer a vast array of ready-to-use assets, including high-quality 3D models, animations and sound effects. With the help of these marketplace products, development time may be distributed more effectively, public resources can be created and smaller development teams, with less experience and knowledge, can achieve their envisioned goals.

### 3.3 Procedural Generation

The process of creating content or data algorithmically rather than manually is known as procedural generation. The method involves creating something dynamically by utilizing a variety of rules, algorithms or mathematical operations, resulting in varied and often unique outcomes. Research is constantly focused on this process and has observed incredible evolution, being documented to be a beneficial tool in many research fields [59].

Procedural generation has significant applications across different domains and industries, shaping the experiences and content people encounter in their daily lives. Mainly focusing in visual representations, procedural generation is employed in the majority of visual effects present in movies or television series. Taking as an example the 2022 movie, *Avatar: The Way of Water*, generated all visible vegetation using procedural generation [60]. Other not so notorious examples are the generation of dynamic and realistic effects, such as fire, smoke, water, explosions and natural phenomena like clouds and weather patterns [61]. The video-game industry also utilizes procedural generation, the popular games *Minecraft* [62] and *No Man's Sky* [63] use procedural generation to create infinite open-world experiences.

Procedural generation can be used to create a convincing and realistic topography. K.S. Emmanuel et al. [64] documented the basic tool for terrain generation, Random Number Generator (RNG) algorithms. RNG take various forms and the most adequate for this subject are Noise Generators. A. Lagae et al. [65] performed an evaluation on Noise Generators. Although, various noises were tested, they were evaluated comparatively to Perlin Noise [66], indicating that it still is the best approach to noise generation. The Perlin Noise creator, Ken Perlin, in 2001, documented the flaws of its original creation and developed a deriva-

tion called Simplex Noise [67]. Simplex Noise, compared to Perlin Noise, is computationally lighter and easier to implement in hardware. Additionally, it does not have noticeable directional artifact, has a well defined and continuous gradient and can be scalable to the forth and fifth dimension.

To construct a terrain mesh that closely resembles natural topography only requires the generation of 2D simplex noise. This produces a 2D matrix where each value represents the height at a given point. However, when applied to a mesh, it translates to a smooth undulation, i.e., a non-realistic terrain. This can be solved by stacking multiple levels of noise and varying the influence of each layer, adding micro-variations and affecting the final noise as seen in Figure 3.4. The Figure 3.5 demonstrates the previously mentioned effect of noise when applied to a mesh and the effect of noise layering.

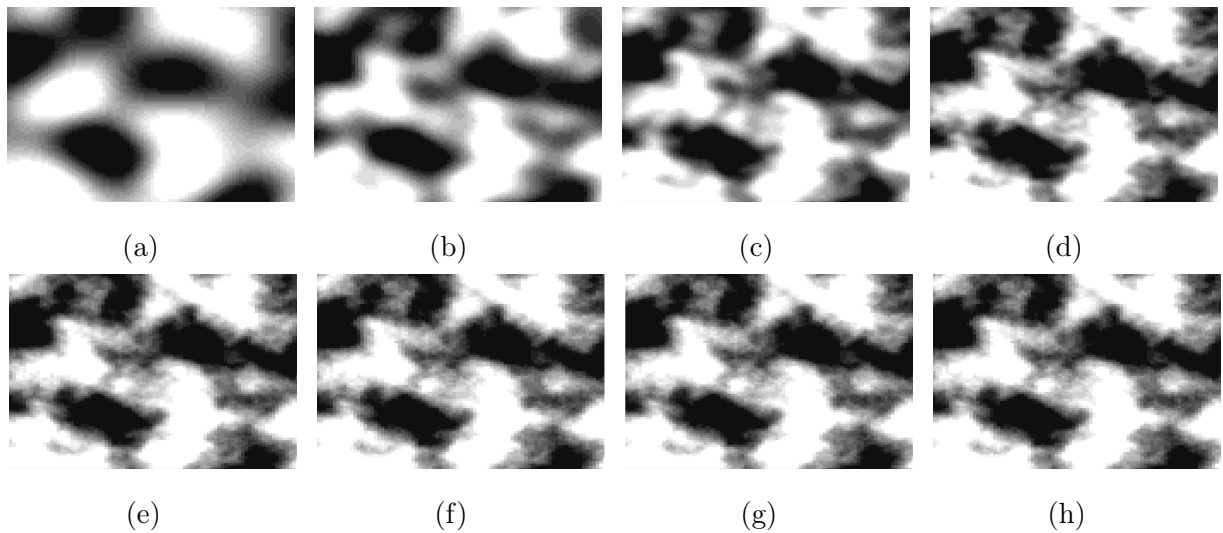


Figure 3.4: Simplex noise generated through layering. Number of layers applied: one (a), two (b), three (c), four (d), five (e), six (f), seven (g) and eight (h). Generated using [68].

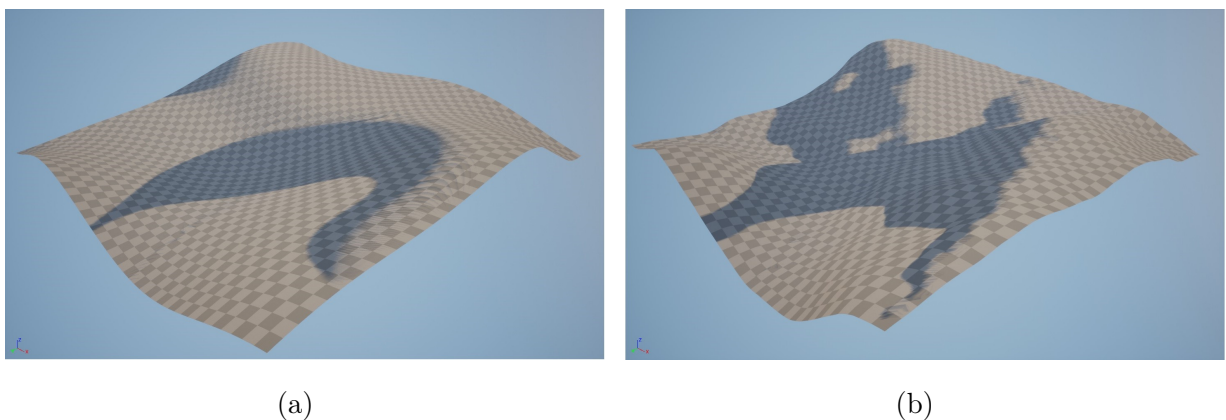


Figure 3.5: Influence of one (a) and ten (b) layers Simplex Noise in a terrain mesh.

The Simplex Noise algorithm is not available with the default installation of UE5. The integration of this algorithm was achieved with an open-source plugin called SimplexNoise [69].

### 3.4 Extraction of Synthetic Data

Virtual cameras are an idealized version of real, physical cameras. A virtual image, without post-processing methods, will produce the most accurate representation of the captured object. Real cameras, on the other hand, create images with artifacts like grainy patterns and noise. These artifacts are caused by a number of factors, including dim illumination, highly sensitive sensors, prolonged exposure times, heat or physical deterioration of the camera. Additionally, noise is an important tool in DL and has been documented by T.S. Nazaré et al. to improve algorithm resilience and adaptability [70].

To closely recreate real camera attributes, noise must be added to the system’s extracted images. UE5 cameras have post-processing settings that allow the recreation of real cameras. Noise is added by applying film grain, a post-processing feature based on Gaussian noise, to the RGB images. In Figure 3.6 are presented both types of capture for comparison.

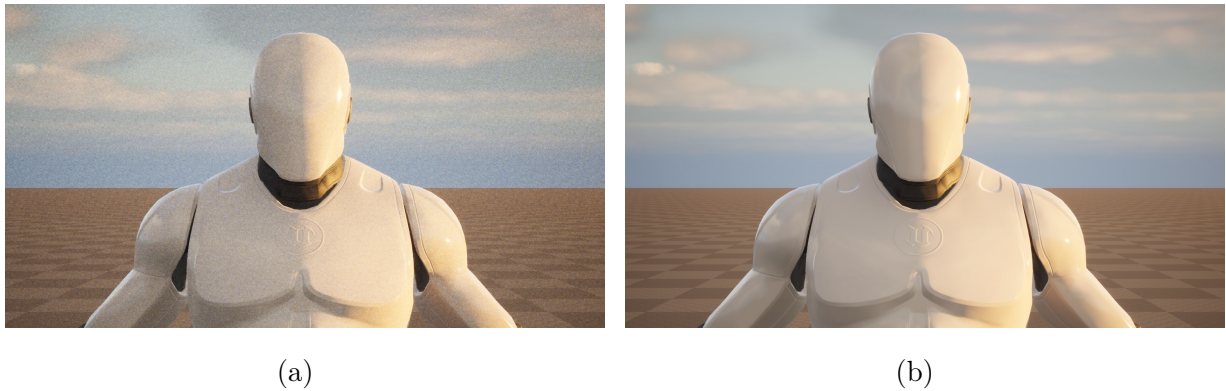


Figure 3.6: Same scene with (a) and without (b) film grain.

# 4 Generation of Synthetic Forest Environment

Forest environments are intricate landscapes consisting of several components, that extend across a significant area. In an ideal simulation, the environment would be infinite. However, due to the constraints of computer graphics and computing power, an infinite, fully rendered world is impossible to generate. The closest viable alternative consists of a system that generates an approximation of this infinite world by generating and rendering only the portion of world that is visible by the camera. To accomplish this, the world needs to be split into smaller segments. This segmentation will result in finite and contained portions of the world, called chunks. These chunks represent a part of the world and contain every element that composes the scene at that specific place. However, partitioning the world into chunks is not sufficiently granular to perform procedural generation. The elements include the terrain, paths, vegetation, among others, requiring a fragmentation of each chunk into smaller, more manageable pieces. This Section will address this fragmentation and the logic behind each element, as well as their construction and placement in the final scene. Finally, the interaction between each element will be described and their role in the final system explained.

## 4.1 Terrain

As determined previously, a chunk represents a portion of the larger world. Therefore, it has a constant size throughout its iterations. Similarly, the terrain generated for each chunk will follow the same size constraint.

From a computer graphics standpoint, and following a 3D representation, the terrain is simply an arrangement of vertices and triangles evenly distributed along two axes and the third axis represents the height value at that point. Following the integrated referential in

UE5, the terrain is generated along the X and Y axes and the height value is assigned to the Z axis. In order to expedite the construction process, it was defined that the chunks will be equilateral in the X and Y axis, to allow the construction of a chunk grid composed of squares. This decision prevented the usage of extra values and offsets during the placement of each chunk onto the grid. Each chunk acts as an actor in the UE5 context and, as such, has access to the functions related to the Actor object class.

### 4.1.1 Mesh Construction

The mesh construction that will represent the terrain begins by generating a 2D grid of equidistant points. The distance between points is determined by the chunk size, described above, and the number of points needed to generate the grid. The height value assigned to each point aims to simulate the natural topography of the forest environment. The Simplex Noise Algorithm was employed following the study documented in Section 3.3. UE5 does have an integrated noise generator but it does not produce the required noise. However, through some adjustments to the code from an open-source plugin created for a previous version of UE [69], it was possible to integrate the noise generator in the construction process. With the plugin incorporated, the height value is calculated by a node that generates 2D simplex noise and returns the value of the noise at the given coordinates. The node produces an output dependent on the following parameters:

- **Point Coordinates:** 2D coordinates of the point in analysis, relative to the global world referential.
- **Octaves:** Number of layers that will be combined to obtain the noise output. This variable is required to be a positive integer.
- **Lacunarity:** factor of frequency increase by octave. This variable can assume any real value above 1. The most common value is 2.
- **Persistence:** determines amplitude behavior of each octave, i.e., the weight of each octave in the final output. This variable can assume any real value between 0 and 1, but the most common value is 0.5.
- **In factor:** value of the frequency. Represents the level of detail per unity of space.

In addition to these parameters, the simplex noise function uses a global optional parameter, called global seed, that establishes the behavior of the function and allows multiple

outputs for a fixed set of previous parameters. The definition of parameter and the value assigned to it is explained in 4.4.2. The function outputs the normalized value of the noise calculated at the point given, with values ranging between -1 and 1. This value is then multiplied by the terrain elevation factor in order to generate increased height variation.

Since this is a construction process developed for the multiple iterations, the expected point coordinates by the function need to be relative to the world. To achieve this and guarantee the continuity between chunks, the point coordinates are calculated by retrieving the actor location in the world and the predefined chunk size. The coordinates are saved in an array of three-dimensional vectors called Vertices. Following computation of the points and respective 3D coordinates, the construction of the triangle logic and texture attribution take place, by creating an array of point indexes and respective texture coordinates in a separate array.

All the values calculated are then used to create a mesh Section. First, the arrays are used as input to the UE5 function that calculates the tangents and normal vectors for the final mesh. Subsequently, the five arrays are used to generate a Procedural Mesh Section, a type of object used by UE5 to create meshes defined by vertices generated at runtime.

### 4.1.2 Object Positioning

During the construction of the terrain's mesh grid, the coordinates of the center point are saved and used as an internal seed for the chunk. This allows the creation of a Random Stream variable, that generates a list of random numbers exclusive to the chunk. This variable is then used to create multiple arrays of four-dimensional vectors that will hold the information related to the objects that will be placed in the chunk, namely vegetation. The random stream is used to differentiate the type of object that will be placed, by generating percentile values and inserting them into one of three classes according to the associated density.

With the class of object defined, the random stream generates float values used to specify the position of each object, following the logic present in Figure 4.1, the scale and the rotation around the Z axis of said object. This process is applied to all classes. However, if the class represents the objects associated with ground vegetation, an additional computation is performed. Before the attribution of values related to positioning, scale and rotation, the random stream is used to determine how many objects will be associated with the grid point in analysis.



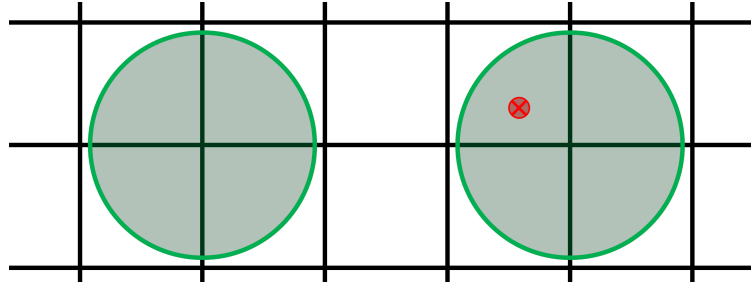


Figure 4.1: Diagram of the object placement logic, seen from above. Green circles represent the admissible area of placement and the red circle represent the random place attributed to the object.

It is worth noting that only the class of the object is decided in this process. The other attributes are defined latter, during the placement process.

## 4.2 Path

The fundamental essence of autonomous driving algorithms lies in their ability to detect the optimal path. Taking the urban environment as an example, these algorithms will choose the road over a sidewalk. In nature there are no defined roads, however most of these scenarios include trails or pathways that facilitate the navigation of the environment. To replicate these paths and include them in a procedurally generated world, and taking into account the building structure of the latter, it is necessary to implement a path generation module capable of creating a continuous non-linear path that blends with the existing topography. In order to satisfy these requirements, it was concluded that the path generation would operate separately from the chunk generation. This decision allowed the construction of a larger path and avoided discontinuity in chunk borders.

The path generation starts with the creation of a UE5 spline component, by assigning twenty-five equidistant random points. A spline is a function that is defined by a set of points and their associated tangents. The coordinates of these points are determined by the global seed, except the Z coordinate. By fixing the Z coordinate, all points are ensured to be coplanar. This twenty-five-point spline is then resampled, and the number of points is multiplied by a factor of fifty. The resampling of the spline is performed in order to achieve photorealism. Increasing the number of points in the spline grants the construction of smoother curves that translate into smoother path edges.

The resampled spline is then projected downwards, towards the already placed chunks, to perform a collision trace. Iteratively, each point's projection is performed an odd number of

times to guarantee path symmetry. Initially with a negative offset applied to the right vector of each point. In each iteration, the offset gradually varies until it reaches the opposite value used in the first iteration. The result is an array of points, organized in longitudinal order, characterized by the coordinates of collision of each projection with the chunks. Additionally, the collision points collected from the projection parallel to the Z axis are used to generate a new spline. This spline is planar with the terrain and is used as navigation guide for data extraction. Finally, the collected collision points are used to construct a procedural mesh component by applying the same method used in Section 4.1.1. Figure 4.2 presents a simplified version of this process, with a visual representation of each step.

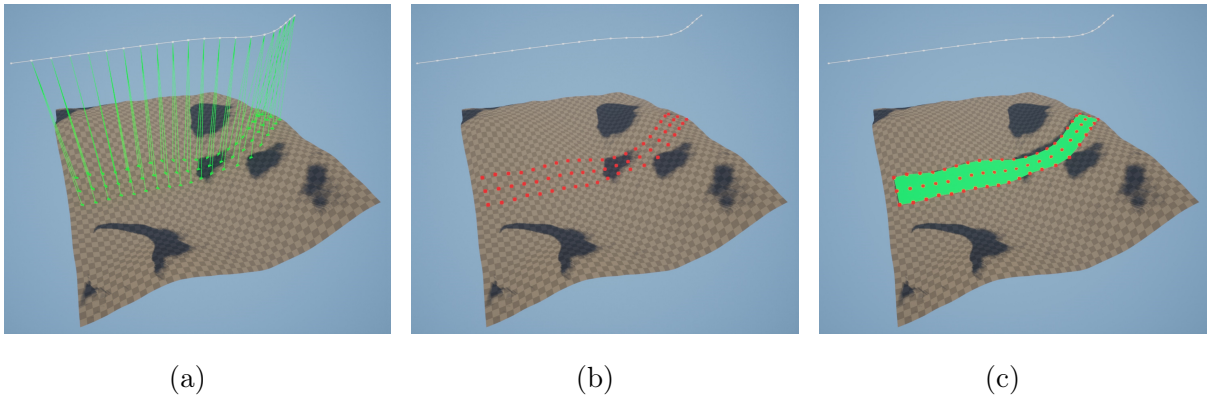


Figure 4.2: Simplified version of the path building process. Projection of the spline downwards (a), collision points obtained through the projections (b) and mesh built with the collision points (c).

Considering that the length of the spline generated for the creation of the path may exceed the limits of the already placed chunks, an update feature was implemented to allow the path to be reconstructed whenever a new chunk is added.

### 4.3 Materials

The two components mentioned in the previous Sections are constructed using the UE5 available functionalities. These components lack textures or materials because they are entirely constructed using the developed logic. As integral parts of the generated environments, a suitable material for each component was built by combining textures available in EpicGames Marketplace. Figure 4.3 depicts the materials that were produced.

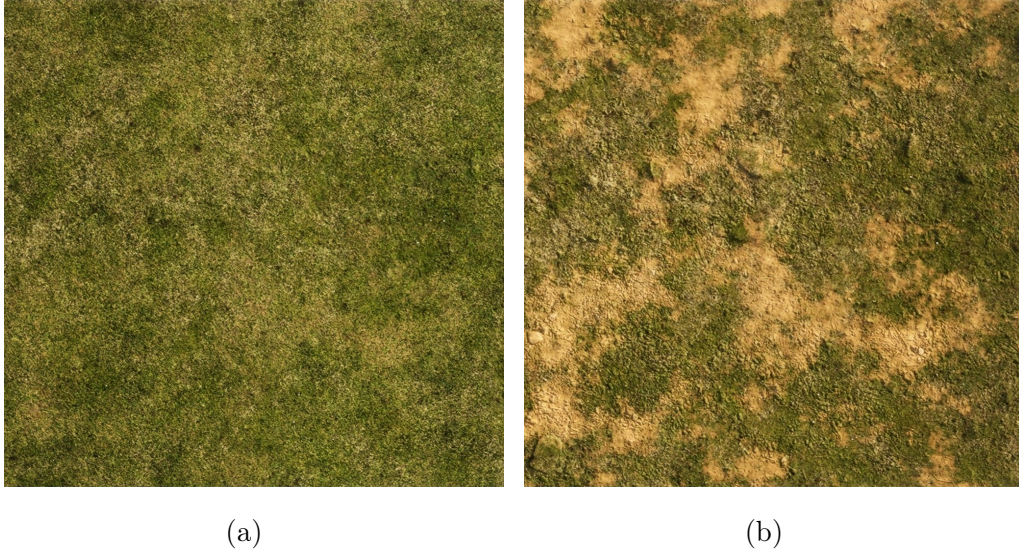


Figure 4.3: Materials built for the terrain component (a) and path component (b).

