Designing a Photorealistic Simulation Platform for Robust Semantic Segmentation in Autonomous Vehicles

Author

Pham Doan Phuong Anh

Student ID: 210191

Supervised by

Prof. Huynh The Dang

A Thesis submitted to partially fulfill the requirements for the degree of
Bachelor of Science in Computer Science

Department of Computer Science
Fulbright University Vietnam
2025

Abstract

This capstone project proposes a comprehensive pipeline and foundational framework for developing autonomous driving systems. The primary contribution is the creation of a high-fidelity simulation platform designed to generate realistic urban environments for testing and training autonomous driving models. As a part of this platform, semantic segmentation is demonstrated as one example of an application that benefits from simulated data. The platform enables the generation of synthetic images under varying real-world conditions, facilitating the training of models for tasks such as object detection and scene understanding. The project also addresses challenges like class imbalance, offering insights and solutions for improving model performance in this context. The results highlight the potential of using simulated environments as a cost-effective and scalable approach for developing and testing autonomous driving applications. Further recommendations for refining the platform include incorporating real-world data, improving model generalization, and enhancing segmentation accuracy for deployment in real-time driving scenarios.

Acknowledgments

I would like to express my deepest gratitude to Prof. Dang Huynh for your empathy and support, not only throughout this project but also beyond its boundaries. Your guidance has profoundly influenced my mindset and shaped the way I perceive various aspects of life.

My sincere thanks also go to all the faculty members in the Department of Computer Science for guiding me through the courses, delivering insightful lectures, and equipping me with a solid foundation of knowledge.

To my family, thank you for always being there for me, both financially and emotionally, throughout this journey.

I am especially thankful to my friends and peers who stood by my side during this time. To Trang Nguyen, even though we are 12 hours apart, your constant support meant the world to me. Having you to share this challenging journey with made it all the more fulfilling. Da Lat would not be Da Lat without you and the GDSC Team—Ha Chi, Minh Khue, Quynh Nhu, Ngoc Han, and Ngoc Khanh. My heartfelt thanks to my friends from the Spain Team, Celes, Zach, and Trung, for always making time to cheer me up and lift my spirits. To Thuy Khue, thank you for your companionship. You've been great, and I wish you all the best on your journey next year.

Finally, to all the friends I met during my last year at Fulbright, thank you for being part of this unforgettable chapter.

Along this journey, I came to see that happiness was never the destination, but the path itself.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Background

Autonomous driving [1] refers to the use of advanced technologies, including artificial intelligence (AI), machine learning, computer vision, and sensor fusion, to enable vehicles to operate without human intervention. These self-driving systems are designed to perceive their environment, make decisions, and control the vehicle safely through complex driving scenarios. Autonomous vehicles (AVs) aim to replicate or even surpass human driving performance by continuously learning from massive datasets and improving their ability to handle diverse and unpredictable real-world conditions. The significance of autonomous driving lies in its potential to revolutionize transportation across several dimensions [2]. First, it promises to improve road safety by reducing human error, the leading cause of traffic accidents worldwide. AVs can also improve mobility for people to

drive, such as the elderly or people with disabilities. From an economic and environmental perspective, autonomous systems can increase transportation efficiency, reduce congestion, optimize fuel consumption, and support the transition to electric and shared mobility models [2]. Furthermore, AVs are expected to transform industries such as logistics, ride-hailing, public transportation, and urban planning.

The development of autonomous vehicles (AVs) relies heavily on large-scale, diverse datasets to train machine learning models responsible for perception, decision-making, and navigation [3]. With more than 10 trillion automobile miles driven globally each year, the range of possible driving scenarios is vast, making comprehensive data collection essential and challenging. Traditional methods of acquiring real-world driving data face significant barriers, including high operational costs—exemplified by the cost of LiDAR sensors in India, which can equal that of two entry-level cars—and regulatory and safety limitations, especially in regions lacking robust infrastructure, traffic law enforcement, or standardized AV frameworks.

In response to these challenges, simulated environments have emerged as a powerful alternative, offering scalable, controlled, and repeatable conditions for training and validating AV systems. Industry leaders such as Waymo and Baidu have integrated cloud-based simulation into their AV development pipelines, enabling accelerated testing and model refinement in safe virtual settings [4]. These simulation platforms often feature unified systems for data storage, HD map generation, and deep learning model training, all of which are crucial for developing safe and reliable autonomous vehicles.

A key component of AV perception is environmental understanding, including identifying drivable areas and detecting surrounding obstacles. Semantic segmentation, which classifies each pixel in an image into meaningful categories, plays a vital role in this task. However, training robust semantic segmentation models requires large volumes of annotated image data, and manual labeling of real-world data is both time-consuming and resource-intensive [5].

To address this bottleneck, this thesis proposes a simulation-based pipeline for generating synthetic driving scenarios aimed at efficiently creating labeled datasets for semantic segmentation. The system is designed to simulate realistic driving environments and automatically generate annotated image data. Using the synthetic dataset, the DeepLabV3 model—a state-of-the-art semantic segmentation network—is fine-tuned to improve its performance in perception tasks. This approach demonstrates the potential of simulation to reduce reliance on manual data annotation while accelerating the development of perception models for autonomous driving.

Looking ahead, the future development of this simulation platform can be tailored to support underrepresented countries like Vietnam, where AV research is still in early stages. By incorporating Vietnamese-specific graphics, road layouts, signage, and typical vehicle types such as motorbikes, the platform can simulate traffic behavior that accurately reflects Vietnam's dynamic urban environments. This would enable more locally relevant training data, helping to address the unique challenges of deploying autonomous systems in Southeast Asian contexts and further democratizing AV technology globally.

1.2 Problem Statement

A central problem hindering the advancement of autonomous driving systems is the lack of diverse and large-scale datasets that accurately represent the complexities of real-world traffic scenarios [4]. The sheer variety of real-world driving conditions is immense. Autonomous vehicles need to operate in diverse operational design domains (ODDs), ranging from highways to crowded urban areas and even unstructured environments [4]. This creates millions of novel situations where autonomous systems could potentially fail. Existing datasets often struggle to capture this full spectrum of possibilities.

Transferring knowledge from simulated environments to the real world (sim2real) is challenging due to discrepancies in data. While simulators like the AutoDRIVE Simulator can generate labeled data, the dynamic behavior of scaled or full-scale vehicles and the sensory data obtained in simulation can differ significantly from real-world conditions [6]. This necessitates significant effort in re-tuning algorithms when moving from simulation to deployment. Implementations based on visual perception are particularly sensitive to variations in sensory data [6].

The need for billions of miles of testing in an almost infinite range of situations underscores the data scarcity. To ensure the safety and reliability of AI-powered self-driving systems, they ideally need to be tested across a virtually limitless number of real-world scenarios [7]. Acquiring such a vast amount of real-world driving data is a monumental and ongoing task. Regional variations in traffic patterns, driver behavior, and infrastructure further exacerbate the dataset problem. Autonomous systems trained on data from one region might not generalize well to others due to differing traffic rules, road layouts, driving habits, and the types of vehicles present [6]. Datasets need to account for these diverse regional specificities. In essence, the current problem with data sets for autonomous driving systems is that they often do not adequately represent the diversity,

complexity, and regional nuances of real-world traffic. This limitation directly impacts the ability to train AI models that can understand the spatial information and semantic information across a wide range of conditions [4]. Addressing this gap through the collection of more diverse real-world data and the development of high-fidelity simulation environments to augment this data remains a critical area of focus [6], [4].