## 4.4 Environment Assembling

The assembling of the forest environment with the described components is performed by a special class of blueprint, the Level Blueprint. This blueprint is executed only once at the start of the level and, therefore, the optimal choice to accommodate the world building logic.

### 4.4.1 Chunk Management Module

As stated earlier, the best approximation of an infinite world is the generation and rendering of what is visible to the camera. To accomplish this for a static camera means that the only computations needed are the generation and rendering of a fixed number of chunks inside the frustum of the camera. However, with a moving camera, it is necessary to manage what and where something is rendered and what should be deleted to free up memory and improve performance. To fulfill these requirements, a chunk manager blueprint was developed. In the context of UE5, all moving objects are classified as actors, and as such, the camera actor has access to the functions associated with that class.

The task performed by the chunk manager is an iterative position checker and is repeated every tick, i.e., every frame and is dictated by the four following variables:

- **Active Actor Position:** The world position of the camera actor.
- **Render Distance:** The number of chunks visible from the active actor's position in all directions.

- **Rendering Radius:** Product of the Render Distance and the Chunk Size.
- **Last Chunk Visited:** A register of the chunk where the active actor was in the previous tick.

As the active actor moves, its position and corresponding chunk are compared with the Last Chunk Visited. If the active actor moves to a different chunk, it is performed a search, in order to identify the chunks outside the rendering radius for deletion. The placement of the new chunks is performed by verifying if the available positions inside the rendering radius do not have an already loaded chunk. Figure 4.4 presents a visual description of the logic applied.

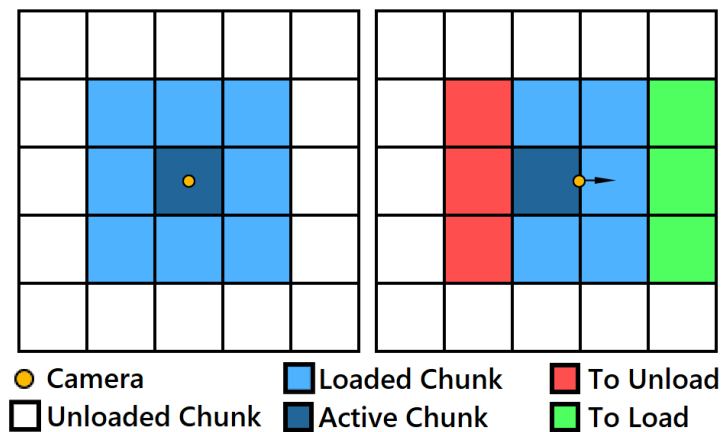


Figure 4.4: Visual representation of the chunk manager logic.

During development, it became clear that as the complexity of the scenario increased, the performance of the system decreased. The rendering and deletion of the chunks became unmanageable, taking multiple seconds to perform. Numerous approaches have been tested to solve this problem, primarily reducing the rendering radius to address this performance issue, and many parameter combinations have been tried. However, the computations needed to delete old chunks, generate and populate the new ones were too intensive. Additionally, the primary component in chunk generation, the Procedural Mesh Component, is documented as an experimental, computation heavy feature [71]. As an alternative to this limitation, it was decided to suspend the usage of the chunk manager and instead place a single mega chunk in the world. This process increased the computation needed for the initial world generation, however, it removed all the issues during runtime. As a result of these adjustments, from this point forward, the world generation will be explained by considering only one chunk.

## 4.4.2 World Building Logic

This process begins by assigning a value to the global seed. This variable guides the pseudo-random generation that is performed by all components. The global seed is determined by taking the start time of the execution and appending the values of each unit to generate a random integer. This integer is then used as a seed to a random stream variable, in order to generate a ten-digit number. In turn, this ten-digit number is assigned as the global seed's value for the current iteration of the world. This value ensures the worlds reproducibility for future iterations. With the seed defined, the chunk component is generated and placed in the world map, followed by the path component.

### Object Placement

As unpredictable as a forest environment can be, there are some behaviors that can be determined. In nature, when a trail or pathway is used repetitively, the ground becomes coarse and hardened, preventing the growth of larger vegetation. To replicate this phenomenon, object placement must be performed with some constraints. Thus, by using the information of positioning generated during the terrain construction and the points collected by building the path, an approximation of the natural object placement can be achieved.

The several four-dimensional arrays generated during the terrain construction contain the 2D coordinates, the scale and the rotation around the Z axis for each object, of each class. The coordinates are used to verify the placement of the object, by performing a downwards projection and checking the collision coordinates. The projection is configured to ignore all collisions except the ones with the path mesh. By not identifying a collision coordinate, the process of placement resumes, and another projection is performed to determine the value of the Z coordinate for the object placement. Occasionally, two or more objects may share placement coordinates and be placed in the same spot, resulting in immersion breaking visuals. Luckily, UE5 includes a feature that manages this type of situations by placing the second object at the periphery of the first. The placement verification logic is illustrated in Figure 4.5.

Finally, using a random stream, controlled by the global seed, a mesh is selected from a library correspondent to the class of the object, which is then placed in the world. The collection of assets used for the objects was gathered through EpicGames marketplace [72, 73, 74, 75, 76]. The chosen assets are shown in appendix A.

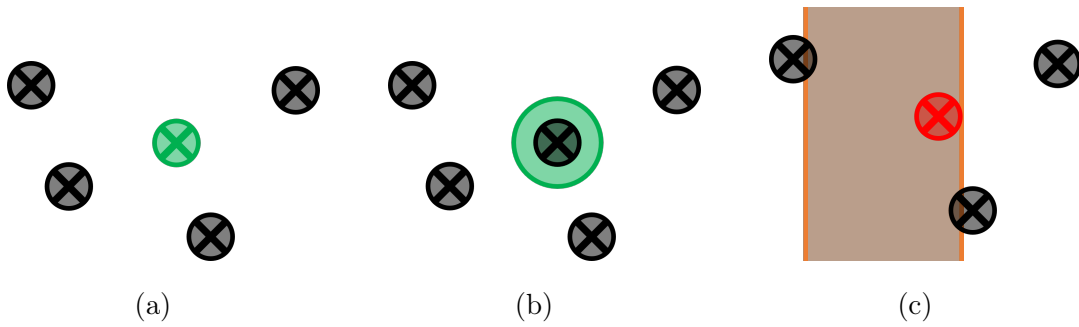


Figure 4.5: Visual representation of placement verification logic. Black circles represent already placed objects and the orange rectangle represents the path. The green and red circles represent the to be placed object and the validity of its location. Correct placement (a), adjusted placement (b) and no placement (c).

## Illumination

The illumination of a scene can make or break the photorealism and the method of illumination integrated in UE5 was one of the reasons for its selection in this project. The illumination model, lumen, operates in real-time. It is a fully dynamic global illumination method that calculates indirect lighting and accurate light propagation while also enabling realistic interactions between the light surfaces. Lumen eliminates the need for pre-computed lightmaps and allows for more flexible and interactive illuminated scenarios. Additionally, the lumen illumination method, with its dynamical properties, provides an emulation of the eye and lens adaptation on varied illumination conditions.

The global illumination of the world is composed of a mixture of light and ambient components, that work as a unit to deliver an accurate and atmospheric illumination. The present components are as follows:

- **Directional Light:** component that emulates the primary light source. Main component in shadows and lens flares calculations.
- **Sky Lighting:** uniform secondary light. Primary source for the non-direct illumination.
- **Sky Atmosphere:** emissive component that controls the behavior of the light that reaches the ground.
- **Sky Sphere:** Visual component that emulates the appearance of the sky.
- **Volumetric Clouds:** Visual component that generates and controls clouds' behavior.

- **Exponential Height Fog:** visual component that creates realistic atmospheric fog effects for light scattering. Allows to visually blend the sky sphere with the horizon line.

These six components have multiple parameters that allow precise control over the behavior of light and its vast interactions. Aiming to recreate the best approximation to natural illumination, various combinations of parameters were implemented. Being a subjective issue, the parameters went through a process of iterative experimentation and refinement until the final combination was achieved, resulting in the implementation of three types of illumination.



Figure 4.6: Environment with the three definitive illumination conditions, day (a), dusk (b) and night (c).

## 4.5 Graphical User Interface

As a full-fledged system, SynPhoRest v2.0 has a Graphical User Interface (GUI). A GUI serves as a critical component in facilitating user interactions and enhancing the overall user experience. It provides users with a visual platform through which they can interact with the system, making it a fundamental element in ensuring usability. As an intermediary layer between the user and the system functionalities, it gives control over the editable settings of the world generation, as well as information about the controller scheme.

### 4.5.1 Main Menu

The main menu serves as an entry point and navigational hub for users, providing access to available functionalities. Composed of just three buttons, it organizes and presents the available choices in a structured and intuitive manner, enabling users to easily locate and select the desired primary world generation options. The "Custom Mode" button grants

access to personalization options and the "Random Mode" button begins the generation of a default map.



Figure 4.7: SynPhoRest v2.0 main menu.

## 4.5.2 Custom Mode Menu

The custom mode menu provides the users with a comprehensive range of options and parameters in order to personalize their experience and adapt the system to meet their specific needs. Users can utilize the "Seed Input" option to specify a particular seed value, enabling consistent reconstruction across different executions. The "Output Path" option allows users to define the desired destination folder or directory for saving the extracted data. Additionally, the "Time of Day" option provides the flexibility to adjust the lighting conditions, allowing for dynamic environments or specific atmospheric effects. The "Output Resolution" setting enables users to select the desired image resolution for every type of generated data, ensuring optimal quality for their intended purpose. Users can also define the "Number of Random Captures" value, specifying the quantity of randomly generated images. The "Vegetation Density" option offers control over tree density and consequently influencing the placement of other components. Moreover, the "Max. Terrain Elevation" setting enables users to define the upper limit for the elevation of the generated terrain. The "Aerial Camera" and "Disable Path" options provide users with the ability to enable or disable drone perspective and path integration, respectively.



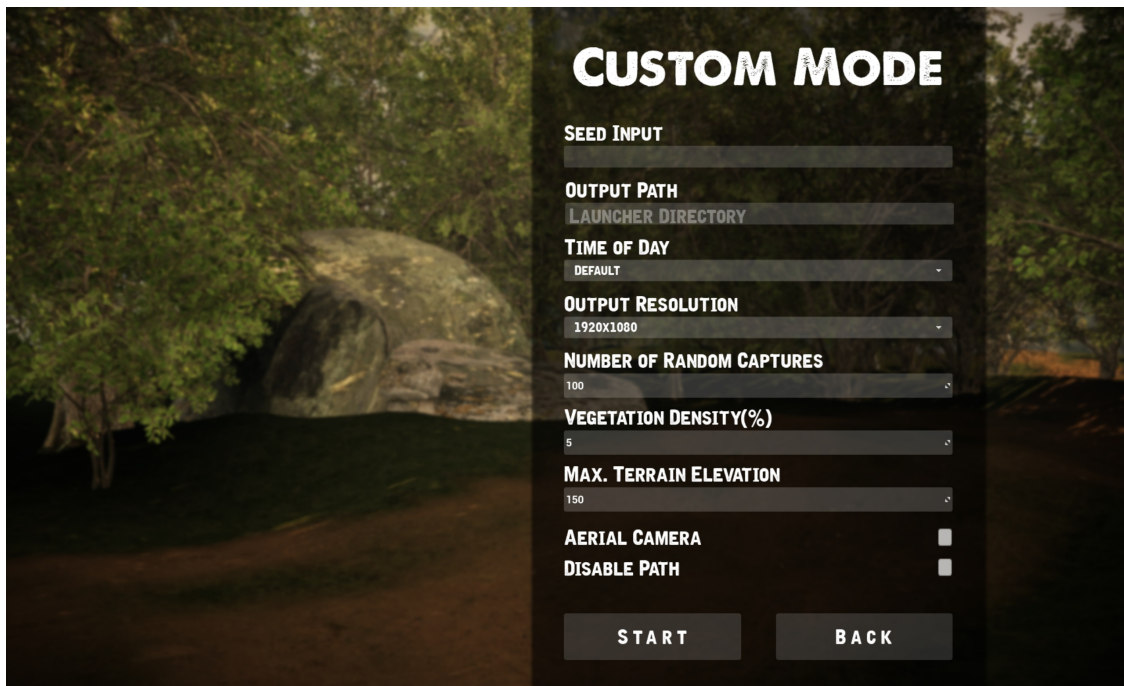


Figure 4.8: SynPhoRest v2.0 custom mode menu.

### 4.5.3 Pause Menu

The pause menu offers convenient access to essential information over the system's button mapping. The "World Seed" information provides users with seed value of the current world seed. Additionally, the option "Copy Seed" is available, enabling users to easily save the seed value in the computer's clipboard, enabling future reconstructions. This feature is not integrated in UE5 and a plugin was specifically developed for the effect. Another crucial aspect of the pause menu is the "Sequence Controls" information, which provides the user with the button mapping of the various capture sequences. "Auto Pilot" enables the automated sequence of captures along the path, "Random Capture Sequence" enables the random placement of the camera on the map. "Custom Capture" executes the capture of the current perspective. Furthermore, the pause menu includes "View Toggles" listing, informing the users the button mapping to switch between different visual mode. "Semantic Segmentation" and "Depth Map" activates or deactivates the view mode of the same name. Finally, the user has access to three buttons, that redirects the system accordingly to their names. The navigational button mappings are not displayed because they are typical of computer video-games.



Figure 4.9: SynPhoRest v2.0 pause menu.

## 4.6 Extracted Data

The main reason for creating this system is its ability to perform the correct data extraction related to forested environments. The capture of real-world data is performed on site, using various cameras and sensors. To translate these tools to the generated virtual environment, it requires the simulation of the individual systems. The following Section explain the methods used for the simulation of each capture methods along with the explanation of the data generated.

### 4.6.1 RGB Images

The first type of data extracted is the RGB image and is used as input data to the training and validation of DL algorithms. RGB images represent the actual appearance of the scene and are the primary source of data for deployed models. The RGB data captured by the two available perspectives in the system are presented in Figure 4.10.

To extract this type of data, two 2D scene capture components were attached to the in-game cameras associated with the player character, one to each prospective. This relation assists the translation of the in-game camera movement to the the respective 2D scene capture component, assuring the capture of the user’s desired scene. As explained in Section 3.4, the addition of noise is performed through post-process camera settings.




Figure 4.10: RGB images captured from the ground perspective (a) and the aerial perspective (b).

## 4.6.2 Semantic Segmentation Maps

The semantic segmentation maps offer information about the different objects present in the image. These maps provide pixel-perfect annotation and classification related to each object captured. The generated data distinguishes the objects by assigning a class, and its corresponding color, to the area that the object occupies in the image. The system is able to detect and classify the following classes:

- **Background:** Sky or far objects that do not influence the scene.  
Class color:
- **Ground:** Terrain or soil.  
Class color:
- **Path:** Recommended traversable area or trail .  
Class color:
- **Tree Trunks**  
Class color:
- **Tree Canopy**  
Class color:
- **Dead Tree**  
Class color:
- **Stump:** Remains of rotten or cut down trees.  
Class color:


- **Ground Vegetation**

Class color: 

- **Shrub:** All vegetation that has a solid trunk but is not classified as tree, such as sapplings.

Class color: 

- **Obstacle:** Solid object that hinders or impedes traversal, such as rocks or logs.

Class color: 

The semantic segmentation image is captured with the previous described 2D scene capture components. To attain a correct and precise classification of each object, the scene illumination is removed and the materials of each object are replaced with a solid material specifically created for the effect. The materials are build using an unlit model, in order to remove any light interaction. These materials are devoid of any texture, enabling a correct and smooth masking. Due to the absence of light, these materials have emissive properties and project to the camera their respective colors. When the capture is completed, the illumination and original materials are restored. The semantic segmentation data captured by the two available perspectives can be observed in Figure 4.11, corresponding to the RGB images presented in Figure 4.10.

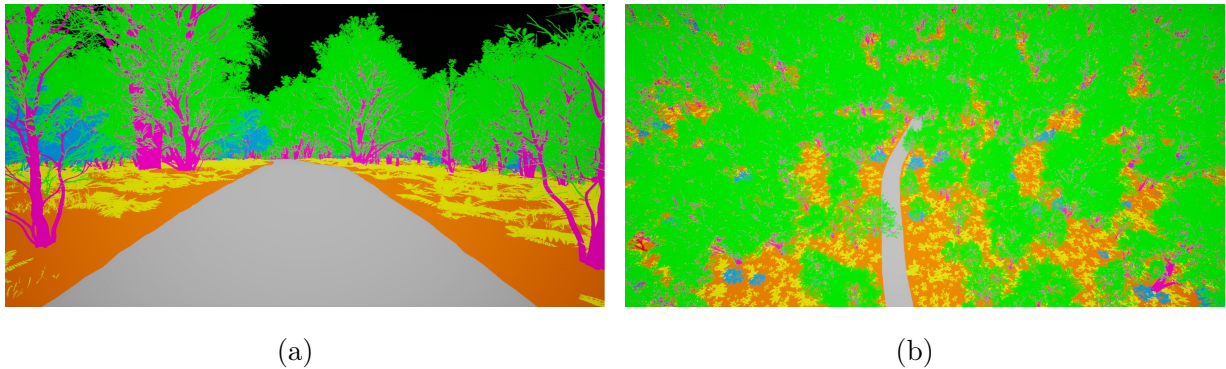


Figure 4.11: Semantic segmentation map captured from the ground perspective (a) and the aerial perspective (b).

### 4.6.3 Depth Maps

Depth maps provide information related to the distance or depth of objects the capture viewpoint. Unlike the RGB images, which capture the scene's color information, depth maps represent distance values to each point of the scene. Following common practice, the

depth maps present a color grading metric, where lighter regions indicate objects that are closer, while darker regions represent objects that are farther away. Depth maps provide the additional spatial information related to the scene geometry, not achievable with the previous data types.

To generate this type of information, the system employs the use of a post-process volume with an post-process material constructed for the purpose. This material captures the scene depth from the center point of the camera to a maximum distance of a hundred meters and uses the interpolation of the colors white and black with an interpolation factor defined by Equation 4.1, where  $I_f$  represents the interpolation factor and  $d$  represents the distance in centimeters. The behavior of this function provides a more precise depth evaluation to objects closer to the camera and offers better definition and distinction between objects.

$$I_f(d) = 1 - v(d) \quad \wedge \quad v(d) = \begin{cases} 0 & , e^{-10d} \leq 0 \\ e^{-10d} & , 0 < e^{-10d} \leq 1 \quad \forall d \in [0, 10000] \\ 1 & , e^{-10d} \geq 1 \end{cases} \quad (4.1)$$

The depth map is captured with same 2D scene capture components used in the other two type of data. The effect of the depth map is applied to the scene by activating the post-process volume setting of infinite extent. When the capture is completed, the infinite extension setting of the post-process volume is deactivated. In Figure 4.12 can be observed the depth map captured by the available perspectives, related to Figures 4.10 and 4.11.

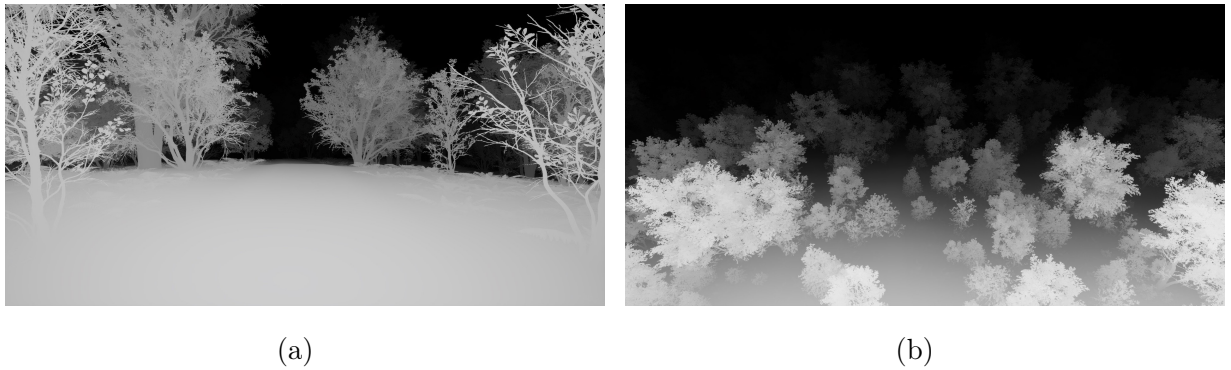


Figure 4.12: Depth map captured from the ground perspective (a) and the aerial perspective (b).

#### 4.6.4 Capture Methods

As seen in Section 4.5.3, the system offers three capture options, called sequences, of which two are fully automated. The sequence "Auto Pilot" is only available when the generation includes the path component and enables the capture of multiple images along the path. This sequence performs three captures at each path point path with rotations of  $-45^\circ$ ,  $0^\circ$  and  $45^\circ$ . The path has 230 possible locations, resulting in 690 captures, i.e., 2070 files per map. The sequence by the name "Random Capture", by default, performs one hundred random placements of the camera throughout the map and performs a capture at each point. The number of random captures can be edited in the custom generation menu. The last sequence, the "Custom Capture", give the user full control and allows the capture of the perspective than is currently visible. To assure that the data stays consistent in all three captures, the player controls are disabled during collection. All captures are saved, in PNG format (.png), in a folder, named with the world seed, and each generated image is identified with the type of data and the number of the capture following the naming convention present in Figure 4.13.

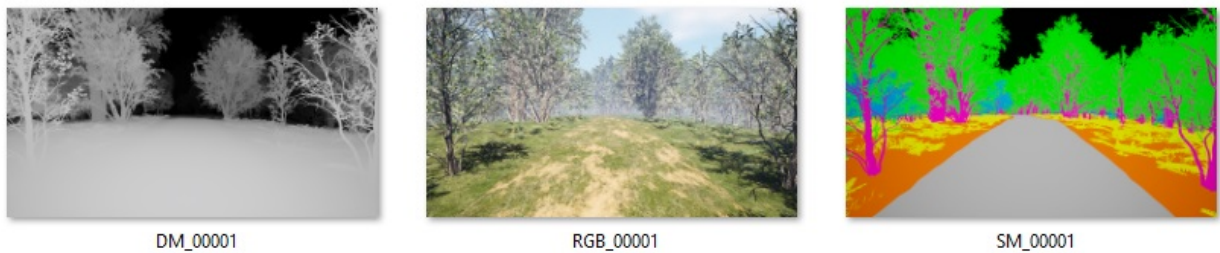


Figure 4.13: Naming convention of each type of capture performed.

# 5 Results and Discussion

The current Chapter encompasses the evaluation of the developed system as a ML tool for the generation of synthetic datasets. The quality of the generated data is assessed through visual inspection and compared with data from different sources. In light of this, a discussion framed by a qualitative and quantitative analysis, as well as visual outcomes for all forms of created data, will be provided, along with the difficulties raised by this type of analysis.

## 5.1 Performance Analysis

The system developed meets the requirements of executing in real-time and enables users with navigational abilities, as well as begin a full-fledged software ready for public release. However, these reasons are not sufficient to confirm the effectiveness of the system, and further evaluations, especially of the performance of the system and the results obtained, are needed. All tests were performed using the same computer used for development. The characteristics of this computer are as follows:

- **Processor:** Intel<sup>®</sup> Core<sup>™</sup> i9-13980HX 2.20Ghz
- **GPU:** Nvidia<sup>®</sup> GeForce RTX<sup>™</sup> 4060 2.37Ghz 8Gb
- **RAM:** 32Gb
- **OS:** Windows 11 version 22H2

### 5.1.1 Generation Evaluation

Assessing the system's generation performance means analyzing the time required to completely render a fully generated environment. The system generation capabilities were evaluated using different parameters of world generation. Each experimentation measures

the time required for the complete world generation, starting at the beginning of generation and ending at the fully rendered scene. All parameters had their default value except the one in analysis.

The first parameter assessed related to the generation was the vegetation density. The influence of this parameter over the generation time is present in Table 5.1.

Vegetation Density	1%	5% <sub>(default)</sub>	10%	20%	25%
Time (s)	18.102	16.428	15.237	16.901	15.006

Table 5.1: Values of vegetation density and respective generation times in seconds.

The results show that the required generation time, contrary to the expected, reduce with the increase in density. This is as a result of the logic used for the object placement, which is mentioned in Section 4.1.2. Increasing tree density reduces the density of ground vegetation and consequently the number of objects placed, accelerating the generation process.

The second generation parameter evaluated was the maximum terrain height, which controls the terrain elevation, and the generation time related to the various values tested are present in Table 5.2. The observed generation times do not present significant variation and indicate that the height value does not interfere with the generation process.

Max. Terrain Height (cm)	5	50	100	150 <sub>(default)</sub>	250	350
Time (s)	16.225	16.637	16.558	16.428	16.477	16.387

Table 5.2: Values of maximum terrain height and respective generation times in seconds.

Evaluating the last two parameters related to the generation process, the illumination conditions and the path, resulted in the generation times present in Tables 5.3 and 5.4, respectively.

Time of Day	Day <sub>(default)</sub>	Dusk	Night
Time (s)	16.429	16.874	16.703

Table 5.3: Illumination conditions and respective generation times in seconds.

As observed, the illumination conditions do not affect the generation time of the scenario. On the other hand, the inclusion of the path expedites the generation process. As mentioned in Section 4.4.2, generating an environment with path invalidates the positioning of numerous objects and reduces the number of objects placed at generation.



Path Included	Yes <sub>(default)</sub>	No
Time (s)	16.429	18.874

Table 5.4: Generation with and without path inclusion and respective generation times in seconds.

The results show that the system’s main drawback is the density of the ground vegetation, i.e., the quantity of objects positioned. In fact, the generation time was decreased to 6.338 seconds as a result of the complete exclusion of ground vegetation placement.

### 5.1.2 Capture Evaluation

Evaluating the system’s capture performance requires the measurement of the time required to fully capture the data for one frame, which includes a RGB image, a semantic segmentation map and a depth map. The system ability to capture data was evaluated using parameters related to the extraction of data and to the world generation. Each experimentation measures the time required for the complete collection of a set of data, starting at the loss of camera control and ending at the enabling of movement. All parameters had their default value except the one in analysis.

The first condition studied was related to the world generation. Various values of vegetation density were tested and their influence on capture time were registered. The values of each iteration can be observed in Table 5.5.

Vegetation Density	1%	5% <sub>(default)</sub>	10%	20%	25%
Time (s)	36.103	35.537	35.528	35.296	34.805

Table 5.5: Values of vegetation density and respective data collection times in seconds.

The presence of fluctuation in capture time was expected, based on the findings related to Table 5.1. The reduction of the number of the objects placed in the scene speed up the capture process. However, the vegetation density parameter does not affect this process the way it does to the generation.

The second and last parameter that impacts the capture process is the output resolution. The four resolution options were tested and registered the following capture times.

As seen in Table 5.6, the resolution chosen has little impact in the capture time due to the fact that UE5 has export functionalities that are executed in parallel.

Capture Resolution	720x480	1280x720	1920x1080 <sub>(default)</sub>	2560x1440
Time(s)	34.783	36.127	35.823	36.836

Table 5.6: Output resolution options and respective data collection times in seconds.

For the purpose of documentation, various amounts of frames were captured and the time required per set was registered. The captures were made with the default configurations. The elapsed times are present in Table 5.7.

Number of Captures	1	5	10	50	100
Time (s)	35.622	181.121	374.249	1890.315	3810.781
Approximate Time	$\simeq 35s$	$\simeq 3min1s$	$\simeq 6min14s$	$\simeq 31min30s$	$\simeq 63min50s$

Table 5.7: Number of captures and respective data collection times.

The testing of the "Auto Pilot" sequence was only performed for a small fraction of the generated path, however is estimated that the full path is executed in approximately in seven hours, resulting in 2070 files of data.

An additional experiment was conducted to evaluate the capture time required for extracting only one type of data at a time. The measurement was repeated multiple times and the average times recorded for each type of data can be seen in Table 5.8.

Type of Data	RGB Image	Semantic Segmentation Map	Depth Map
Average Time (ms)	459.5	387.333	418.8

Table 5.8: Average capture time to extract one frame of a specific data type.

This test assesses the real-time capabilities of the system and confirms its ability to generate a consistent output of two frames per second, given that it only generates one type of data. This output, for instance, can be utilized for simulation purposes. These results lead to another extra evaluation, the measurement of the time that the system requires to perform the switch between data types. The obtained result are present in Table 5.9.

Through observation, the switch between RGB data and Semantic Segmentation data is the limiting factor of the the time that is required to perform a complete frame extraction. Due to the importance of the system's navigational aspect, the collection of data is performed sequentially for each frame. However, a fully autonomous version of the SynPhoRest system, let's call it SynPhoRest v2.0: Hands-Off Edition, where there is no navigational implemen-

Type of Data Switch	RGB to SS	SS to RGB	RGB to Depth	Depth to RGB
Average Time (s)	8.820	8.913	0.545	0.528

Table 5.9: Average switch time from one data type to another.

tation could benefit of a reorganization in the capture process. Performing all captures of one data type at a time and then performing the switch in data type could, theoretically, decrease the time required to perform the full path extraction, i.e., 690 sets of data or 2070 files, in just under fifteen minutes. A preliminary test was conducted to evaluate a crude implementation of this reformulation. The experiment was executed for 60 captures of each data type and resulted in an elapsed time of seven minutes and thirty four second. Due to the rough nature of this implementation, every third frame was not correctly generated, concluding that this reformulation requires further development.

## 5.2 Extracted Data Analysis

The system meets the requirements related to synthetic data generation, producing RGB images and their respective semantic segmentation and depth maps. The evaluation process related to the quality of the synthetic data is a little challenging as a consequence of the format of the captures. Images can only be evaluated through visual inspection and therefore can not be classified so accurately. In spite of that, it can be concluded through observation that the environments generated achieve a considerable level of photorealism. The collection of selected assets and their disposition closely mimic the natural world and, together with the illumination choices, reinforce the feeling of a real forest. Various example sets of the extracted data are included in appendix A.

The extracted semantic segmentation maps, as described in Section 4.6.2, achieve pixel precision granted by the material switching method used. Confirming the accuracy achieved, using the popular image editor software Adobe Photoshop [77], version 13.1.2, the RGB image and the semantic segmentation map were layered, with the second having its opacity reduced. The results of this experiment can be seen in Figures 5.2, 5.3, 5.4 and 5.5.

The generated depth maps exhibit a remarkable level of precision attributable to the feature described in Section 4.6.3. The virtual nature of the environment enables an exact measurement of the distance to each pixel, and in conjunction with the integrated scene depth readings provided by UE5, allow the construction of a precise representation of the scene's

depth. Generally, depth maps are created using two passive cameras with slightly different viewpoints of a scene using stereo matching. As a result, it is possible to compute pixel-level disparities and estimate scene depth using the processing of related features between the images, yielding only approximations. The depth maps provided by the developed system are comparatively superior and offer a true reading of the scene’s depth. Figure 5.1 presents the comparison between a common stereo depth map, a depth map generated with the algorithm proposed by A. Aslam et al. [78] and the depth map extracted by SynPhoRest v2.0. Moreover, the system generates depth maps with resolutions that exceed the resolution of the depth cameras currently on the market, offering larger amounts of data and with better precision.

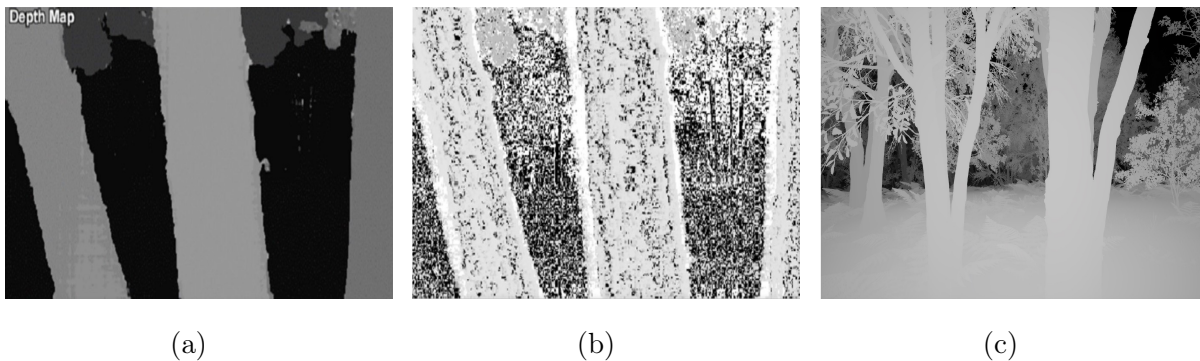


Figure 5.1: Comparison between common stereo depth maps (a), "Aslam" algorithm (b) and the depth maps generated by SynPhoRest v2.0 (c). Adapted from [78].

Performing a quantitative analysis, the system is able to produce six hundred and ninety fully automated sets of data, totaling in two thousand and seventy files. Additionally, the possibility of random and custom capture add to the total amount of files per environment. Each world generation has a corresponding ten-digit integer seed, as indicated in Section 4.4.2, which equates to the possibility of producing over four billion unique environments, even without accounting for the many adjustable generational parameters. Considering this information, theoretically, the system has the ability to produce more than two trillion unique frames.

Overall, the SynPhoRest v2.0 system performs as expected, generating random environments procedurally and extracting correct and precise data in all three forms. As a ready-to-ship software, SynPhoRest is a tool that tackles the existing gap in the field of non-urban navigation DL algorithms. Having accurate labelling capabilities and true depth maps, SynPhoRest is a great step forward for ML research and has the potential to reinforce existing datasets, boosting models performances. With the growing desire to relinquish

manual hand-labeling methods in favor of automated techniques, SynPhoRest provides the perfect training data for algorithms developed for labeling real-world data. Nonetheless, synthetic data differs from real data and without proper experimentation, it is impossible to determine the efficiency of the generated data. Moreover, as seen in Chapter 2, datasets exclusively build with artificial data tend to offer negative results and synthetic data is only used as an augmentation tool. Additionally, a fully synthetic dataset generated with SynPhoRest can be used for training purposes, in researches related to the construction of automatic labeling tools.

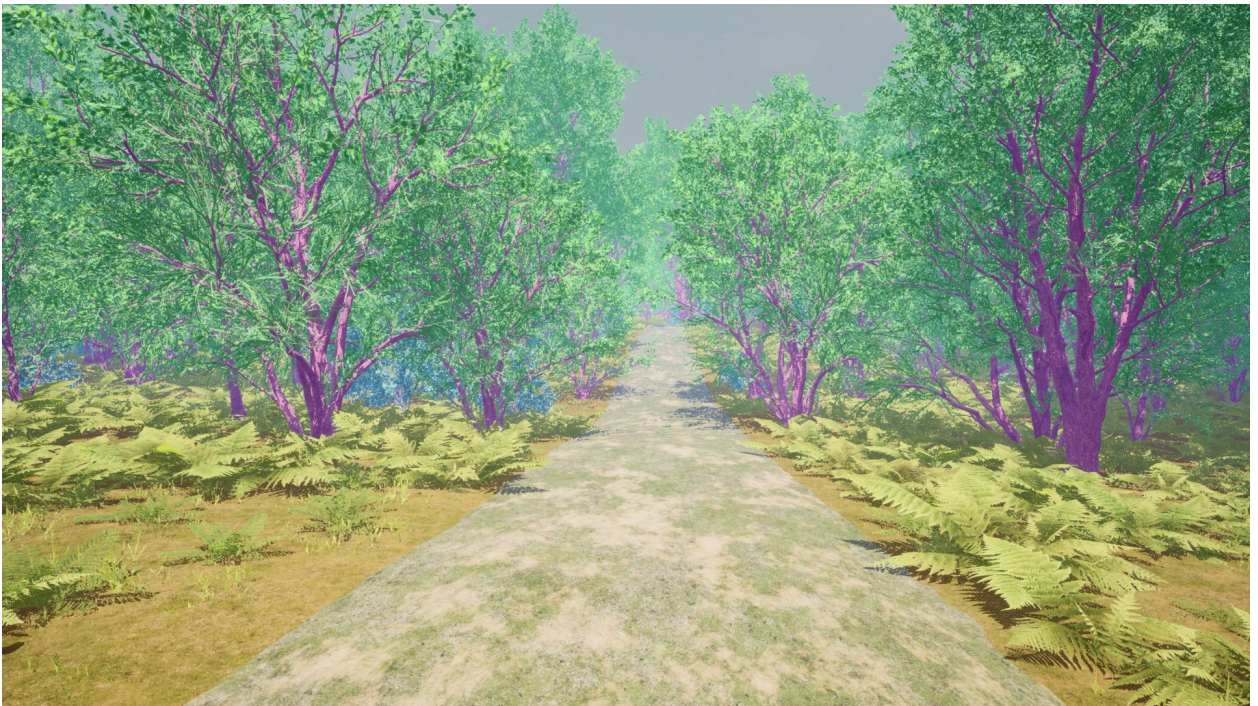


Figure 5.2: Example of semantic segmentation maps precision verification. Layering of the RGB and semantic segmentation map images, with opacity at 100% and 25%, respectively.



Figure 5.3: Example of semantic segmentation maps precision verification. Layering of the RGB and semantic segmentation map images, with opacity at 100% and 25%, respectively.

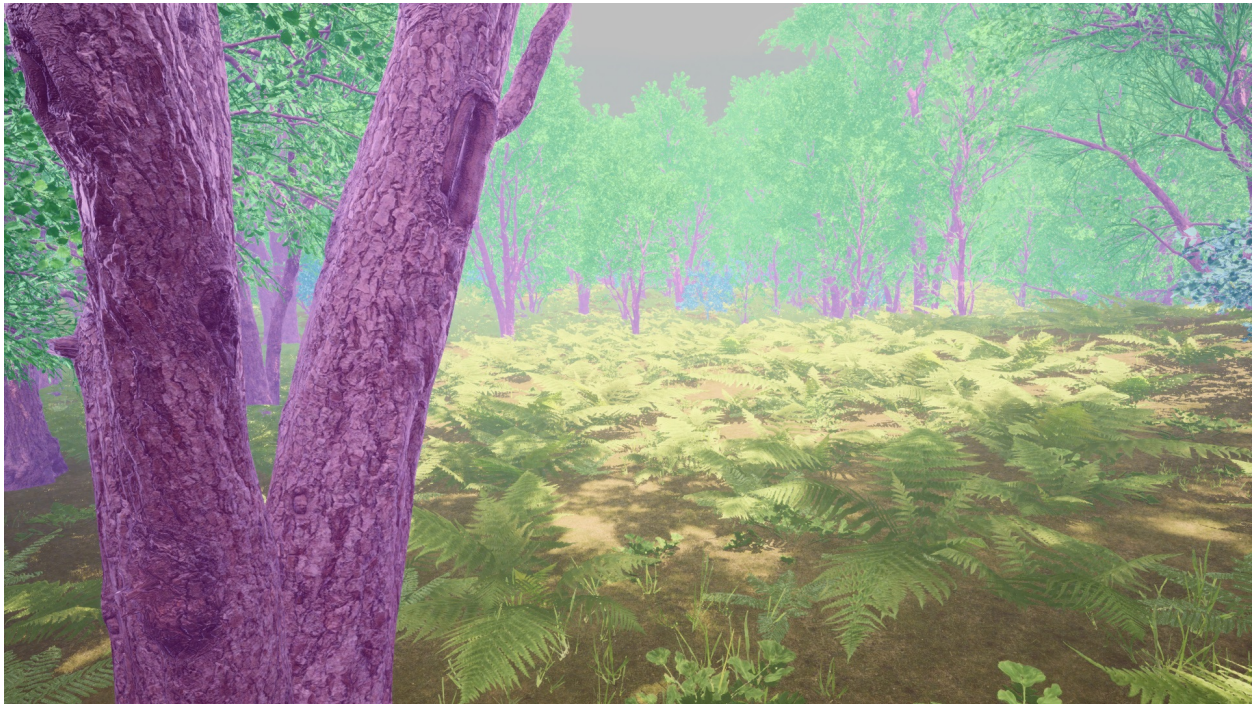


Figure 5.4: Example of semantic segmentation maps precision verification. Layering of the RGB and semantic segmentation map images, with opacity at 100% and 25%, respectively.