In essence, the current problem with data sets for autonomous driving systems is that they often do not adequately represent the diversity, complexity, and regional nuances of real-world traffic. This limitation directly impacts the ability to train AI models that can understand the spatial information and semantic information across a wide range of conditions [4]. A particularly important consideration in this context is that traffic in Vietnam presents unique challenges that differ significantly from the driving environments in many Western countries. This thesis is motivated by the need to address these distinct characteristics by developing a simulation environment that accurately reflects the special traffic conditions in Vietnam. The goal of this research is to lay the foundation for future studies in autonomous driving systems tailored to the Vietnamese context. By creating a simulation that enables customizations, this thesis aims to provide a critical tool for further research and development in autonomous driving technology specific to the underrepresented region [6], [4].

1.3 Objectives

Driven by the motivation to explore innovative technologies in autonomous driving, this thesis aims to develop a simulation platform that generates synthetic driving scenarios to fine-tune the DeepLabV3 model for autonomous driving tasks, with a particular emphasis on enhancing environmental perception through semantic segmentation. The specific objectives of this research are as follows:

- Design and develop a Simulation Platform using Houdini and Unreal Engine 5.
- Generate Synthetic Training Data from the Developed Platform.
- Fine-Tune `DeepLabV3` for Autonomous Driving Tasks.
- Evaluate Model Performance.
- Explore Future Applicability for Underserved Regions.

1.4 Scope and Limitations

1.4.1 Proof-of-Concept Implementation

This thesis focuses on the development of a proof-of-concept implementation of a simulation platform designed to generate synthetic driving scenarios for fine-tuning the DeepLabV3 model. The platform is not intended to be a fully comprehensive autonomous driving system, but rather a testing environment that demonstrates the viability of using synthetic data to enhance environmental perception tasks such as semantic segmentation. As such, the implementation will be limited to a set of predefined traffic scenarios and model configurations, serving as a foundational step toward broader applications in autonomous driving.

1.4.2 Simulated Environment Constraints

The simulated environment used for generating synthetic data will have certain constraints in terms of the diversity of scenarios and the realism of the driving conditions. While the simulation will incorporate a variety of weather conditions, lighting, and traffic patterns, it is impossible to perfectly replicate all real-world driving variables, such as unpredictable driver behavior, complex road layouts, and edge cases. Additionally, the platform will rely on pre-programmed traffic rules and vehicle dynamics, limiting the level of spontaneity in some traffic scenarios. These constraints may affect the generalization of the model trained on synthetic data when deployed in real-world environments.

1.5 Contributions

1.5.1 Scalable Synthetic Dataset for Traffic Scenarios

A key contribution of this thesis is the development of a scalable platform for generating synthetic datasets that simulate diverse urban traffic scenarios. This simulation platform encompass various weather conditions, lighting scenarios, and dynamic traffic behaviors, enabling the training of deep learning models in autonomous driving applications. By automating the creation of labeled synthetic data, this platform significantly reduces the reliance on time-consuming and costly

manual annotation. The platform provides a more efficient and cost-effective means of developing robust perception models, addressing challenges such as class imbalance in training data, and enhancing the generalization of the model to real-world scenarios.

1.5.2 Open-Source Platform for Autonomous Driving Research and Education

In addition to the dataset generation platform, this thesis contributes an open-source initiative aimed at advancing autonomous driving research and education. The platform provides tools and frameworks for simulating synthetic driving scenarios, training deep learning models for perception tasks, and evaluating the performance of these models. By offering access to generated datasets and modeling frameworks, the open-source platform supports collaborative research and accelerates innovation in autonomous driving systems. It enables other researchers, developers, and educators to leverage these resources to explore further advancements, fostering progress toward safer, more efficient, and robust autonomous vehicle technologies.

2

Literature Review

Contents

Waymo [8], Tesla [9], Cruise [10], and Baidu [11], as well as traditional car manufacturers like Audi [12], BMW [13], and Toyota [14], have made significant investments in full-scale autonomous vehicle (AV) development. These corporations frequently operate in huge, privately owned test facilities that are inaccessible to outside researchers and students. As a result, academic institutions have begun constructing scaled-down AV platforms such as MIT Racecar [15] to promote education and research. However, many of these platforms rely on costly, commercially available RC cars and high-end sensor suites, posing financial and logistical challenges, particularly in developing countries like Vietnam.

Despite the availability of a broad range of commercial and open-source simulators for autonomous driving—ranging from industry-grade platforms like Ansys Autonomy [16], CarMaker [17], and NVIDIA Drive Constellation [18] to research-oriented tools such as Gazebo [19], CARLA [20], AirSim [21], Sim4CV [22], TORCS [23] and LGSVL [24] - few of these fully accommodate the specific requirements of scaled-down autonomous vehicle development. Most lack support for

smaller hardware-in-the-loop integration or do not provide realistic kinodynamic modeling that is consistent with scaled automobile behavior. Furthermore, their limited customizability, license constraints, and high computational or financial costs make them unsuitable for widespread educational or low-resource research contexts, particularly in emerging countries like Vietnam. This leaves a huge gap in tools for both accessible development and reliable sim2real transfer. To address these limitations, this thesis proposes creating an open-source, game-based simulation framework that allows scaled car dynamics, modular scenario creation, and the generation of limitless synthetic datasets for self-driving applications. By leveraging widely adopted game engines and open development practices, the proposed system aims to provide a low-cost yet high-fidelity virtual environment tailored to research, education, and innovation in autonomous driving, particularly in underrepresented regions like Vietnam.

2.1 Existing Traffic Simulation Approaches

Existing traffic simulations widely used in the autonomous driving industry include CARLA [20], AirSim by Microsoft [21], Sim4CV [22], BeamNG.tech [25], TORCS [23], and LGSVL by LG Electronics [24]. These simulators have become integral tools for developing, testing, and validating autonomous vehicle systems due to their ability to generate realistic driving environments, complex traffic scenarios, and varied sensor data. They provide high-fidelity simulations, allowing researchers and developers to refine algorithms before deploying them in real-world conditions.

Most popular simulation platform, CARLA [20] offers an open-source simulation platform that provides a highly detailed urban environment for autonomous driving research. It supports a wide range of sensors, traffic scenarios, and weather conditions, making it ideal for testing autonomous vehicle algorithms in dynamic environments.

Recently, AirSim [21], developed by Microsoft, also take part in field of drone and autonomous driving simulations. The simulation offers photorealistic environments, diverse sensor models, and a strong integration with machine learning frameworks, making it highly useful for developing autonomous systems that require real-world-like data.