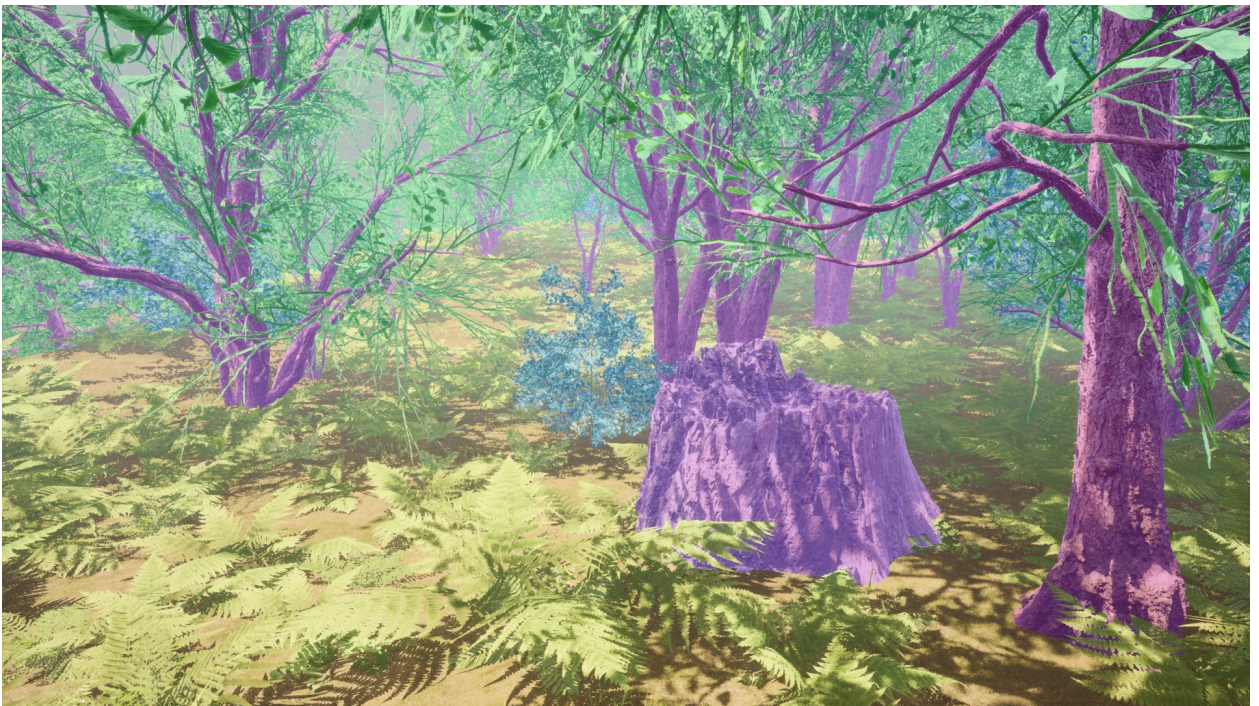


Figure 5.5: Example of semantic segmentation maps precision verification. Layering of the RGB and semantic segmentation map images, with opacity at 100% and 25%, respectively.

## 6 Conclusion

The purposed solution of this master thesis aimed to develop a comprehensive system built entirely in UE5 for generating synthetic data from a procedurally generated virtual environment. The system successfully achieved its objectives by providing a range of relevant data types, that can be utilized for various ML and DL applications.

The development process emphasised the creation of a versatile tool for data collection. The system incorporates three data extraction methods and various generation parameters, ensuring flexibility and adaptability to different research and application scenarios. Researchers and developers can leverage this system to create customized datasets that meet specific requirements, enabling them to explore and train machine learning models with diverse virtual forest data.

By utilizing UE5's advanced graphics rendering capabilities, the system effectively captures the intricate details of a virtual forest environment. The generated synthetic data closely resembles real-world forest imagery, enabling more accurate and reliable training of classification or autonomous navigation algorithms. This approach provides a cost-effective alternative to acquiring large-scale annotated datasets. Moreover, the system's procedural generation techniques allow for the creation varied virtual forest environments, enhancing the generalization capabilities of machine learning models and enabling better performances.

In conclusion, the developed system represents a valuable contribution to the field of ML and DL applications concerning forest environments. Research teams can utilize this system to generate high-quality synthetic data for training, testing and validating their algorithms. The work developed lays the foundation for future advancements in the generation of synthetic data and its application to machine learning tasks in diverse environments. It is anticipated that this work will inspire further research and development in the field, leading to improved algorithms, better training methodologies and more accurate computer vision systems.



## 6.1 Future Work

While SynPhoRest v2.0 has achieved its primary objectives in developing a system for generating synthetic data from a virtual forest environment, there are several opportunities for further development that can enhance its capabilities. Firstly, a collaboration project with multidisciplinary team can vastly increase the authenticity of the generated environment. The inclusion of a botanist and a geologist would enable the identification of the correct vegetation and terrain composition. Furthermore, the addition of a 3D artist would allow for an accurate recreation of the forest components and increase the diversity of the selected assets used in the development of this system.

Related to performance, as observed in Section 5.1.2, the system could be reorganized and optimized to improve extraction times. Additional development of the chunk manager module could overcome the difficulties faced and expand the number of captures per generation exponentially.

Data is always a necessity and the inclusion of more sensors leads to more data. New sensors would enable the construction of more complete datasets. The example of LIDAR sensors come to mind. This type of extraction could be performed with two virtual cameras in a fronto-parallel configuration and the addition of several acquisition patterns would allow the inclusion of point cloud data on the generated datasets. Another possible feature is the generation of a second depth map. Compared to common depth maps, the depth maps generated by SynPhoRest offer an exact representation of the environment. The recreation of a less accurate depth map could be achieved using two fronto-parallel cameras and the technique of stereo matching, producing data more similar to real data depth maps.

For the purpose of simulation and using the real-time data presented in Section 5.1.2, an API could be developed to bridge SynPhoRest with a Robot Operating System (ROS) implementation for algorithm testing. This API would perform the communication between both systems, sending RGB images to the algorithm and translate ROS commands to virtual navigational controls.

During development of the system, some testing to the procedural generation was performed. These test allowed the detection of a generational error, present in Figure 6.1. The issue was documented and the problem was addressed. However, this event incited an idea for a possible generation implementation. By applying a water material to the plane that appeared through the terrain and using an appropriate second noise generator, the topography could be manipulated to create riverbeds and increase forest authenticity.



Figure 6.1: Generation error detected during testing.

## 6.2 Personal Considerations

On a personal note, the year-long development of SynPhoRest v2.0 was really important to my personal growth. It gave me the chance to hone skills I'd developed during my academic journey and reaffirmed the idea that the knowledge collected in the academy lays the groundwork for all future endeavours. This experience has been very rewarding and allowed for an interesting introduction to the field of computer graphics.

## 7 Bibliography

- [1] Fortune Business Insights. Machine Learning (ML) Market Size, Share & COVID-19 Impact Analysis. <https://www.fortunebusinessinsights.com/machine-learning-market-102226>. Accessed: 2023-01-23.
- [2] David Schubmehl. Accelerating the Development of Machine Learning Applications. <https://partners.wsj.com/aws/accelerating-development-machine-learning-applications/>. Accessed: 2023-01-23.
- [3] Niall O’Mahony, Sean Campbell, Anderson Carvalho, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Point Cloud Annotation Methods for 3D Deep Learning. pages 1–6. DOI: 10.1109/ICST46873.2019.9047730.
- [4] Theophano Mitsa. How do you know you have enough training data? <https://towardsdatascience.com/how-do-you-know-you-have-enough-training-data-ad9b1fd679ee>. Accessed: 2023-06-23.
- [5] Food and Agriculture Organization of the United Nations. A fresh perspective - Global Forest Resources Assessment 2020. <https://www.fao.org/forest-resources-assessment/2020/en/>. Accessed: 2023-07-05.
- [6] Divisão de Gestão do Programa de Fogos Rurais. 8.º RELATÓRIO PROVISÓRIO DE INCÊNDIOS RURAIS. <https://www.icnf.pt/api/file/doc/4e8a66514175d0f7>. Accessed: 2023-01-23.
- [7] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2015. 10.1109/LRA.2015.2509024.

- [8] Linus Lind. *Deep learning navigation for UGVs on forests paths*. PhD thesis, KTH ROYAL INSTITUTE OF TECHNOLOGY SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE, Sweden, March 2018. <https://www.semanticscholar.org/paper/Deep-learning-navigation-for-UGVs-on-forests-paths-Lind/2d15e12411f07c86e5db23f3dfc248914a97396f>.
- [9] Aiswarya Munappy, Jan Bosch, Helena Holmström Olsson, Anders Arpteg, and Björn Brinne. Data management challenges for deep learning. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 140–147, 2019. DOI: 10.1109/SEAA.2019.00030.
- [10] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds, May 2020. arXiv:1911.11236 [cs.CV].
- [11] V7Labs. V7 Darwin - Auto-Annotate Complex Objects 10x Faster. <https://www.v7labs.com/auto-annotation>. Accessed: 2023-07-04.
- [12] Synced. Data Annotation: The Billion Dollar Business Behind AI Breakthroughs. <https://syncedreview.com/2019/08/28/data-annotation-the-billion-dollar-business-behind-ai-breakthroughs/>. Accessed: 2023-01-23.
- [13] Tyler Ganter. I performed Error Analysis on Google’s Open Images dataset and now I have trust issues. <https://towardsdatascience.com/i-performed-error-analysis-on-open-images-and-now-i-have-trust-issues-89080e03ba09>. Accessed: 2023-01-23.
- [14] Zhengxia Zou, Keyan Chen, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object Detection in 20 Years: A Survey. *Proceedings of the IEEE*, 111(3):257–276, 2023. DOI: 10.1109/JPROC.2023.3238524.
- [15] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollar. Panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. arXiv:1801.00868 [cs.CV].
- [16] Mostafa Mahmoud Ibrahim, Qiong Liu, Rizwan Khan, Jingyu Yang, Ehsan Adeli, and You Yang. Depth map artefacts reduction: A review. *IET Image Processing*, 14(12):2630–2644, 2020. DOI: 10.1049/iet-ipr.2019.1622.

- [17] Hyung-Min Jeon, Long Hoang Pham, Duong Nguyen-Ngoc Tran, Huy-Hung Nguyen, and Jae Wook Jeon. 3d synthetic image training and testing data for autonomous driving. In *2022 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pages 1–4. IEEE. DOI: 10.1109/ICCE-Asia57006.2022.9954858.
- [18] Larisa Zherdeva, Evgeniy Minaev, Denis Zherdev, and Vladimir Fursov. Synthetic dataset for navigation tasks of autonomous systems and ground robots. In *2021 International Conference on Information Technology and Nanotechnology (ITNT)*, pages 1–4. IEEE. DOI: 10.1109/ITNT52450.2021.9649285.
- [19] Rui Nunes, João Ferreira, and Paulo Peixoto. Procedural generation of synthetic forest environments to train machine learning algorithms. <https://openreview.net/pdf?id=rpzgjNCe4G9>.
- [20] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949. DOI: 10.1080/01621459.1949.10483310.
- [21] Sergey I. Nikolenko. *The Early Days of Synthetic Data*, pages 139–159. Springer International Publishing, Cham, 2021. DOI: 10.1007/978-3-030-75178-4\_5.
- [22] Donald B Rubin. Statistical disclosure limitation. *Journal of official Statistics*, 9(2):461–468, 1993. <https://www.scb.se/contentassets/ca21efb41fee47d293bbee5bf7be7fb3/discussion-statistical-disclosure-limitation2.pdf> Accessed: 2023-07-10.
- [23] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data Augmentation Generative Adversarial Networks. March 2018. arXiv:1711.04340 [stat,ML].
- [24] Gabriel J. Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, January 2009. DOI: 10.1016/j.patrec.2008.04.005.
- [25] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. arXiv:1604.01685 [cs.CV].

- [26] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018. arXiv:1802.02611 [cs.CV].
- [27] Guosheng Lin, Chunhua Shen, Anton van den Hengel, and Ian Reid. Efficient piecewise training of deep structured models for semantic segmentation, 2016. arXiv:1504.01013 [cs.CV].
- [28] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions, 2016. arXiv:1511.07122 [cs.CV].
- [29] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for Data: Ground Truth from Computer Games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, volume 9906, pages 102–118. Springer International Publishing, Cham, 2016. DOI: 10.1007/978-3-319-46475-6\_7.
- [30] Xiangyu Yue, Bichen Wu, Sanjit A. Seshia, Kurt Keutzer, and Alberto L. Sangiovanni-Vincentelli. A LiDAR Point Cloud Generator: from a Virtual World to Autonomous Driving, March 2018. arXiv:1804.00103 [cs.CV].
- [31] aitorzip. DeepGTAV plugin. <https://github.com/aitorzip/DeepGTAV>, 2017. Accessed: 2023-06-20; Commit: b1dc8759300c6651706a0035a88d53d4cfd60b43.
- [32] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1887–1893, 2018. DOI: 10.1109/ICRA.2018.8462926.
- [33] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012. DOI: 10.1109/CVPR.2012.6248074.
- [34] Braden Hurl, Krzysztof Czarnecki, and Steven Waslander. Precise synthetic image and lidar (presil) dataset for autonomous vehicle perception. pages 2522–2529, 2019. DOI: 10.1109/IVS.2019.8813809.
- [35] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on*

- Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 13–15 Nov 2017. <http://proceedings.mlr.press/v78/dosovitskiy17a/dosovitskiy17a.pdf> Accessed: 2023-07-10.
- [36] David Griffiths and Jan Boehm. Synthcity: A large scale synthetic point cloud, 2019. arXiv:1907.04758 [cs.CV].
- [37] Blender Foundation. Blender. <https://www.blender.org/>. Accessed: 2023-06-30.
- [38] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. Blensor: Blender sensor simulation toolbox. In *Advances in Visual Computing: 7th International Symposium, ISVC 2011, Las Vegas, NV, USA, September 26-28, 2011. Proceedings, Part II 7*, pages 199–208. Springer, 2011. DOI: 10.1007/978-3-642-24031-7\_20.
- [39] Ishaan Paranjape, Abdul Jawad, Yanwen Xu, Asiih Song, and Jim Whitehead. A modular architecture for procedural generation of towns, intersections and scenarios for testing autonomous vehicles. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 162–168, 2020. DOI: 10.1109/IV47402.2020.9304625.
- [40] Asiih Song and Jim Whitehead. Townsim: agent-based city evolution for naturalistic road network generation. pages 1–9, 08 2019. DOI: 10.1145/3337722.3341852.
- [41] Augmented Design Lab. Intgen. <https://github.com/AugmentedDesignLab/intgen>, 2019. Accessed: 2023-06-23; Commit: 1d4f85479a820a78b541ef2da15eab35f3a291e8.
- [42] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. DOI: 10.1109/CVPR.2016.352.
- [43] Magnus Wrenninge and Jonas Unger. Synscapes: A photorealistic synthetic dataset for street scene parsing, 2018. arXiv:1810.08705 [cs.CV].
- [44] Callum Newlands and Klaus-Peter Zauner. Procedural generation and rendering of realistic, navigable forest environments: An open-source tool. Number arXiv:2208.01471. arXiv. arXiv:2208.01471 [cs.GR].
- [45] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. The algorithmic beauty of plants. In *The Virtual Laboratory*, 1990. DOI: 10.1046/j.1469-8137.1997.00655-1.x.

- [46] callumnewlands. Forestgenerator. <https://github.com/callumnewlands/ForestGenerator/>, 2022. Accessed: 2023-06-28; Commit: 2052b1dc0ed1459e996feb0a7b816fd373f784c8.
- [47] Unity Technologies. Unity. <https://www.unity.com/>. Accessed: 2023-06-28.
- [48] Inc. Tesla. Tesla. <https://www.tesla.com/>. Accessed: 2023-06-29.
- [49] Inc. Tesla. Tesla ai day 2021. <https://www.youtube.com/watch?v=j0z4FweCy4M>. Accessed: 2023-06-29.
- [50] Jing Xu, Yu Pan, Xinglin Pan, Steven Hoi, Zhang Yi, and Zenglin Xu. Regnet: Self-regulated network for image classification, 2021. arXiv:2101.00590 [eess.IV].
- [51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. arXiv:1706.03762 [cs.CL].
- [52] YoYo Games. Gamemaker studio. <https://www.gamemaker.io/en>. Accessed: 2023-06-30.
- [53] Juan Linietsky and Ariel Manzur. Godot engine. <https://www.godotengine.org/>. Accessed: 2023-06-30.
- [54] DICE. Frostbite engine. <https://www.ea.com/frostbite>. Accessed: 2023-06-30.
- [55] Epic Games. Unreal engine 5. <https://www.unrealengine.com/en-US/unreal-engine-5>. Accessed: 2023-06-30.
- [56] UGuruz. Unity vs unreal : Graphics comparison (hdrp demo). [https://www.youtube.com/watch?v=sMIi\\_Z33pkw](https://www.youtube.com/watch?v=sMIi_Z33pkw). Accessed: 2023-06-30.
- [57] Pasquale Scionti. My new archviz photoreal scene unreal engine 5 lumen and nanite. <https://www.artstation.com/artwork/xJE2a2>. Accessed: 2023-06-30.
- [58] Wayne Maxwell. Does unreal engines 5 nanite make poly count irrelevant. <https://cgobsession.com/does-unreal-engines-5-nanite-make-poly-count-irrelevant/>. Accessed: 2023-06-30.
- [59] Joon-Seok Kim. Procedural city generation beyond game development. *ACM SIGSPATIAL Special*, 2018. DOI: 10.1145/3292390.3292397.



- [60] Ci-Lovers. Avatar: A technological masterpiece that changed the face of cinema. <https://cilovers.com/avatar-a-technological-masterpiece-that-changed-the-face-of-cinema/>. Accessed: 2023-06-30.
- [61] Theodore McKenzie. Procedural explosion vfx made in houdini nuke. <https://80.lv/articles/procedural-explosion-vfx-made-in-houdini-nuke/>. Accessed: 2023-06-30.
- [62] Mojang. Minecarft. <https://www.minecraft.net/>. Accessed: 2023-06-30.
- [63] Hello Games. No man's sky. <https://www.nomanssky.com/>. Accessed: 2023-06-30.
- [64] Kati Steven Emmanuel, Christian Mathuram, Akshay Rai Priyadarshi, Rishu Abraham George, and J Anitha. A beginners guide to procedural terrain modelling techniques. In *2019 2nd International Conference on Signal Processing and Communication (ICSPPC)*, pages 212–217, 2019. DOI: 10.1109/ICSPPC46172.2019.8976682.
- [65] Ares Lagae, Sylvain Lefebvre, Robert L Cook, Tony DeRose, George Drettakis, David S Ebert, John P Lewis, Ken Perlin, and Matthias Zwicker. State of the art in procedural noise functions. *Eurographics (State of the Art Reports)*, pages 1–19, 2010. DOI: 10.2312/egst.20101059.
- [66] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, jul 1985. DOI: 10.1145/325165.325247.
- [67] Ken Perlin. Noise hardware. <https://redirect.cs.umbc.edu/~olano/s2001c24/ch09.pdf>, 2001. Accessed: 2023-06-30.
- [68] Klayton Kowalski. Simplex noise generator. <https://klaytonkowalski.github.io/bundles/fractal-noise/index.html>. Accessed: 2023-06-31.
- [69] devdad. Simplexnoise plugin. <https://github.com/devdad/SimplexNoise>, 2020. Accessed: 2023-01-24; Commit: b57598706afd8fc4d50164da1ed58515595699b2.
- [70] Welinton A. Contato Tiago S. Nazaré, Gabriel B. Paranhos da Costa and Moacir Ponti. *Deep Convolutional Neural Networks and Noisy Images*, pages 416–424. 02 2018. DOI: 10.1007/978-3-319-75193-1\_50.
- [71] EpicGames. Unreal engine 5.1 documentation - procedural mesh component. <https://docs.unrealengine.com/5.1/enUS/API/Plugins/ProceduralMeshComponent/UProceduralMeshComponent/>, 2022. Accessed: 2023-06-23.