Sim4CV [22] focused on computer vision applications within autonomous driving. The simulation is designed to generate realistic visual data for training and evaluating vision-based algorithms, which is essential for systems relying on computer vision for tasks like object detection and traffic sign recognition.

TORCS [23] is a racing car simulator used to develop autonomous driving algorithms, particularly for high-performance applications. Its detailed vehicle dynamics and challenging racing scenarios make it a valuable tool for testing control algorithms in competitive environments.

LGSVL [24], developed by LG Electronics, is a versatile simulation platform known for its high-fidelity real-world city models and support for various autonomous driving sensors. It is scalable and flexible, making it suitable for both academic research and industrial applications, where testing across different vehicle types and traffic conditions is essential.

Among the four simulation platforms examined, CARLA [20], AirSim [21], and Sim4CV [22] are the most widely used and actively maintained to date. As a result, a detailed comparison in Table 2.1 was conducted focusing on these three engines.

Table 2.1: A Comparative Analysis of CARLA, AirSim, and Sim4CV Simulation Platforms for Autonomous Driving Research

Feature	**CARLA [20]**	**AirSim [21]**	**Sim4CV [22]**
Platform	Unreal Engine	Unreal Engine	Custom
Open Source	✓	✓	✓
Simulated Vehicles	Car, Bus, Pedestrian, Bicycle	Car, Drone, Boat	Car, Pedestrian
Vehicle Dynamics	High-fidelity, PhysX-based	Moderate fidelity, Unreal PhysX	Moderate fidelity
Scalability (for small-scale vehicles)	Moderate	Moderate	High
Sensor Suite	LiDAR, Camera, Radar, GPS, IMU	Camera, LiDAR, IMU, GPS	Camera, LiDAR, Radar
Weather/Environment Simulation	✓	✓	Limited
Real-time Rendering	✓	✓	✓
Customization	High	High	Moderate
Ease of Use	Moderate	Easy	Moderate
Support for Machine Learning	✓(Integration with ML tools)	✓(Integration with TensorFlow, PyTorch)	✓(Limited)
Supported Programming Languages	Python, C++	Python, C++	Python
Real-world Testing	✓(Real-world dataset support)	✓(Real-world dataset support)	×
Target Audience	Researchers, Developers, Industry	Researchers, Developers, Educational Use	Computer Vision, Robotics Research

The need to develop a new traffic simulation arises from the limitations and specific focuses of existing simulators. While platforms like CARLA [20], AirSim [21], and Sim4CV [22] offer high-

fidelity environments and diverse features, each is tailored to particular use cases—urban driving, drone integration, or computer vision, respectively. However, there is still a gap for more customizable, lightweight, or specialized simulations that can cater to emerging requirements in autonomous systems or reflect underrepresented regions like Vietnam. This motivates the development of new simulation tools that better align with specific research or deployment needs.

2.2 Game Development for Simulation

2.2.1 Houdini as a 3D software solution to create the simulation

Introduction to Houdini

Developing a complex urban environment, such as a city model, requires a 3D software solution capable of handling large-scale content creation with efficiency and flexibility. Given these requirements, Houdini was selected as the primary 3D software due to its renowned procedural approach and suitability for robust pipeline creation.

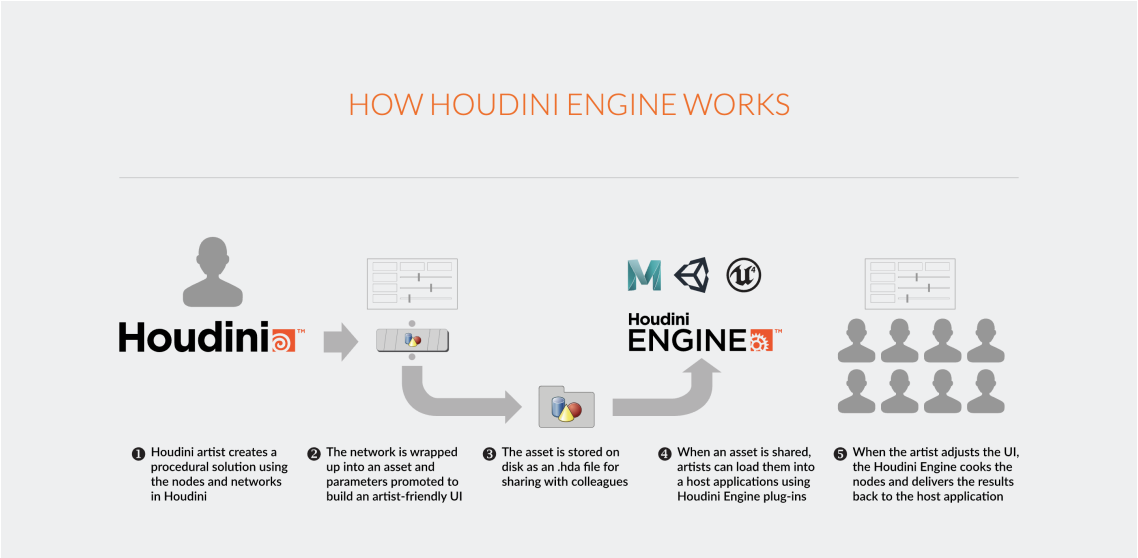


Figure 2.1: Procedural content creation workflow in Houdini, highlighting the steps involved in generating complex 3D urban environments and collaboration between artists [26]

Houdini, a 3D animation software developed by SideFX, is a high-end 3D animation and visual effects software used widely in the film, TV, and gaming industries. It's best known for its node-based procedural workflow show in Figure 2.2, which gives artists a lot of flexibility and control when creating complex visual effects (VFX), simulations, and animations.

The right side of Figure 2.2 illustrates the procedurally generated 3D city layout, while the left side presents the corresponding node-based workflow used to build it. The visible nodes represent key urban components, such as buildings and road networks, each contributing to the procedural system.