- [72] Quixel Megascans. Megascans trees: European black alder (early access). <https://www.unrealengine.com/marketplace/en-US/product/megascans-trees-european-black-alder-early-access>. Accessed: 2023-07-02.
- [73] Quixel Megascans. Megascans - Meadow Pack. <https://www.unrealengine.com/marketplace/en-US/product/megascans-meadow-pack>. Accessed: 2023-07-02.
- [74] Project Nature. Temperate Vegetation: Fern Collection. <https://www.unrealengine.com/marketplace/en-US/product/temperate-vegetation-fern-collection>. Accessed: 2023-07-02.
- [75] Quixel Megascans. Megascans - Dead Stumps. <https://www.unrealengine.com/marketplace/en-US/product/88cef256585249f084ddf51009de5291>. Accessed: 2023-07-02.
- [76] Quixel Megascans. Megascans - Rocky Grassland. <https://www.unrealengine.com/marketplace/en-US/product/megascans-rocky-grassland>. Accessed: 2023-07-02.
- [77] Adobe Inc. Adobe Photoshop. <https://www.adobe.com/products/photoshop.html>. Accessed: 2023-07-04.
- [78] Asra Aslam and Mohd. Samar Ansari. Depth-map generation using pixel matching in stereoscopic pair of images. 2019. arXiv:1902.03471 [cs.CV].

# Appendix A

## SynPhoRest v2.0

This appendix provides additional information and examples related to the extracted data, generation parameters and libraries of assets related to the work developed for this thesis. It offers a glimpse into the data samples, various generation parameter values and the range of assets employed. These examples serve to enhance understanding and provide visual and contextual references for the project's methodology and resources.

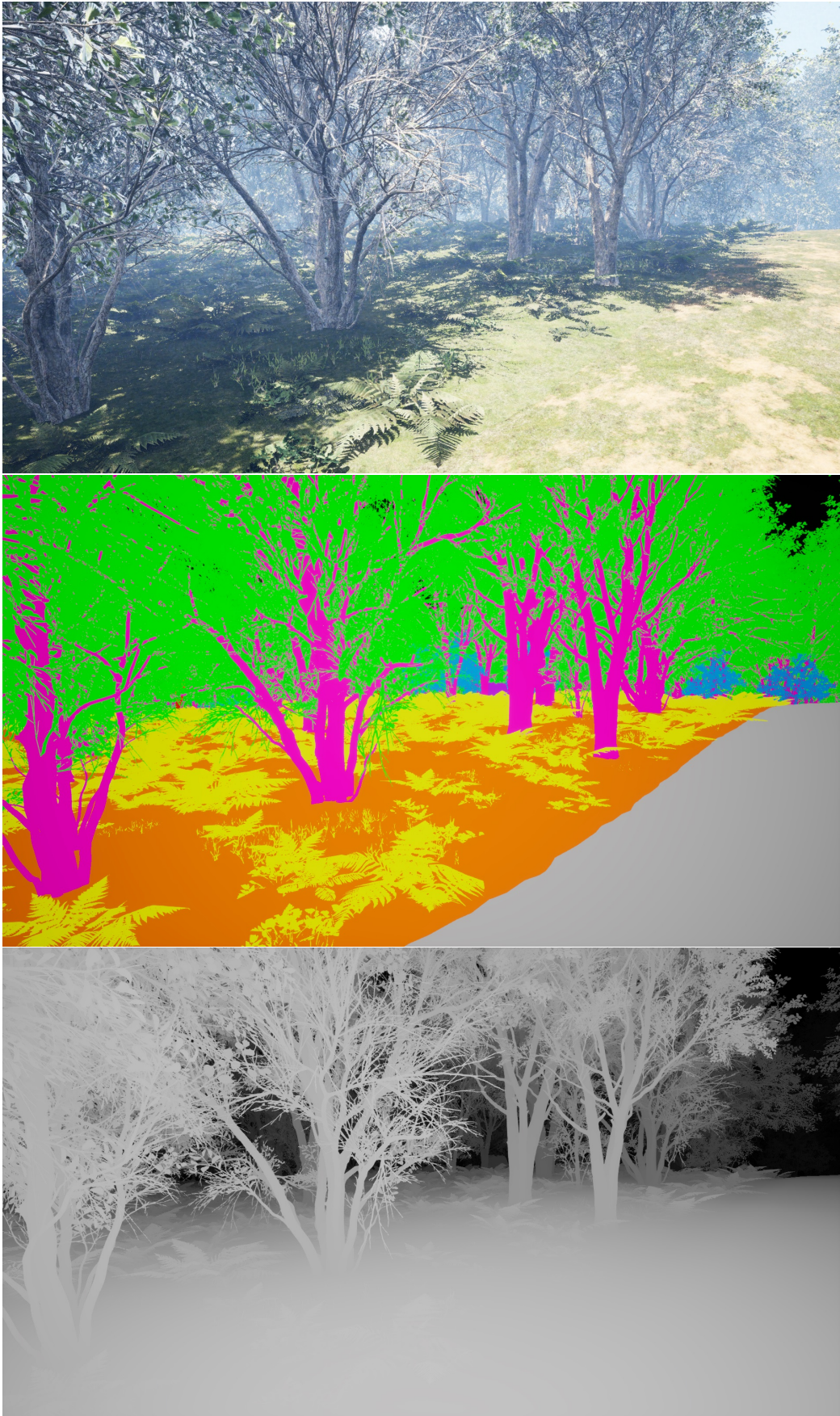


Figure A.1: Set of data extracted from ground perspective, with "Auto Pilot" mode, at  $-45^\circ$ .

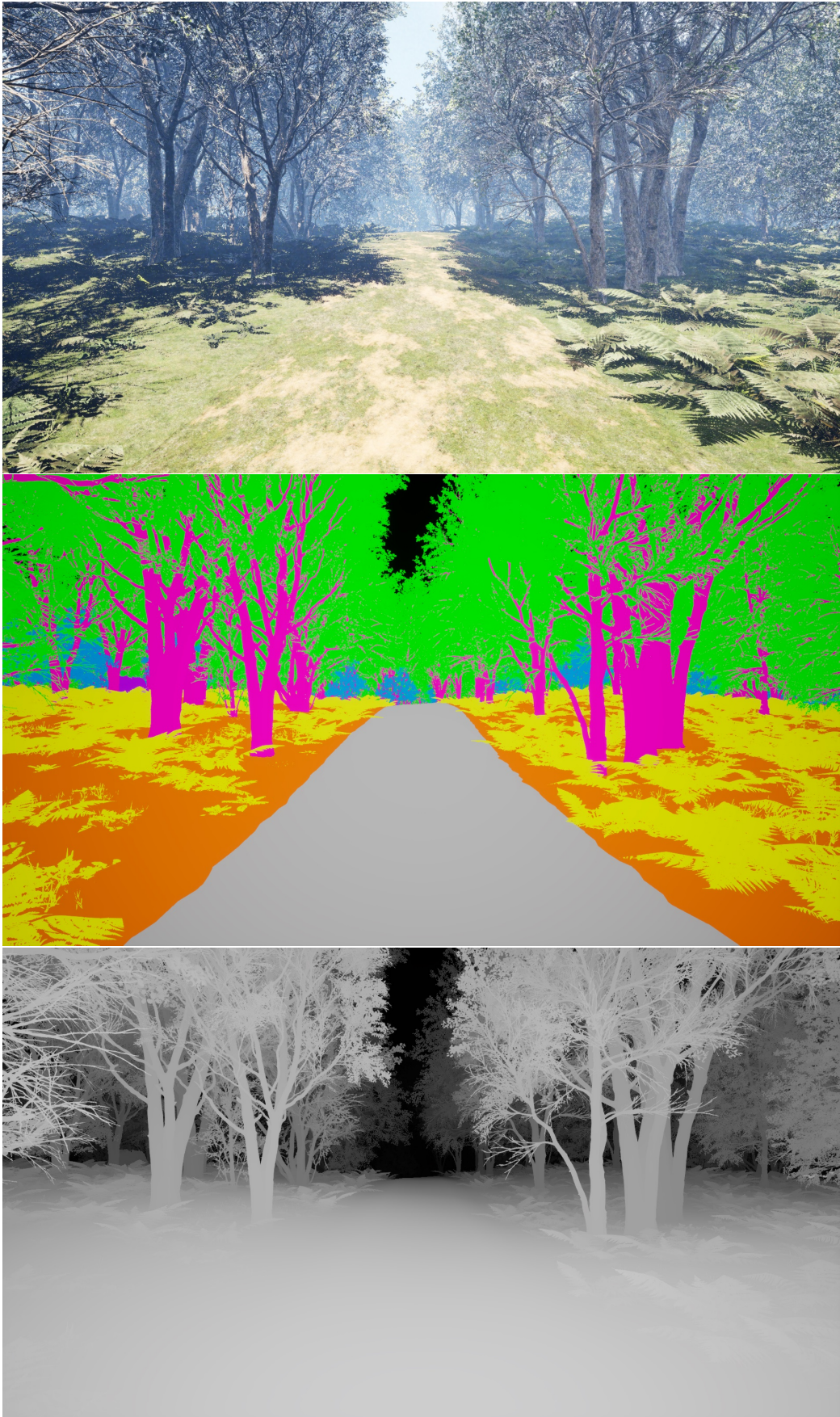


Figure A.2: Set of data extracted from ground perspective, with "Auto Pilot" mode, at  $0^\circ$ .

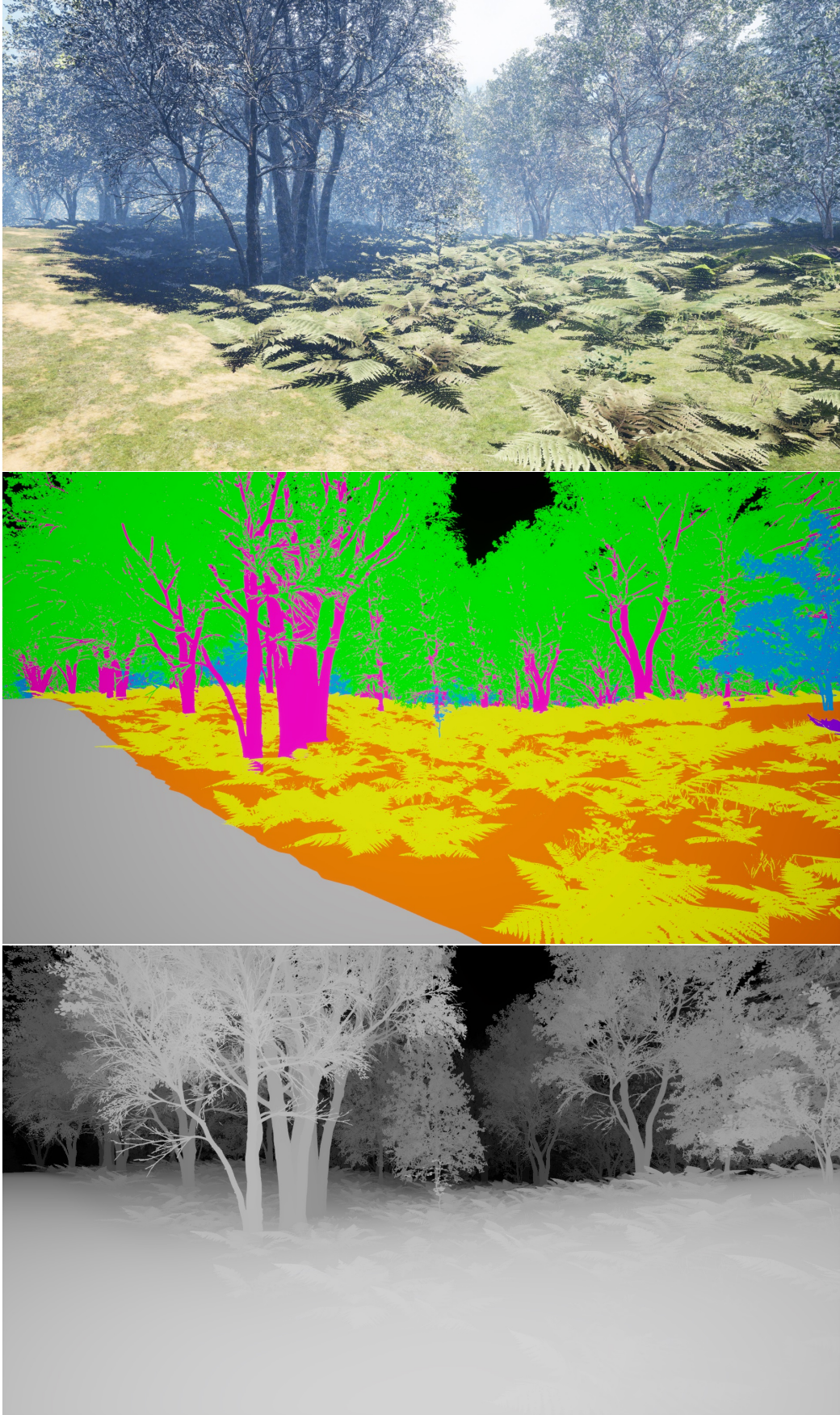


Figure A.3: Set of data extracted from ground perspective, with "Auto Pilot" mode, at 45°.

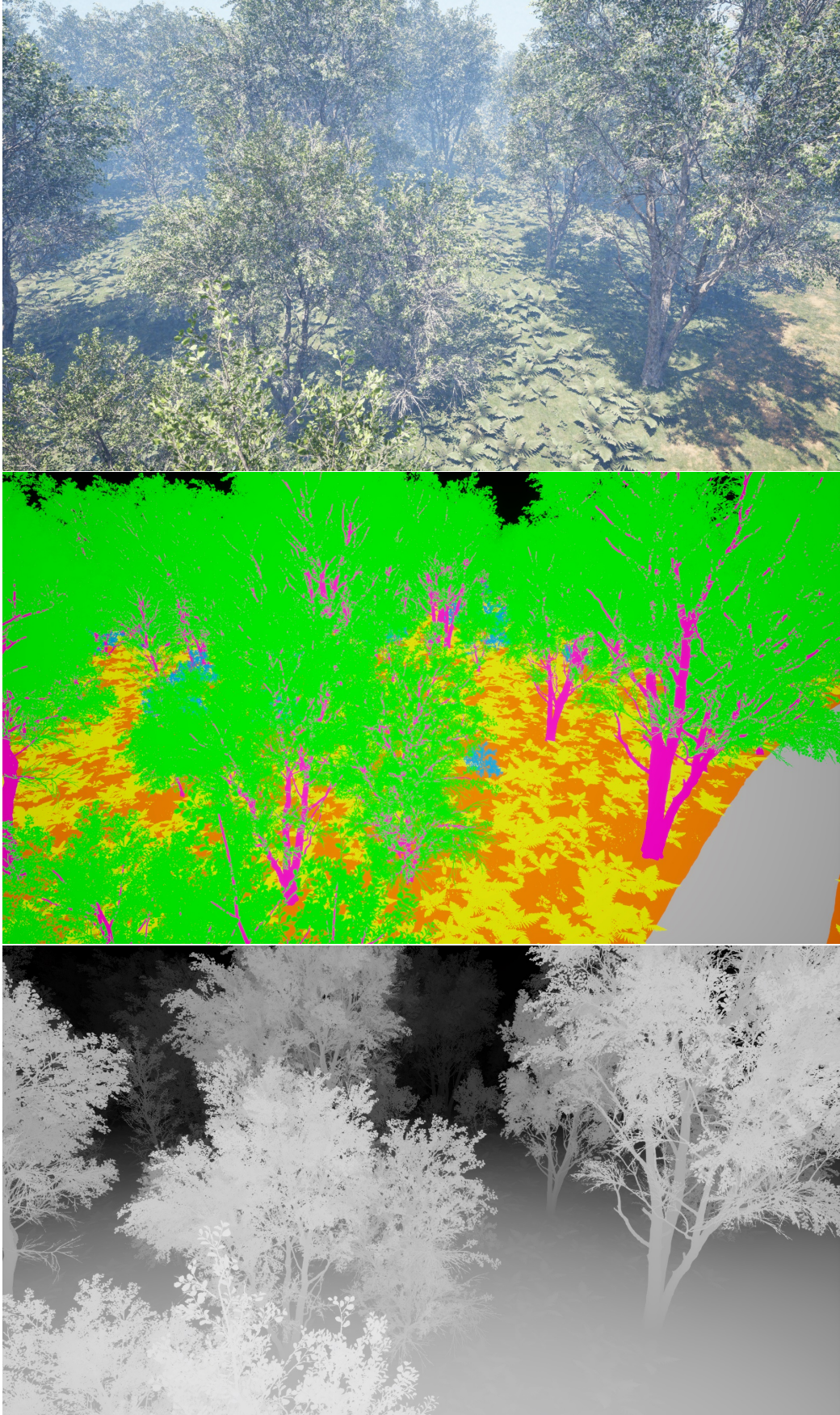


Figure A.4: Set of data extracted from aerial perspective, with "Auto Pilot" mode, at  $-45^\circ$ .