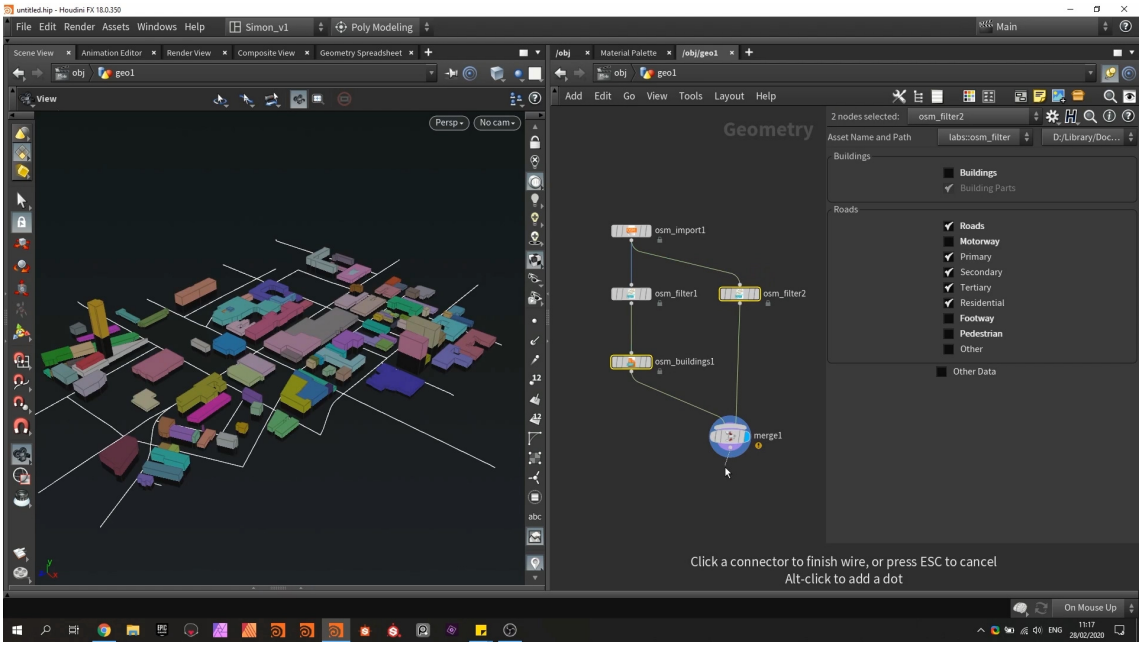


Figure 2.2: Node-based system in Houdini. This screenshot shows the Houdini interface used to procedurally generate an urban environment, with each node representing a specific component or transformation in the city-building process [26]

Functionalities of Houdini

Generating a complex urban environment, such as a detailed city model, requires a 3D software solution that excels in handling large-scale content creation with efficiency and flexibility. With these demands in mind, Houdini was selected as the primary 3D software, fundamentally due to its robust procedural architecture and its proven efficacy in generating intricate environments. This is particularly advantageous when creating detailed and extensive environments like cityscapes, where many elements (buildings, roads, parks, etc.) can be generated and modified based on defined rules and parameters. It becomes possible to explore and prototype ideas early in the development process and subsequently refine and scale the solution to produce the final complex city model in this project.

The creation of procedural content is increasingly vital in production pipelines, including games and visual effects, offering significant reductions in timelines and improving flexibility. This is particularly beneficial for creating extensive cityscapes, where numerous elements can be generated and modified based on defined rules and parameters. Houdini's procedural workflow operates like a "recipe", allowing the reuse of a network with different inputs and parameters to produce unique results with minimal adjustments. Assets created this way are considered "live," meaning their results are easily modified by changing inputs, contrasting sharply with rigid "baked data". This

”live” nature is invaluable for iterating rapidly on city layouts or design elements without costly manual rebuilding [27].

The power of this procedural approach is rooted in Houdini’s nodal network system, where users build creations by connecting nodes that sequentially modify data [27]. Changes made to any node automatically cascade through the network, updating the final output instantly. This allows ”directability” to be maintained late into production, making late-stage modifications less time-consuming and expensive than in non-procedural workflows. Additionally, complex node networks can be encapsulated into Houdini Digital Assets (HDAs), reusable ”super nodes” with user-defined parameters. This facilitates the creation of specialized tools for city modeling, such as procedural building generators or road layout tools, which can be easily controlled. The core insight behind this approach is that the true value lies in the procedural ”recipes” (the network or HDA) rather than just the static output, enabling the transmission of lightweight instructions over heavy final data [27].

Directly supporting this rationale is the application detailed in ”Procedural City Generation with Combined Architectures for Real-time Visualization” [28]. This project specifically chose Houdini for the procedural generation of a city model intended for real-time visualization. Within Houdini, the authors created and integrated major city features such as the surrounding terrain, road networks, individual buildings, and their placement. Different procedural techniques were employed for various elements; for instance, terrain was generated using height maps, roads were largely based on path cost evaluation, and buildings were derived from the parameterization of a collection of representative 3D models. In the paper, the authors found their terrain system, developed in Houdini, to be flexible, easily expandable, and efficient for game engines. This project serves as a concrete demonstration of how Houdini’s procedural capabilities and nodal architecture are successfully leveraged to manage the complexity and specific needs of generating detailed urban environments, aligning perfectly with the requirements of this thesis [28].

A variety of 3D software platforms are used in the creation of digital content, each offering unique strengths tailored to specific production needs. This section provides an overview of four prominent tools: Blender [29], Autodesk Maya [30], Houdini [26], and Cinema4D [31], focusing on their features, applications, and relevance to industry practices.

Blender [29] is a cross-platform, open-source 3D software that supports the entire animation and visual content pipeline, including modeling, animation, shading, rendering, audio integration, and video editing. It offers a highly customizable interface and advanced features such as chroma

keying, camera tracking, masking, and a powerful node-based compositor. Additionally, Blender's support for non-photorealistic rendering through FreeStyle and physically-based rendering via the GPU-accelerated Cycles engine enables both artistic and realistic outputs. Moreover, its built-in Python scripting and compatibility with third-party render engines make it particularly attractive for independent creators, especially in game development and short animation production.

In contrast, *Autodesk Maya* [30] is a widely adopted industry-standard 3D application used extensively in film, television, and high-end game development. It offers a comprehensive suite of tools for modeling, animation, simulation, and rendering. Notably, Maya is known for its strong capabilities in character rigging and animation, as well as its adaptability through scripting and plugin support. Consequently, it remains a foundational tool in professional production pipelines, particularly in studios working on large-scale cinematic projects and virtual production environments.

On the other hand, *Cinema 4D* (C4D) [31] is a 3D modeling and animation application recognized for its ease of use and efficient production workflows. With its streamlined interface and bundled toolsets, C4D allows for the quick development of visually compelling graphics and animations. One of the distinguishing features of C4D is its seamless integration with Adobe After Effects, which facilitates the direct import of 3D scenes for post-production. This unique feature has established C4D as a preferred tool in the motion graphics and broadcast design industries, particularly for title sequences and short-form visual content.

This capstone project involves procedural city generation, which demands a system capable of handling rule-based layouts, parametric control, and dynamic simulation of urban elements. Houdini's procedural paradigm and node-based workflow offer a scalable and flexible approach to constructing large-scale city environments with minimal manual intervention. Its ability to handle complex systems like road networks, building variations, and environmental effects makes it a technically suitable and efficient tool for the goals of this project.

Software	**Strengths**	**Limitations**	**Ideal Use Cases**
Blender [29]	Free & open-source; full production pipeline; Python scripting	Limited industry adoption in major studios	Indie films, game assets, general 3D workflows
Maya [30]	Industry standard; advanced animation tools	High cost; steep learning curve	Film, AAA games, character animation
Houdini [26]	Procedural VFX; node-based flexibility; scalable simulations	Complex interface; weak traditional modeling	VFX, procedural city generation, simulation-heavy tasks
Cinema 4D [31]	Easy to learn; AE integration; motion graphics	Limited in VFX complexity	Broadcast design, motion graphics, short films

Table 2.2: Comparison of 3D Software Tools: This table evaluates four widely-used 3D software tools—Blender, Maya, Houdini, and Cinema 4D—based on their strengths, limitations, and ideal use cases. Strengths highlight key features and capabilities, limitations address potential drawbacks, and ideal use cases suggest optimal applications in industries such as film, gaming, and visual effects (VFX), aiding users in selecting the most suitable tool for their specific project needs.

2.2.2 Unreal Engine as the Simulation

Introduction to Unreal Engine 5

Unreal Engine, developed by Epic Games, is a powerful and versatile real-time 3D game engine widely adopted across various industries, including gaming, film, television, architecture, and simulation. Renowned for its high fidelity, flexibility, and open-source accessibility, Unreal Engine empowers creators to build next-generation interactive experiences and photorealistic visuals with a robust suite of tools [32].

Figure 2.3: Comprehensive Overview of the Unreal Engine Interface for Developing The Driving Simulations [33]

Originally released in 1998 to support a first-person shooter game for PC, Unreal Engine has evolved significantly and now supports a wide range of platforms, including PlayStation 4 and 5, Xbox One, Xbox Series X, Nintendo Switch, and mobile and desktop systems. The engine is written in C++, offering developers the flexibility to implement complex game logic and performance optimizations. In addition to C++, Unreal Engine also includes a visual scripting system called *Blueprints*, which allows developers to prototype and implement game functionality through a node-based interface—ideal for artists, designers, and those who prefer visual programming paradigms [32].

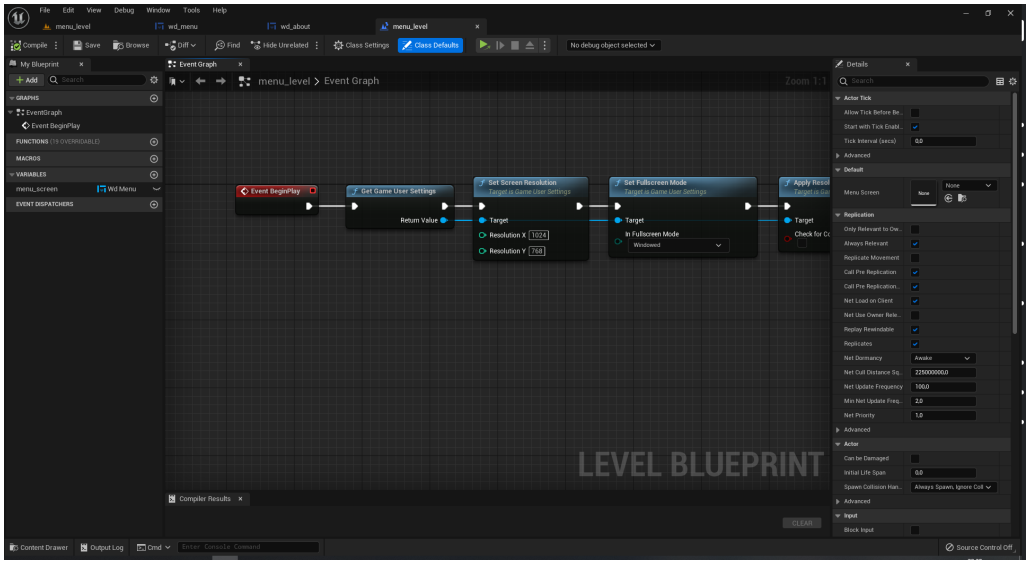


Figure 2.4: Visual Scripting System (Blueprints) in Unreal Engine 5 used to define logic for a virtual environment. Blueprints enable designers and developers to implement gameplay mechanics, object interactions, and environmental behaviors without writing code [34]

Unreal Engine 5 (UE5), the latest version of the engine, comes with advanced features such as *Nanite* (virtualized micropolygon geometry) and *Lumen* (real-time global illumination), making it particularly suitable for creating high-fidelity environments and immersive experiences in real-time.

Figure 2.5: A visual demonstration from Fortnite, which serves as a real-world implementation of Unreal Engine 5.1 technologies, including Nanite, Lumen, Virtual Shadow Maps, and Temporal Super Resolution (TSR). This image illustrates the impact of these advancements on visual fidelity, dynamic lighting, and overall environmental realism in large-scale interactive experiences. These features enable real-time rendering of high-poly assets and physically accurate global illumination, representing a significant leap in game engine capabilities [35]

Integration of Houdini and Unreal Engine 5

The integration between Houdini and Unreal Engine 5 (UE5) is chiefly enabled through the Houdini Engine plugin, which allows procedural assets generated in Houdini to be seamlessly imported and edited within the UE5 environment. This integration empowers developers to utilize Houdini's advanced node-based procedural generation tools, such as those for cityscapes, terrain modeling, and VFX simulations, alongside UE5's robust real-time rendering and interactive features. More important, assets transferred from Houdini retain their parametric properties, enabling real-time adjustments to elements like scale, layout, and material variation directly within UE5, thereby eliminating the need to return to Houdini for modifications. This functionality significantly improves workflow efficiency, especially in projects involving iterative design or large-scale, dynamic environments. Additionally, complex simulations, including fluid dynamics and destruction effects, can be pre-baked in Houdini and imported into UE5 using optimized formats such as geometry caches, Niagara particle systems, or flipbooks, depending on the required balance between performance and visual fidelity. This procedural asset pipeline facilitates rapid prototyping, real-time customization, and high-end visual quality, making it particularly well-suited for the goals of this capstone project. A detailed comparison of Houdini's integration across different game engines is presented in Table 2.3.

The process of transferring assets from Houdini to Unreal Engine is mentioned, specifically exporting building elements from Houdini using ROP Geometry Output nodes as .obj files. Notes that while both software packages use meters as a base unit, there were discrepancies in the .obj export file, which saved units in centimeters. Additionally, Houdini uses a Y-up coordinate system, while Unreal uses a Z-up system. These differences required adjustments on import into Unreal Engine, specifically importing the objects at 100 times scale and rotating them to match the Z-up coordinate system.

Feature / Engine	Unreal Engine [36]	Unity [37]	Godot [38]
Plugin Support	Official Houdini Engine plugin with full support	Official plugin available, moderately supported	No official plugin; limited to manual export/import
Procedural Asset Integration	Full parametric control via Houdini Engine	Partial parameter exposure via plugin	Manual workflow only (e.g., baked meshes, FBX, VDB)
VFX and Simulation Support	Excellent (supports geometry cache, flipbooks, Niagara, etc.)	Limited; external tools required for complex simulations	Poor; no native support for high-end VFX
Real-Time Editing	Live updates inside UE5 editor via plugin	Possible but more limited; often slower	Not supported without external scripting
Performance Optimization Tools	LOD generation, instancing, proxy geometry supported	Somewhat supported via plugin	Manual optimization required
Use Cases in Industry	AAA games, virtual production, VFX	Mobile/indie games, interactive apps	Experimental/indie projects
Learning Curve	Steep, but well-documented with official SideFX resources	Moderate; Unity plugin slightly less mature	High, due to lack of automation and plugins

Table 2.3: Comparative Analysis of Houdini Integration in Unreal Engine 5, Unity, and Godot for Procedural Content Generation in Simulation Environments

Despite its higher hardware requirements compare to its older version, Unreal Engine 4 [39], Unreal Engine 5 (UE5) presents a substantial advantage for developing traffic simulations due to its advanced rendering technologies, optimized performance tools, and support for large-scale, realistic environments. The core features of UE5—such as Lumen for dynamic global illumination and Nanite for virtualized geometry—enable the creation of visually compelling and physically accurate simulations, which are crucial for both scientific validity and stakeholder engagement. The Table 2.4 below outlines key comparative aspects between Unreal Engine 4 (UE4) and Unreal Engine 5 (UE5), specifically in the context of traffic simulation [40]

In this capstone project, the ability to simulate complex urban environments in real time, with dynamic lighting conditions and high-detail models, allows for more realistic traffic modeling and better assessment of behavioral algorithms, vehicle interactions, and environmental impact. In addition, tools like Blueprints for visual scripting provide an intuitive interface for interdisciplinary collaboration, enabling designers, researchers, and developers to co-develop and iterate efficiently.