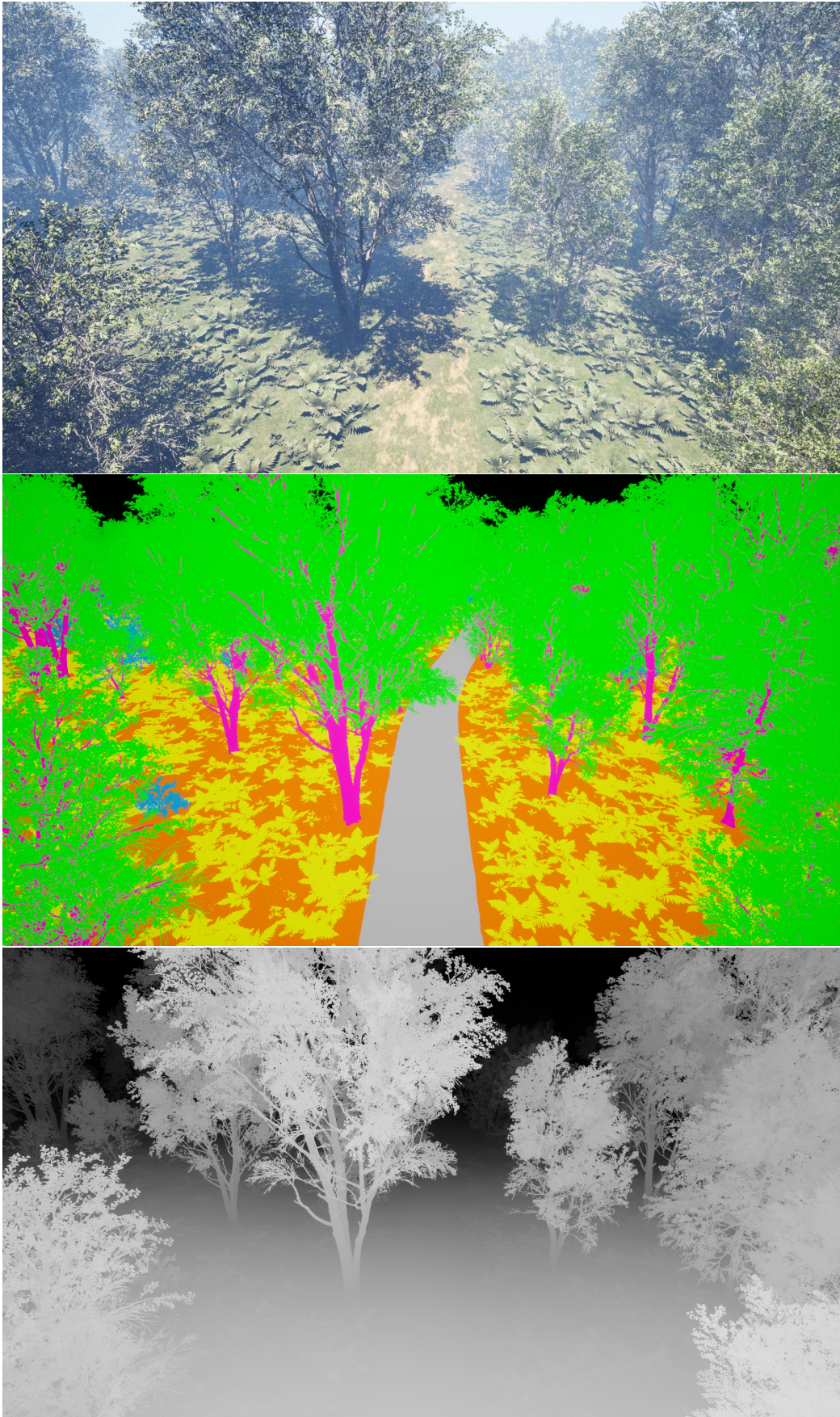


Figure A.5: Set of data extracted from aerial perspective, with "Auto Pilot" mode, at  $0^\circ$ .



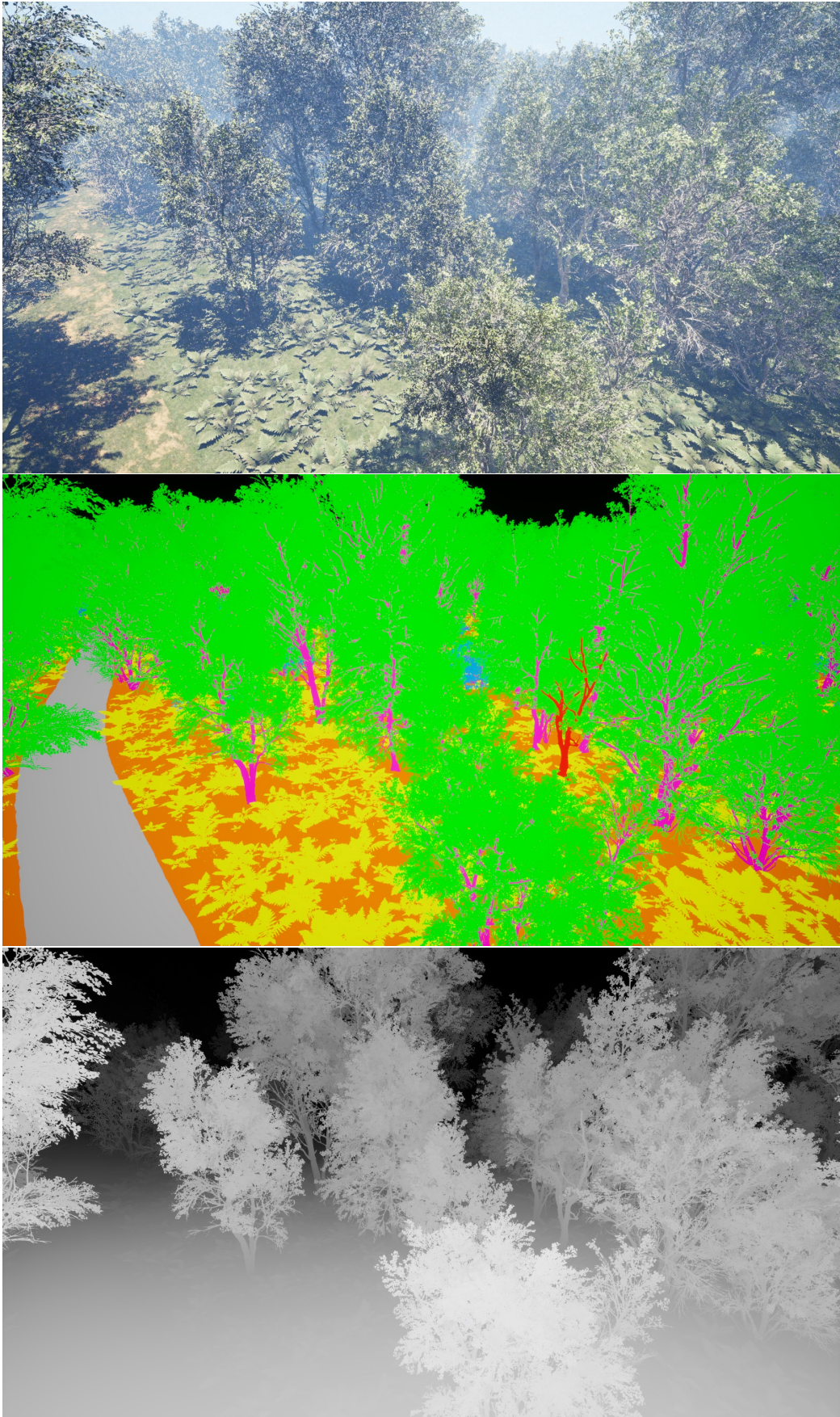


Figure A.6: Set of data extracted from aerial perspective, with "Auto Pilot" mode, at  $45^\circ$ .

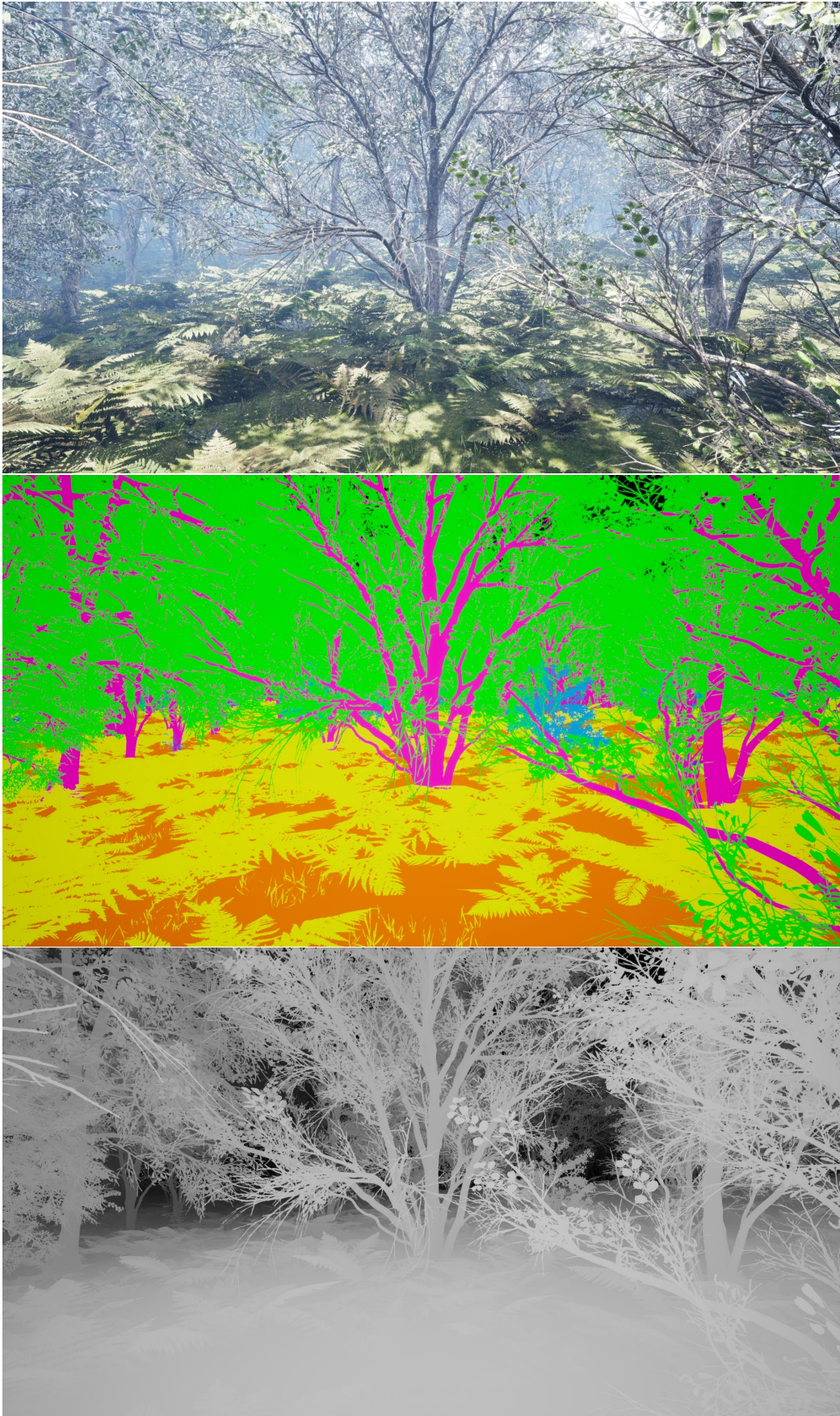


Figure A.7: Set of data extracted from ground perspective, with "Random Capture" mode, with day illumination.

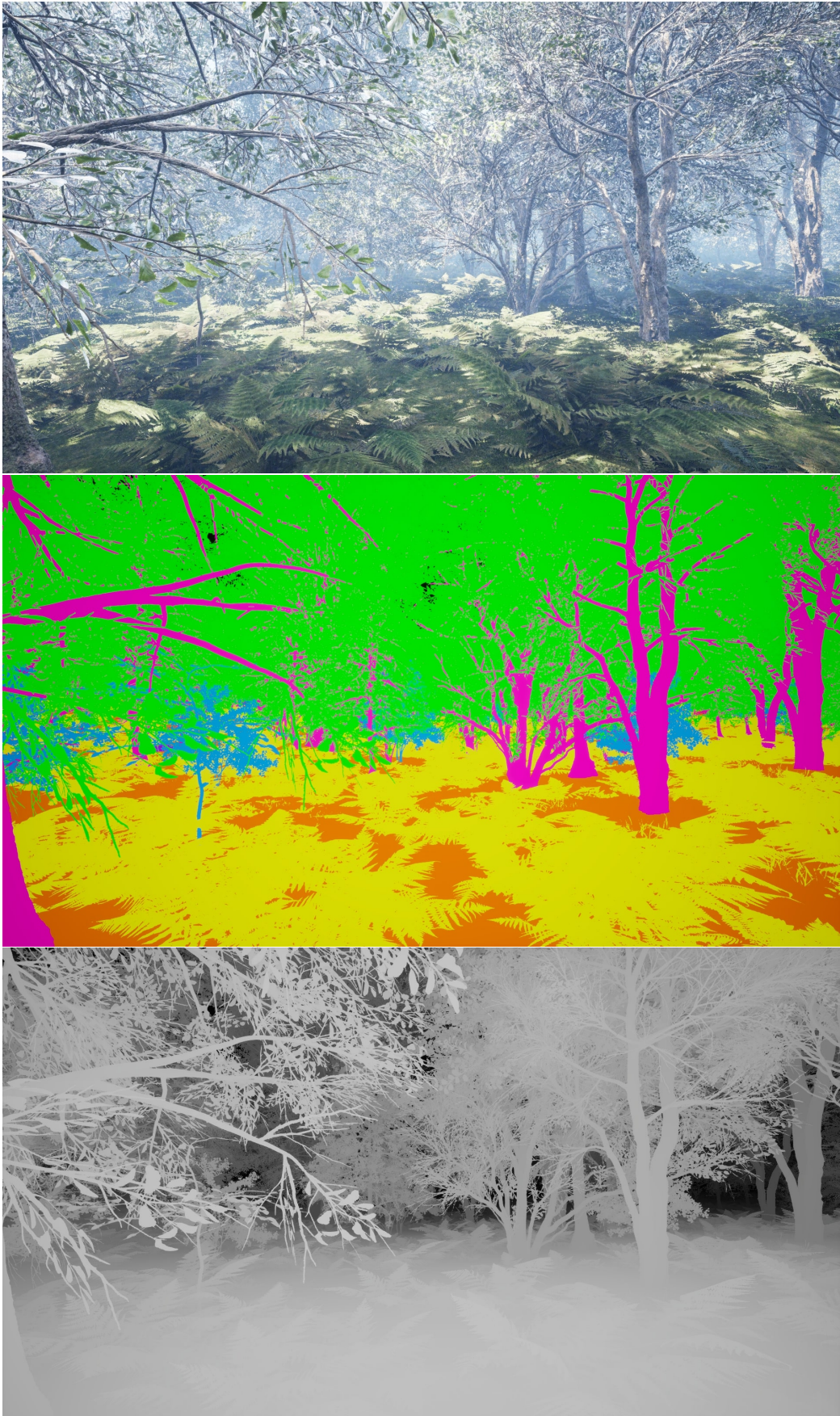


Figure A.8: Set of data extracted from ground perspective, with "Random Capture" mode, with day illumination.

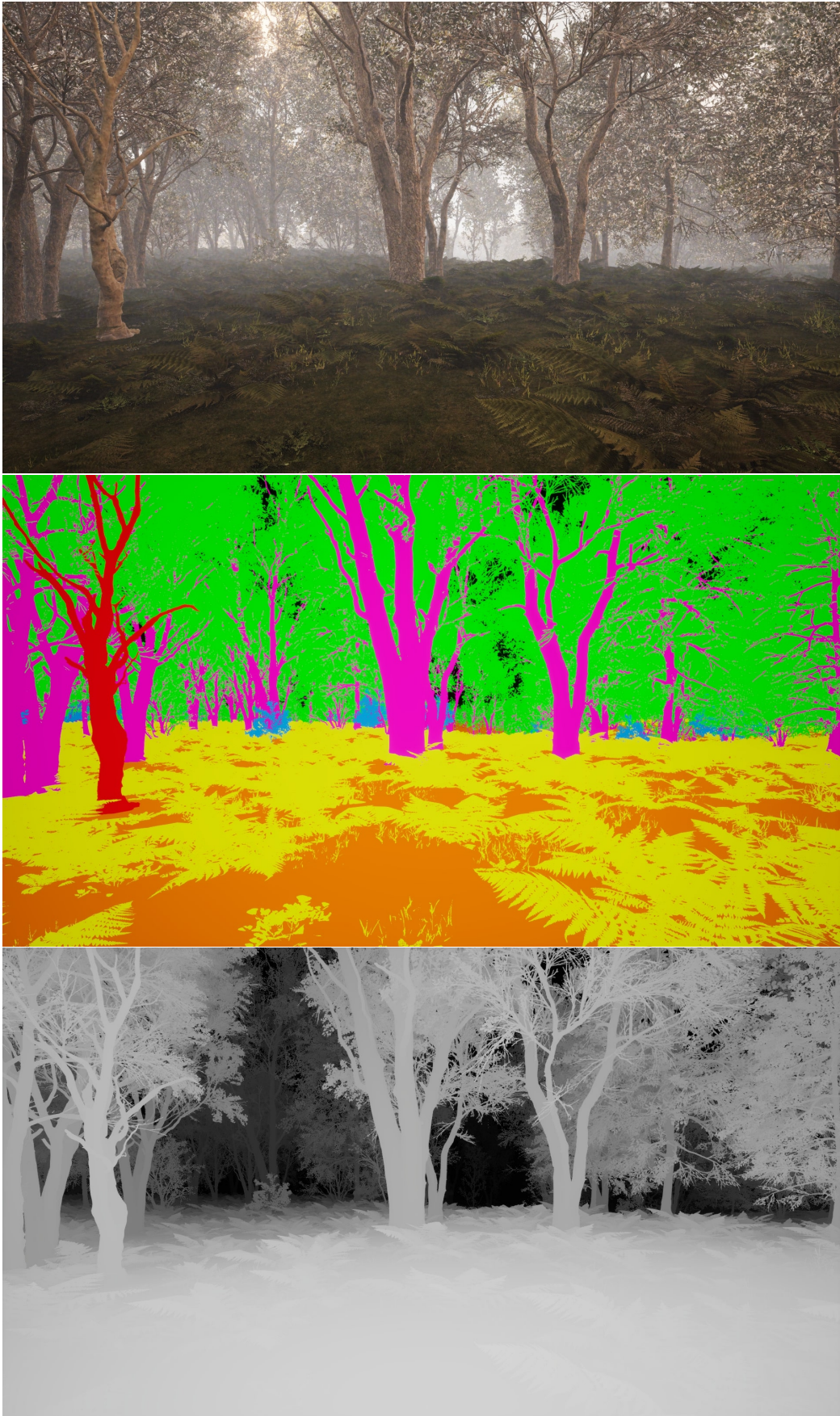


Figure A.9: Set of data extracted from ground perspective, with "Random Capture" mode, with dusk illumination.

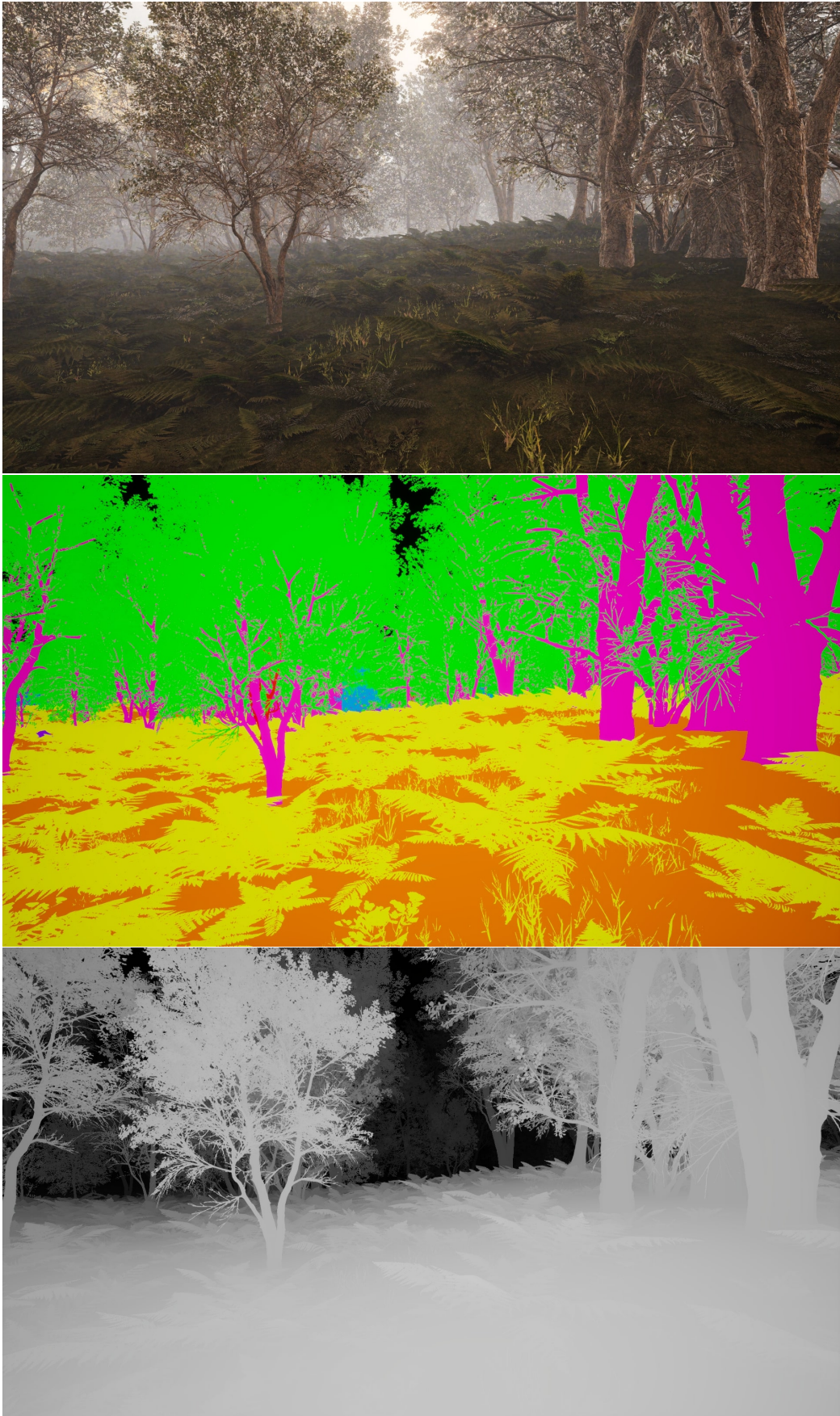


Figure A.10: Set of data extracted from ground perspective, with "Random Capture" mode, with dusk illumination.