To conclude, Unreal Engine 5 represents a significant leap in both graphical fidelity and simulation capability, making it an optimal choice for developing advanced traffic simulations. Its innovative technologies, like Lumen and Nanite, combined with improved performance scaling and

streamlined workflows, provide a powerful platform for academic research and real-world system prototyping. Though it necessitates more capable hardware, the gains in realism, development speed, and simulation scalability justify the investment, particularly for a forward-looking thesis project.

Category	Unreal Engine 4 (UE4) [39]	Unreal Engine 5 (UE5) [36]
Lighting System	Pre-baked lighting, requires lightmap UVs	Lumen provides fully dynamic global illumination with real-time lighting adjustments
Geometry Rendering	Manual LOD generation and polycount management	Nanite virtualized geometry eliminates LODs and polycount restrictions
Visual Fidelity	High-quality, but more manual setup	Cinematic-level detail with minimal performance loss
Performance Optimization	Requires manual optimization for high frame rates	Temporal Super Resolution (TSR) ensures smooth rendering at high fidelity
World Building	Manual world streaming and partitioning	World Partition enables scalable, seamless open-world design
Material System	PBR-based only, stylized effects require workarounds	Advanced Material Layering Pipeline, parametric materials like realistic water/glass
Tooling & Workflow	Blueprint scripting, basic procedural tools	Enhanced Blueprints, MetaSounds, and MetaHumans for creative flexibility
Simulation Use Case Fit	Capable but requires more manual setup for realism	Highly suited for real-time, large-scale traffic and environment simulation
Hardware Requirements	Lower, more accessible	Higher, but compensated by greater rendering efficiency and visual returns

Table 2.4: Comparison Between Unreal Engine 4 and Unreal Engine 5 in the Context of Traffic Simulation

2.3 Semantic Segmentation for Autonomous Driving

Semantic segmentation plays a crucial role in the perception systems of autonomous vehicles by enabling pixel-wise classification of objects within a scene. Over the past decade, semantic segmentation techniques have evolved significantly, beginning with the rise of Convolutional Neural

Networks (CNNs) [41]. The introduction of Fully Convolutional Networks (FCNs) [42] marked a turning point by enabling dense predictions across entire images. Building upon this foundation, numerous CNN-based architectures—such as U-Net [43] and BiSeNet—were developed to enhance both segmentation accuracy and computational efficiency [41].

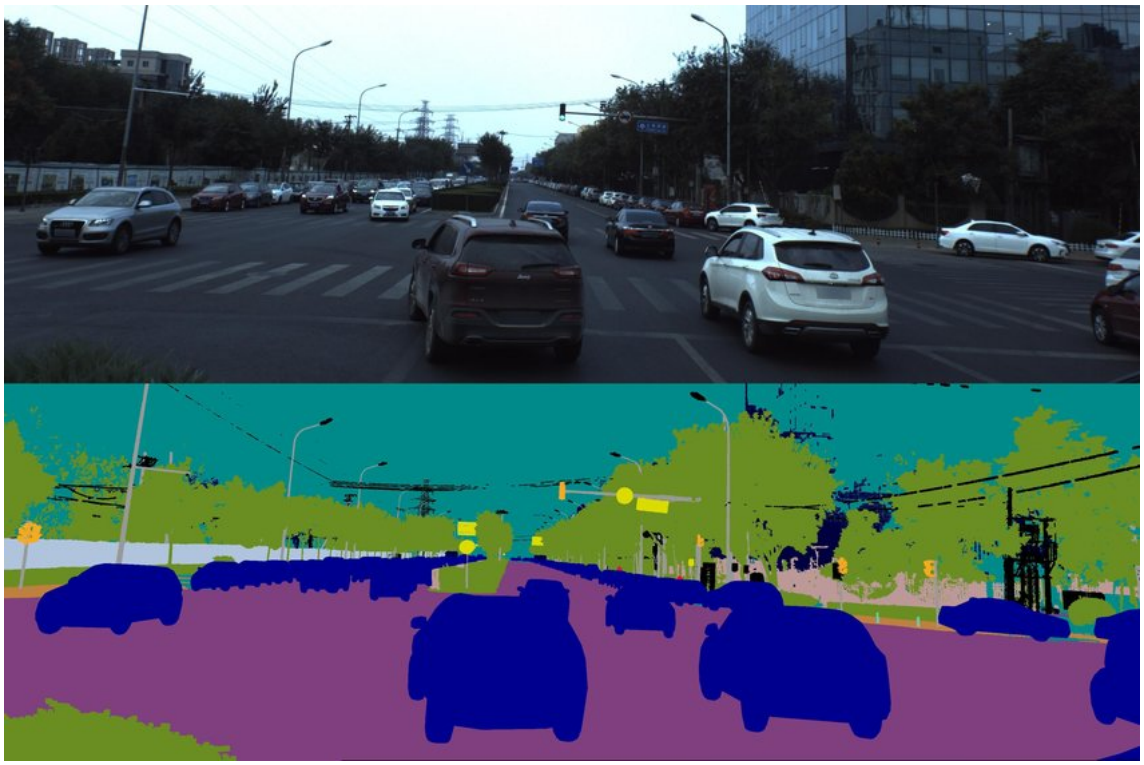


Figure 2.6: An example of semantic segmentation applied to an autonomous driving dataset, where each pixel is classified to identify various objects within the scene, such as roadways, vehicles, pedestrians, and traffic signs, essential for the vehicle's perception system to navigate the environment safely [44]

In recent years, the emergence of Transformer-based models and hybrid CNN-Transformer frameworks has expanded the capabilities of semantic segmentation networks, offering improved global context modeling [41]. However, despite these advances, deploying segmentation models in real-world autonomous driving systems, particularly on resource-constrained embedded devices, presents ongoing challenges. A core requirement in these scenarios is achieving real-time performance, which ensures the vehicle's ability to respond promptly to dynamic environments. Real-time capability is commonly assessed in terms of Frames Per Second (FPS), with models required to maintain a high FPS without compromising segmentation accuracy [5].