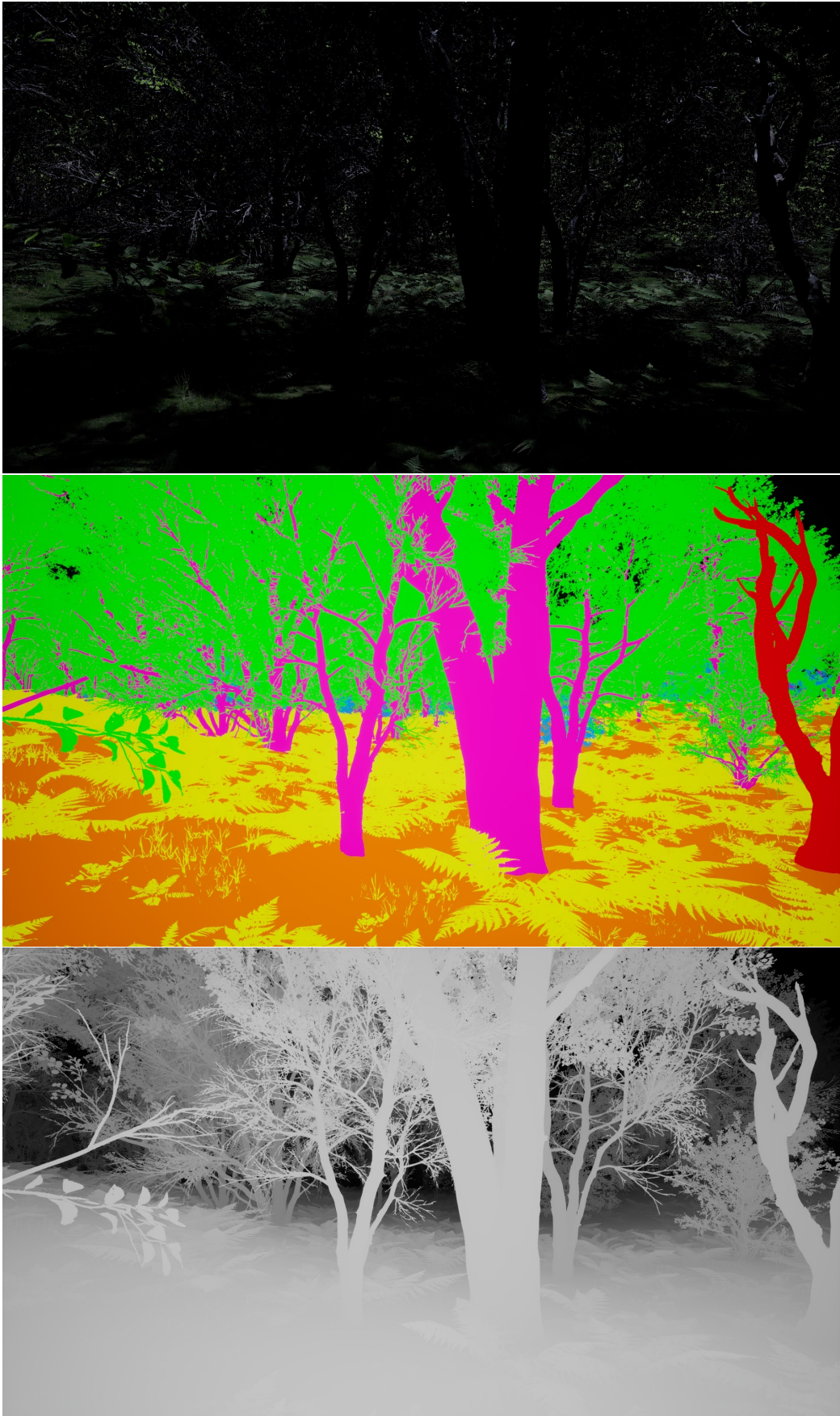


Figure A.11: Set of data extracted from ground perspective, with "Random Capture" mode, with night illumination.

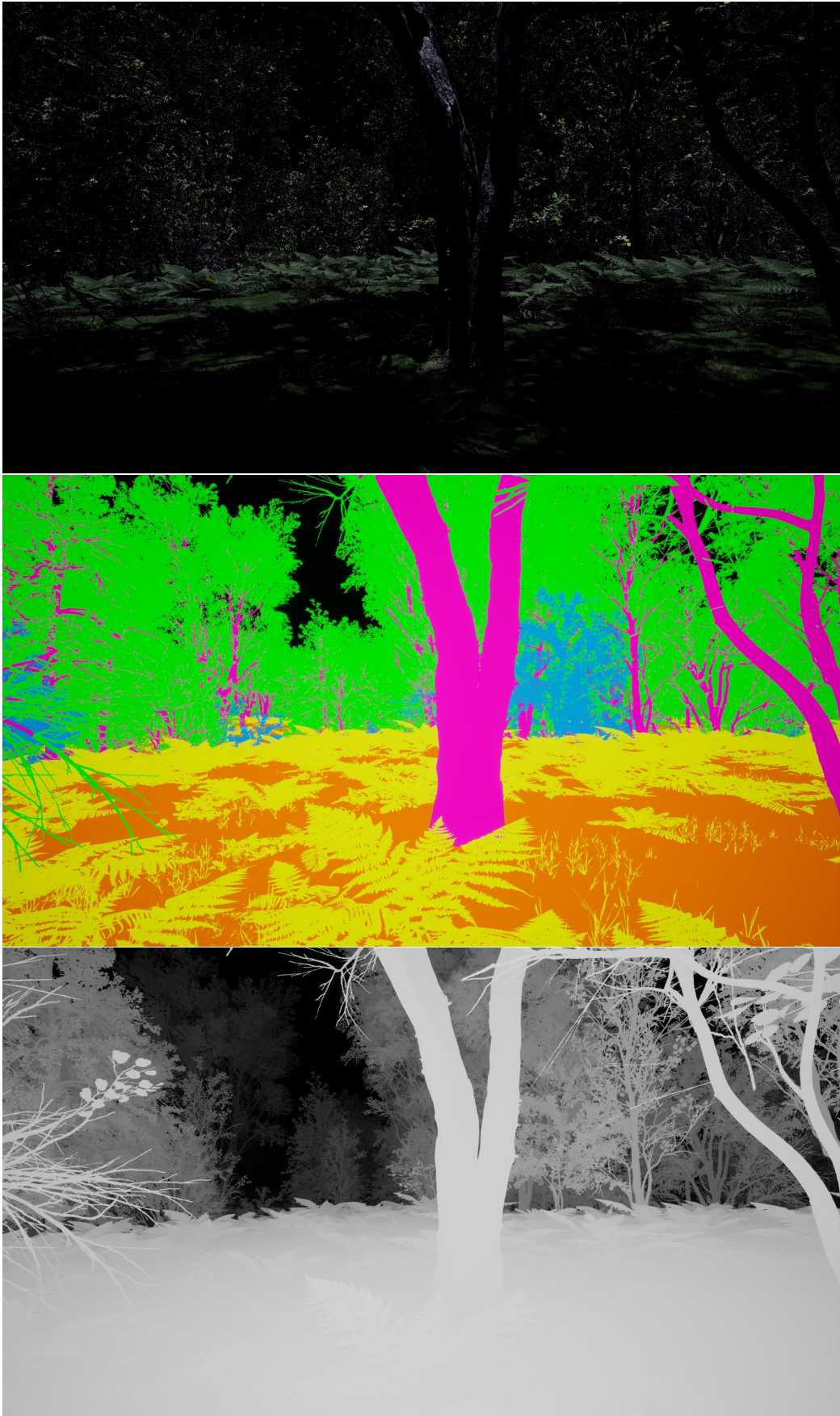


Figure A.12: Set of data extracted from ground perspective, with "Random Capture" mode, with night illumination.

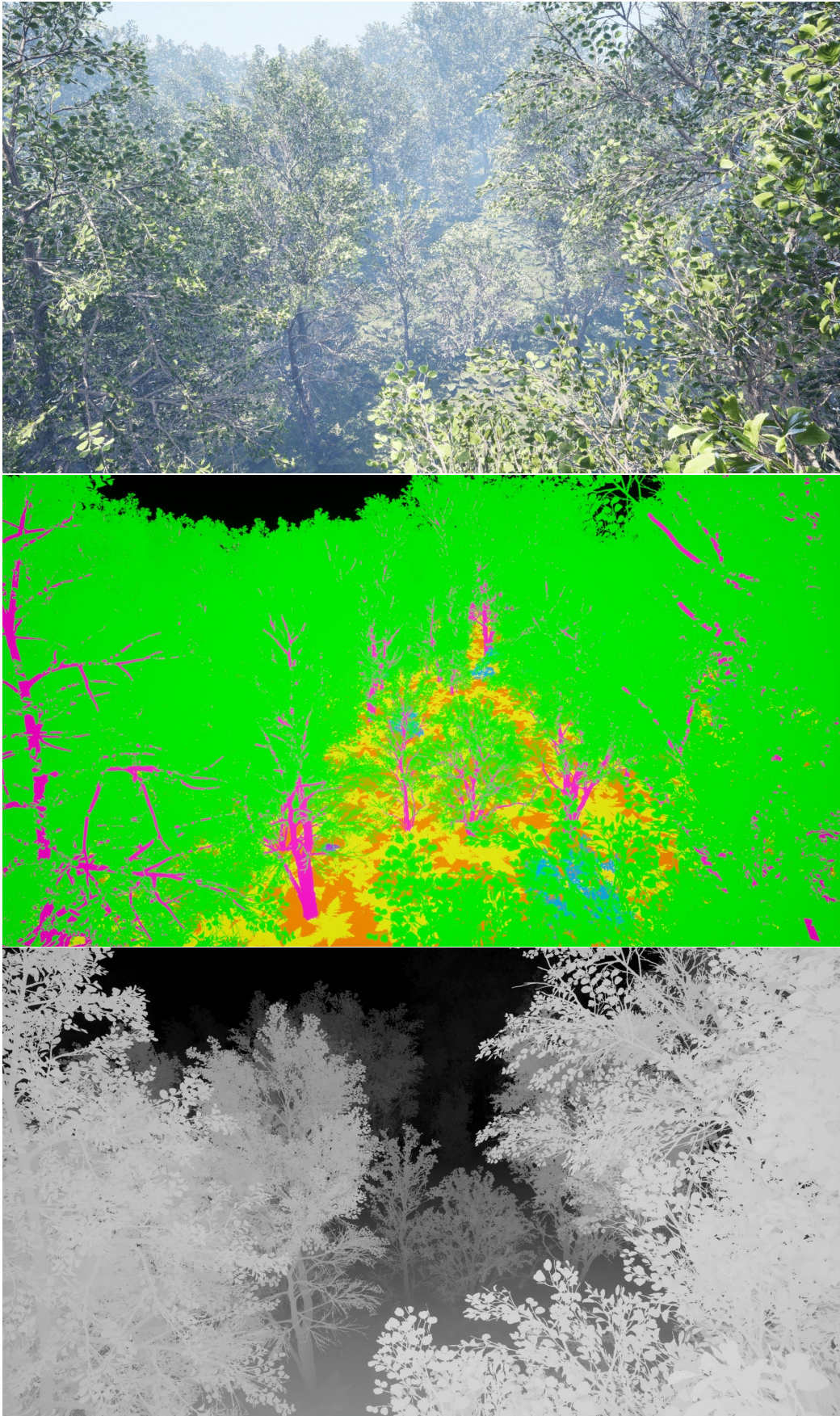


Figure A.13: Set of data extracted from aerial perspective, with "Random Capture" mode, with day illumination.



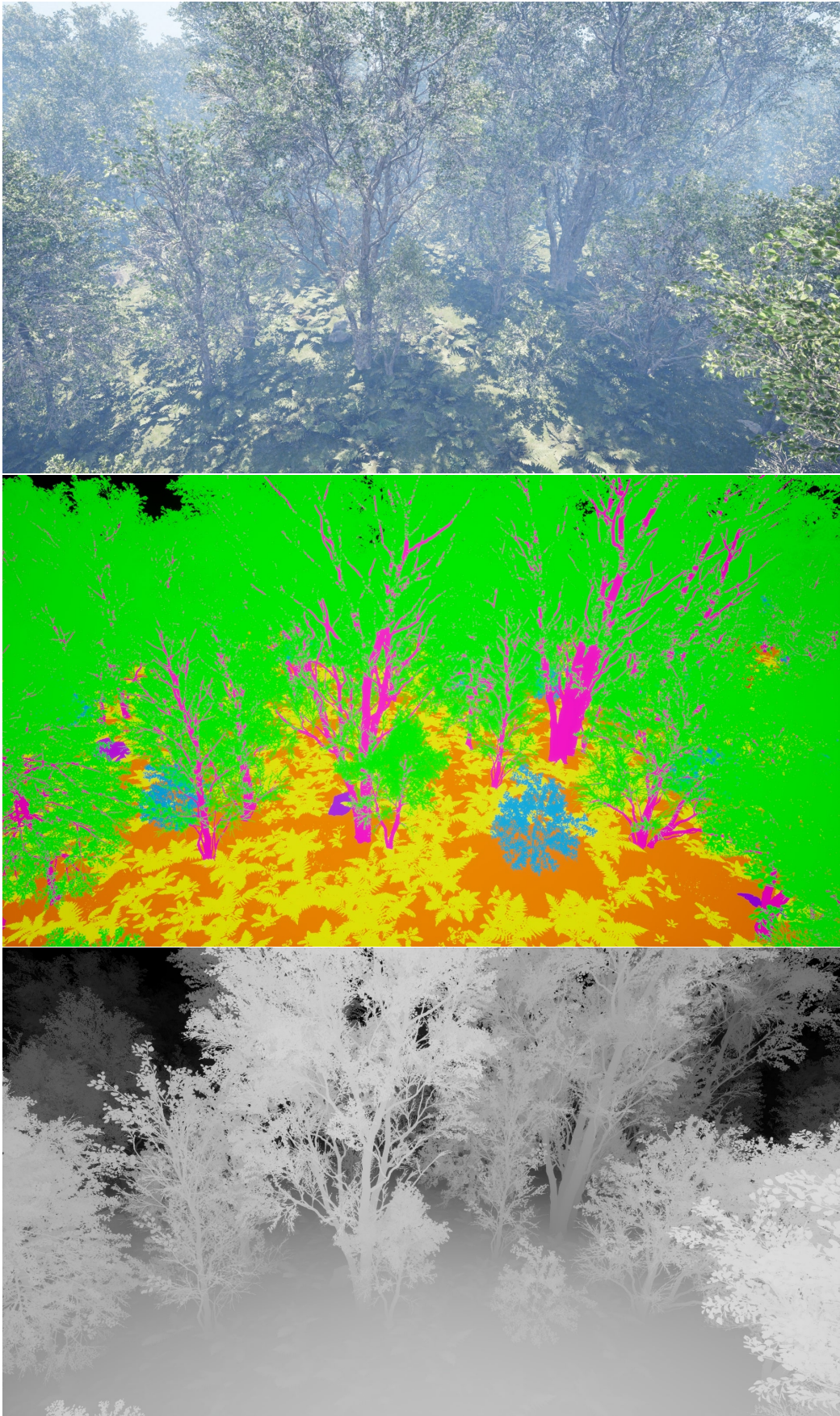


Figure A.14: Set of data extracted from aerial perspective, with "Random Capture" mode, with day illumination.

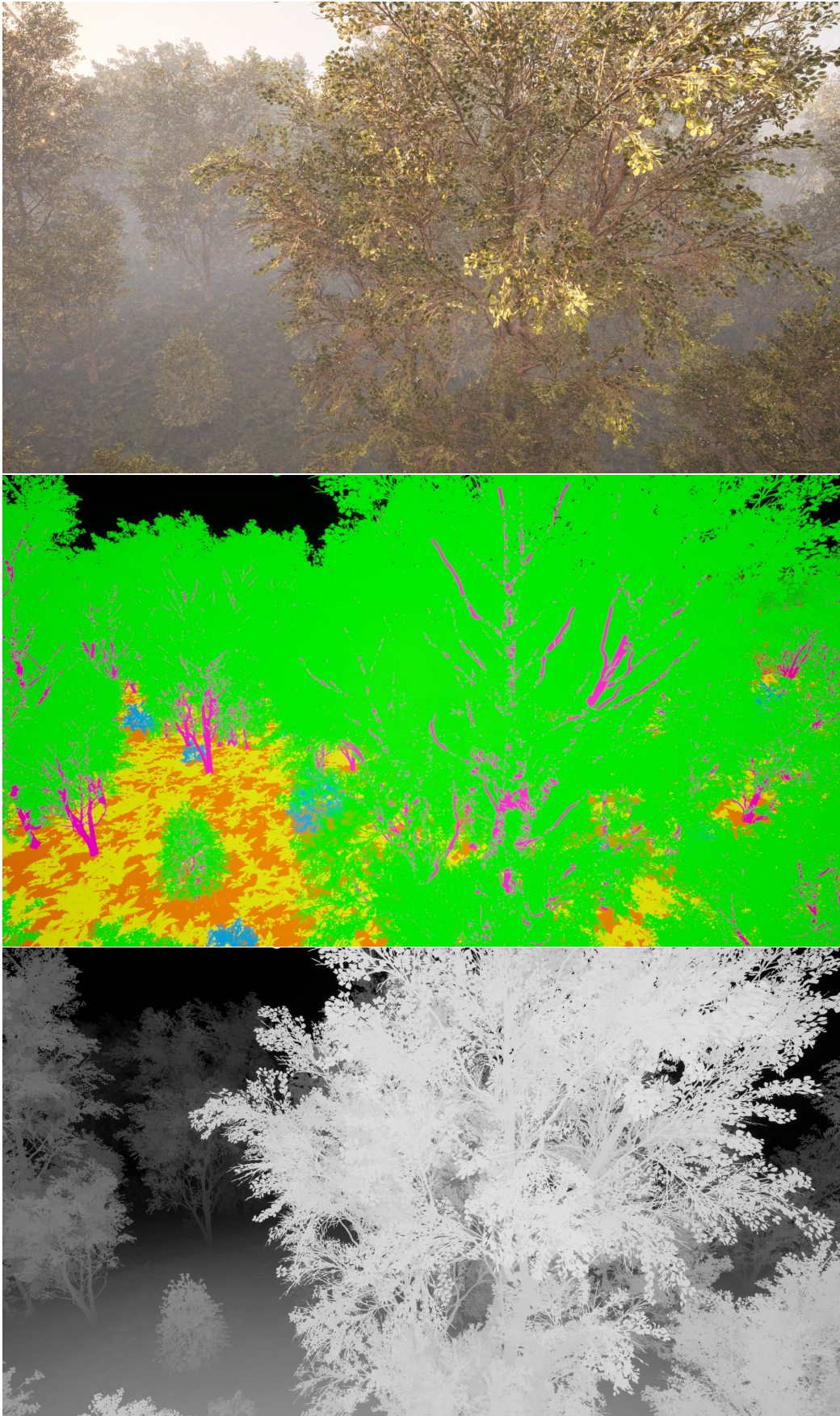


Figure A.15: Set of data extracted from aerial perspective, with "Random Capture" mode, with dusk illumination.