Accuracy in semantic segmentation is typically quantified using the mean Intersection over Union (mIoU), which averages the Intersection over Union (IoU) scores across all semantic classes [5]. For each class, IoU is calculated as the ratio of True Positives (TP) to the sum of True Positives,

False Positives (FP), and False Negatives (FN). Achieving a high mIoU while sustaining real-time inference speed remains a significant research objective [5].

A key challenge in this field is balancing the trade-off between model accuracy and computational efficiency [41]. High-accuracy models often incur substantial processing demands, making them unsuitable for embedded systems. Conversely, lightweight models may sacrifice segmentation precision. To address this, various model optimization techniques have been explored, including network pruning, quantization, knowledge distillation, and Neural Architecture Search (NAS) [41]. A notable example is the FasterSeg model, which demonstrates real-time performance on embedded platforms like the NVIDIA Jetson AGX Xavier. By combining NAS and knowledge distillation, FasterSeg achieves a favorable balance between inference speed and accuracy [5].

Datasets such as Cityscapes, CamVid, and ADE20K are widely adopted benchmarks in semantic segmentation research, offering diverse urban scenarios essential for training and evaluation. These datasets facilitate robust comparison and validation of model performance under varying environmental conditions [41].

Recent studies have also investigated the influence of input perspective on segmentation outcomes. Comparisons between traditional first-person views and transformed bird's-eye view (BEV) inputs suggest that models like FasterSeg can achieve comparable mIoU and FPS across perspectives when evaluated on the same embedded hardware [5]. This indicates that BEV representations, which are particularly useful for spatial reasoning in autonomous driving, can be incorporated without compromising real-time feasibility.

Semantic segmentation has undergone substantial advancement through the development of various deep learning architectures. Among them, DeepLabV3 [45] is considered a state-of-the-art model, particularly for tasks requiring accurate delineation of object boundaries. Below is a comparison between DeepLabV3 and other widely used semantic segmentation models, namely FCN (Fully Convolutional Networks) [42], U-Net [43], and SegNet [46].

2.3.1 Architecture

FCN (Fully Convolutional Networks) [42] was the first CNN-based approach for semantic segmentation. Instead of using traditional fully connected layers, it replaces them with convolutional layers and uses deconvolution (or upsampling) to make pixel-wise predictions. While FCN is straightforward and efficient, it has difficulty accurately capturing fine object boundaries, which

can affect the quality of the segmentation in more complex images.

U-Net [43] originally designed for biomedical image segmentation, U-Net uses a symmetric encoder-decoder architecture with skip connections. These skip connections help retain spatial information that might otherwise be lost during downsampling, making U-Net especially effective for small datasets. Its structure enables it to deliver excellent segmentation performance, particularly when working with limited data.

SegNet [46] follows a similar idea to U-Net, but it takes a slightly different approach by using pooling indices from the encoder for non-linear upsampling in the decoder. This design choice helps reduce memory usage, but it comes at the cost of some segmentation detail compared to U-Net, especially in more intricate areas of an image.

DeepLabV3 [45] introduces atrous (or dilated) convolution, allowing the model to increase its receptive field without losing spatial resolution. Its Atrous Spatial Pyramid Pooling (ASPP) module is key to capturing multi-scale contextual information, which significantly enhances segmentation accuracy. This makes DeepLabV3 particularly powerful for complex and cluttered scenes, where the model needs to understand objects at various scales and resolutions.

2.3.2 Multi-Scale Context Aggregation

FCN [42] lacks an explicit mechanism for multi-scale feature aggregation, which can limit its performance on images containing objects of varying sizes. In contrast, U-Net [43] excels in multi-scale learning through its skip connections, enabling it to capture both low-level and high-level features effectively. SegNet [46], while similar to U-Net in utilizing skip connections, is less efficient at aggregating context due to its simpler upsampling mechanism, which limits its ability to capture fine-grained spatial details. DeepLabV3, on the other hand, stands out for its advanced multi-scale feature extraction, particularly through its Atrous Spatial Pyramid Pooling (ASPP) module. This module aggregates contextual information across multiple scales, and the addition of global average pooling further enhances DeepLabV3's ability to process large areas of the image, making it highly effective in complex segmentation tasks.

2.3.3 Boundary Refinement

FCN [42] & SegNet [46] both face challenges in accurately delineating object boundaries, which can result in suboptimal segmentation, especially in complex scenes with intricate structures.

In contrast, U-Net [43] improves boundary localization through its skip connections, allowing it to capture fine details and enhancing its ability to segment objects with precision. DeepLabV3 [45], however, takes boundary precision to the next level by leveraging atrous convolution, which expands the receptive field while preserving spatial resolution. This approach allows for better capture of contextual information at multiple scales. Moreover, the removal of DenseCRF post-processing—used in earlier versions of DeepLab—further improves boundary accuracy by allowing the network to produce more precise segmentation maps directly, without the need for additional refinement steps.

2.3.4 Performance and Applications

FCN [42] is a lightweight model that performs effectively in real-time applications but tends to struggle with accuracy in more complex segmentation tasks. U-Net [43], on the other hand, excels in medical image segmentation, particularly when working with small datasets that demand high precision. SegNet [46] offers a memory-efficient solution, making it well-suited for embedded systems, striking a balance between segmentation accuracy and computational efficiency. However, DeepLabV3 [45] stands out due to its superior performance on large-scale datasets like PASCAL VOC and Cityscapes. This makes it especially well-suited for challenging applications, such as autonomous driving and urban scene segmentation, where both accuracy and scalability are critical—particularly in the context of this project.

2.3.5 Semantic Segmentation Using DeepLab Series

Progression of Semantic Segmentation Networks — The DeepLab Series (V1 to V3+)

Semantic segmentation is a fundamental task in computer vision that involves assigning a class label to each pixel in an image. This enables the precise delineation of objects and regions, which is essential for applications such as autonomous driving, medical imaging, and augmented reality. With the advent of deep learning, convolutional neural networks (CNNs) have become the dominant approach in semantic segmentation tasks. A notable example is the Fully Convolutional Network (FCN) [42], which restructured traditional classification networks to output per-pixel predictions.

Building upon the foundation laid by FCNs [42], the DeepLab series of models introduced by Google Research has played a significant role in advancing semantic segmentation techniques. This

section provides an in-depth exploration of the architectural evolution and innovations introduced in the DeepLab models, from DeepLab V1 [47] through DeepLab V3+ [48].

DeepLab V1 [47]: Introducing Atrous Convolution and CRFs

DeepLab V1 [47] was introduced as a modification of the FCN framework, incorporating key innovations to improve segmentation accuracy and spatial resolution. It adopted the VGG-16 architecture as the backbone for feature extraction and introduced the following enhancements:

Recognizing the need to balance global context with fine-grained details, DeepLab V1 [47] introduced multi-scale feature aggregation. Features were extracted from multiple intermediate layers and combined to form the final prediction map. This approach allowed the model to capture both local and contextual information, which is critical in semantic segmentation.

One of the core contributions of DeepLab V1 [47] was the use of atrous convolution, also known as dilated convolution. Traditional convolutional layers have limited receptive fields, which can hinder the capture of global context. Atrous convolution addresses this by inserting ”holes” between the kernel elements, thereby enlarging the receptive field without increasing the number of parameters or the computational cost. For example, a 3×3 filter with a dilation rate of 2 effectively covers a 5×5 region. This innovation enabled the model to retain a higher spatial resolution in the feature maps, even in deeper layers, without resorting to aggressive downsampling.