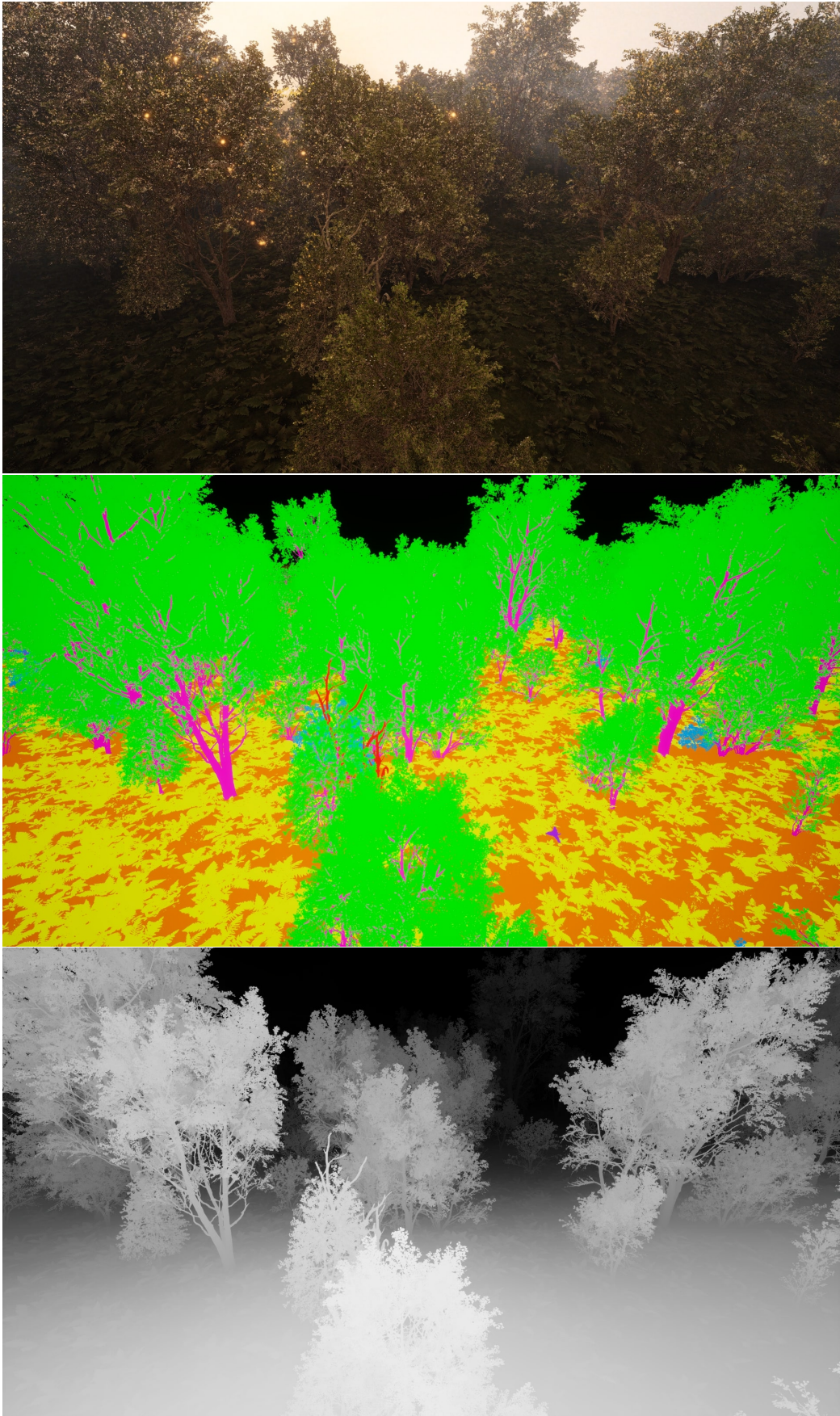


Figure A.16: Set of data extracted from aerial perspective, with "Random Capture" mode, with dusk illumination.

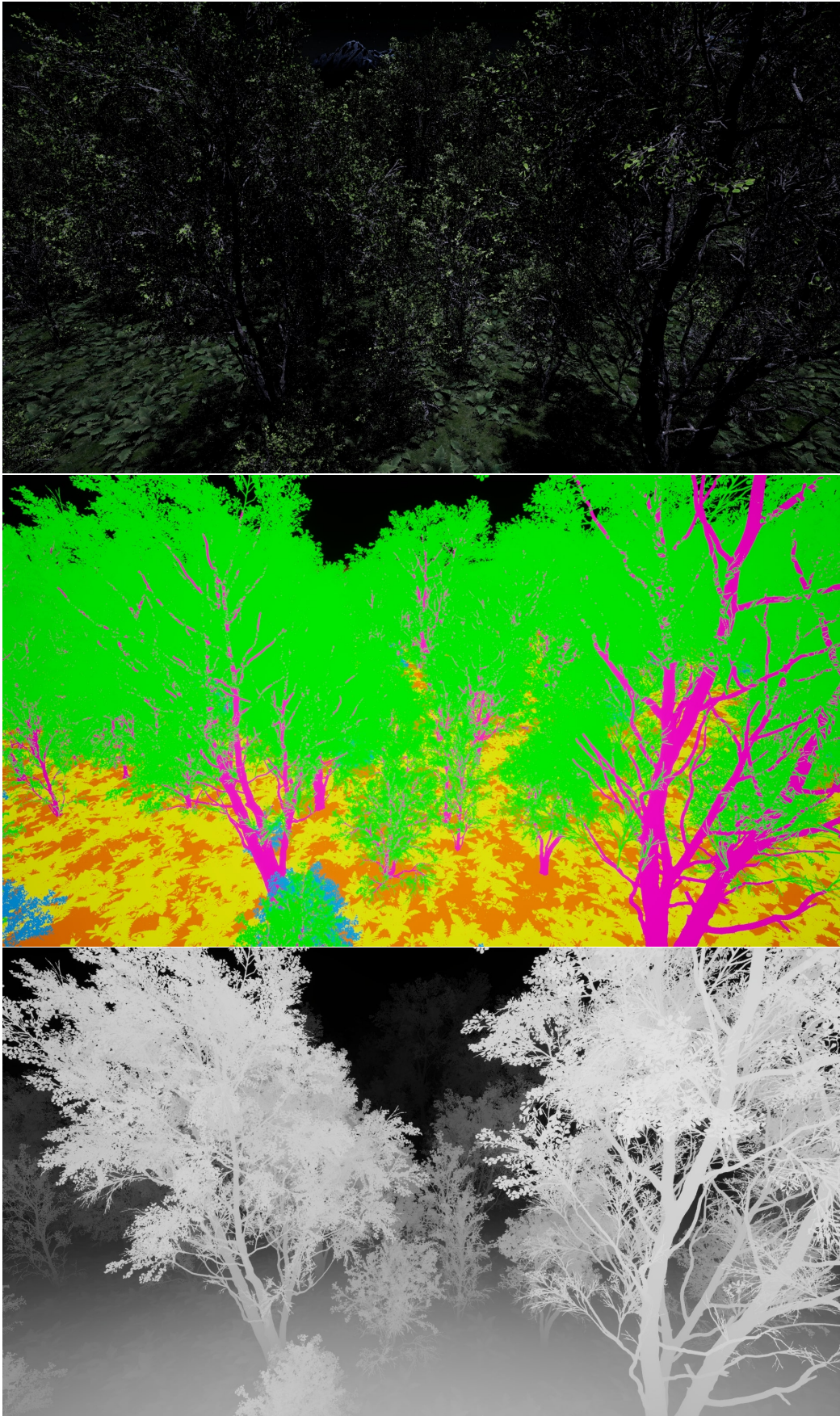


Figure A.17: Set of data extracted from aerial perspective, with "Random Capture" mode, with night illumination.

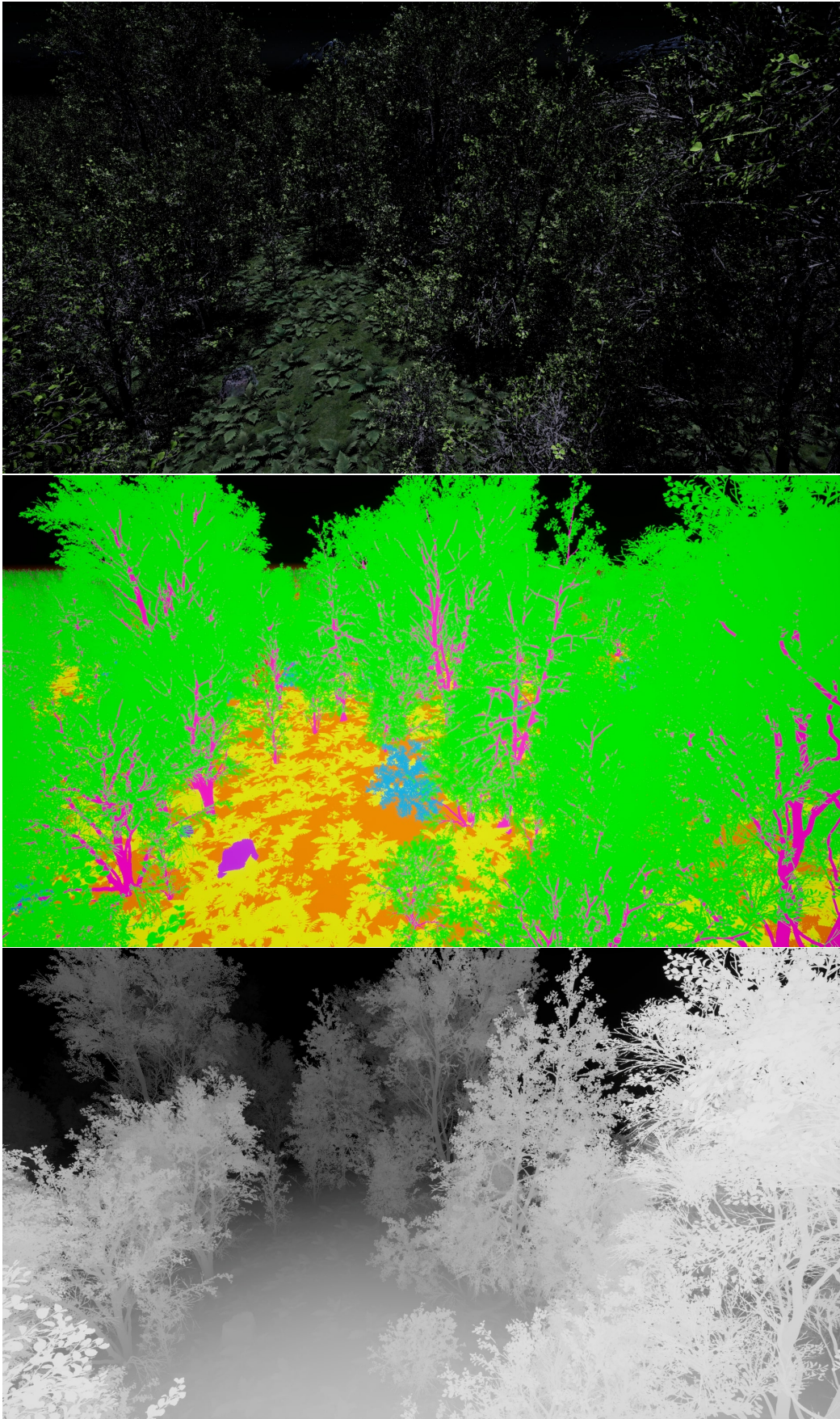


Figure A.18: Set of data extracted from aerial perspective, with "Random Capture" mode, with night illumination.

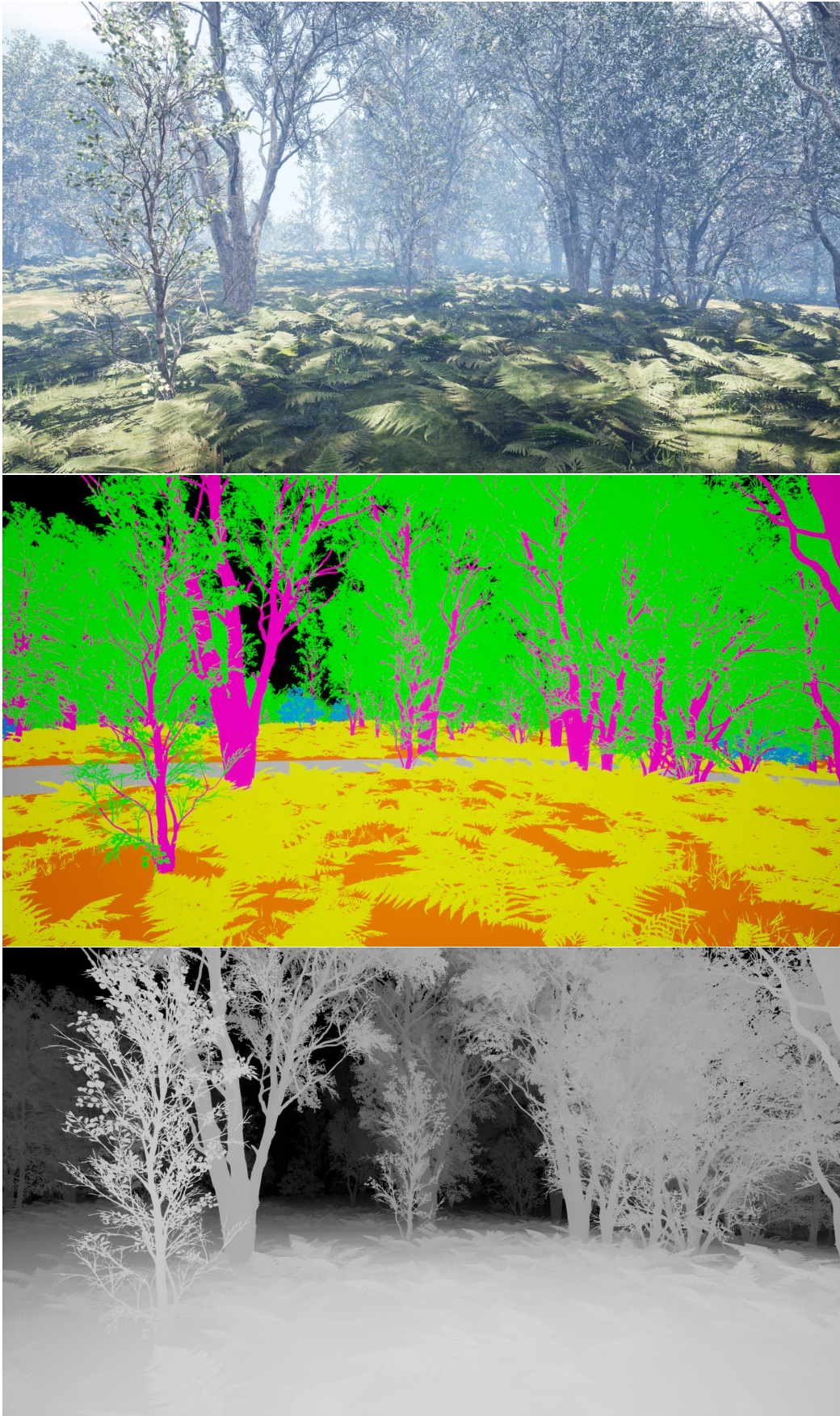


Figure A.19: Set of data extracted from ground perspective, with "Random Capture" mode, where the path is not centered.



(a)



(b)



(c)



(d)



(e)



(f)

Figure A.20: Examples of the effect of various height values in the generation. 5 (a), 50 (b), 100 (c), 150 (d), 250 (e) and 350 (f) centimeters.



(a)



(b)



(c)



(d)



(e)



(f)

Figure A.21: Examples of the effect of various density values in the generation. 1 (a), 5 (b), 10 (c), 15 (d), 20 (e) and 25 (f) percent.



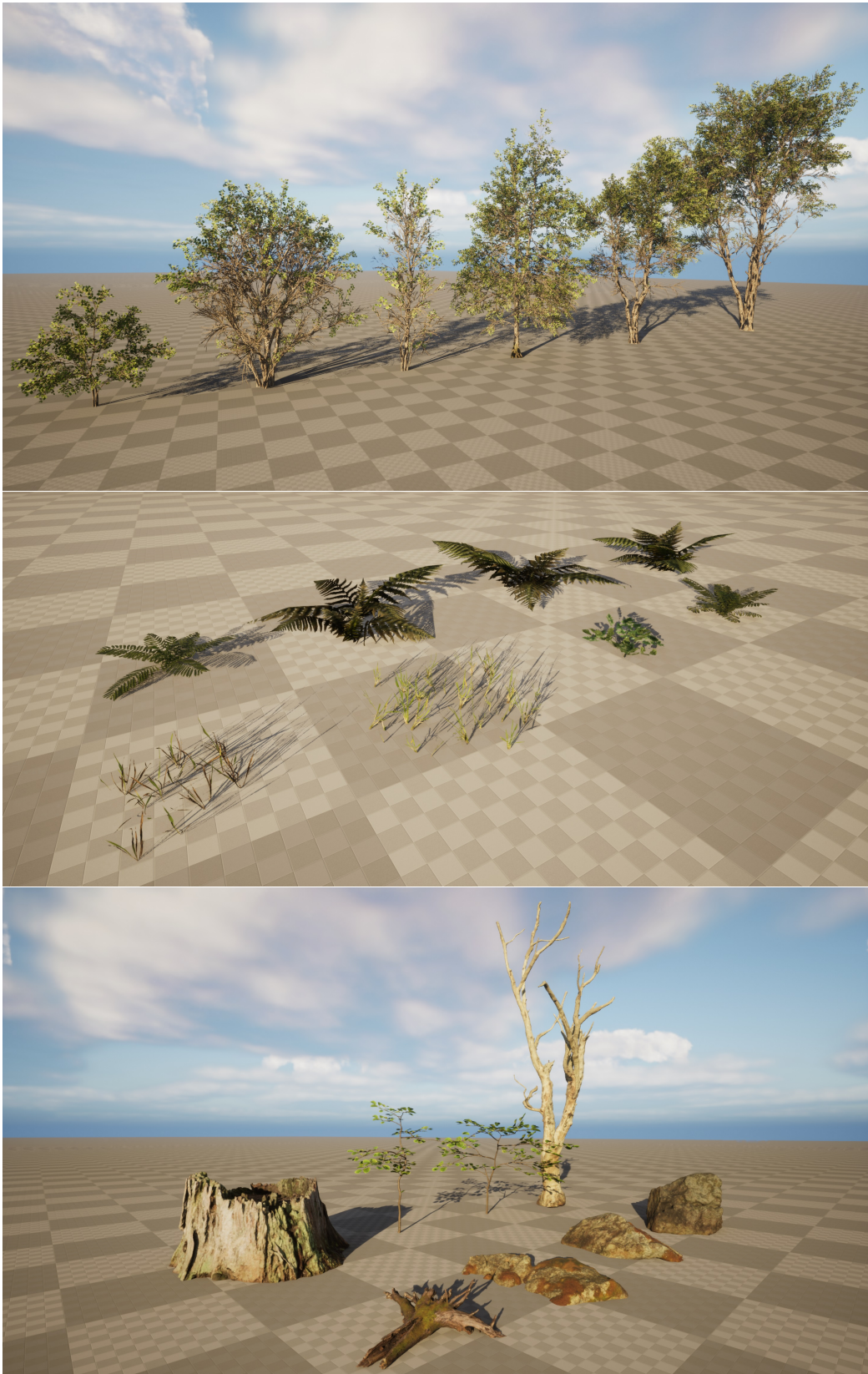


Figure A.22: Collection of assets used in object placement.