To further refine the segmentation boundaries, DeepLab V1 [47] incorporated a fully connected CRF as a post-processing step. CRFs model the spatial dependencies between pixels and encourage label consistency among nearby pixels with similar appearance. This helped sharpen object boundaries and correct misclassifications around edges. However, the CRF module is computationally expensive and not end-to-end trainable, limiting its utility in later models.

DeepLab V2: Atrous Spatial Pyramid Pooling (ASPP)

DeepLab V2 [49] retaine the architectural elements of V1 but introduced Atrous Spatial Pyramid Pooling (ASPP) to address scale variance in object representation. ASPP extends the idea of spatial pyramid pooling by applying multiple parallel atrous convolutions with different dilation rates. This design enables the model to capture multi-scale context more effectively. Specifically, four branches with varying atrous rates are used to probe image features at different scales, which are then concatenated and passed through subsequent layers. This module allowed the model to

become invariant to object scale, a critical factor in real-world image segmentation where objects may appear at different sizes.

DeepLab V3 [45] and V3+ [50]: Enhancing Encoder-Decoder Structure

DeepLab V3 [45] further refined the ASPP module by incorporating image-level features and batch normalization. The inclusion of a global average pooling branch enhanced the model's ability to incorporate global context, complementing the multi-scale features captured by ASPP. Batch normalization improved convergence during training and regularized the model, resulting in higher accuracy. Additionally, DeepLab V3 discarded the use of CRFs, relying solely on network-based learning to predict accurate segmentation maps.

Develop from Deeplab V3, DeepLab V3+ [50] introduced a lightweight decoder module to recover spatial details that are often lost during encoding. This architecture bridges the gap between coarse segmentation outputs and the fine spatial resolution required for boundary delineation. Key features of DeepLab V3+ [50] include the encoder, which combines DeepLab V3's Atrous Spatial Pyramid Pooling (ASPP) and backbone network to capture rich contextual information. Additionally, the decoder refines segmentation predictions by gradually recovering image details using features from early encoder layers, leading to more precise object boundaries and better segmentation results. The introduction of the decoder significantly improved performance on challenging datasets such as PASCAL VOC and Cityscapes, demonstrating the efficacy of combining semantic and spatial information.

Table 2.5: A comparative overview of innovations introduced in successive versions of the DeepLab semantic segmentation architecture.

Version	**Key Innovations**
V1 [47]	Atrous Convolution, Fully Connected CRF
V2 [49]	Atrous Spatial Pyramid Pooling (ASPP)
V3 [45]	Image-Level Features, Enhanced ASPP, Removal of CRF
V3+ [50]	Encoder-Decoder Architecture, Feature Refinement

In this study, I employ the DeepLabV3 architecture with a ResNet-50 backbone—an established CNN-based model known for its strong semantic segmentation capabilities and effective use of atrous spatial pyramid pooling (ASPP) for multi-scale context aggregation. While not explicitly optimized for real-time applications on embedded devices, DeepLabV3+ResNet50 offers a solid benchmark for segmentation accuracy and robustness across different viewpoints. This makes it suitable for comparative evaluation, especially when analyzing performance across standard and transformed input perspectives.

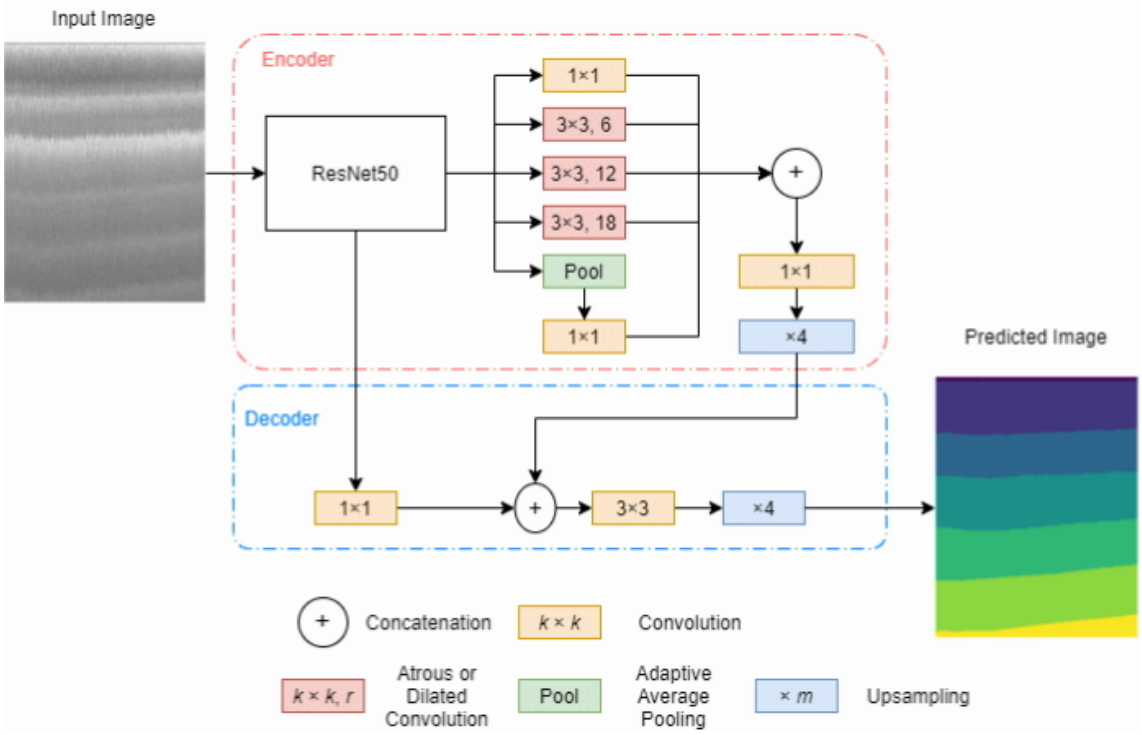


Figure 2.7: Model architecture based on DeepLabv3 with a ResNet-50 backbone, adapted from [51]. This architecture employs atrous spatial pyramid pooling (ASPP) for multi-scale context aggregation and utilizes the ResNet-50 encoder to extract hierarchical features

In summary, semantic segmentation remains a pivotal component in autonomous driving systems. The need for high performance on embedded hardware has led to the development of optimized models and new strategies for efficient deployment. The generation and use of synthetic datasets, alongside thorough benchmarking using models like DeepLabV3 and FasterSeg, constitute a promising direction for advancing semantic segmentation in real-world autonomous driving applications.

2

3

Methodology

Contents

This section outlines the comprehensive pipeline developed for integrating 3D modeling, game logic implementation, image generation, and AI-based image segmentation within a simulated environment. The workflow consists of several interconnected stages, beginning with asset creation and culminating in real-time control using a trained computer vision model.

3

3.1 System Architecture

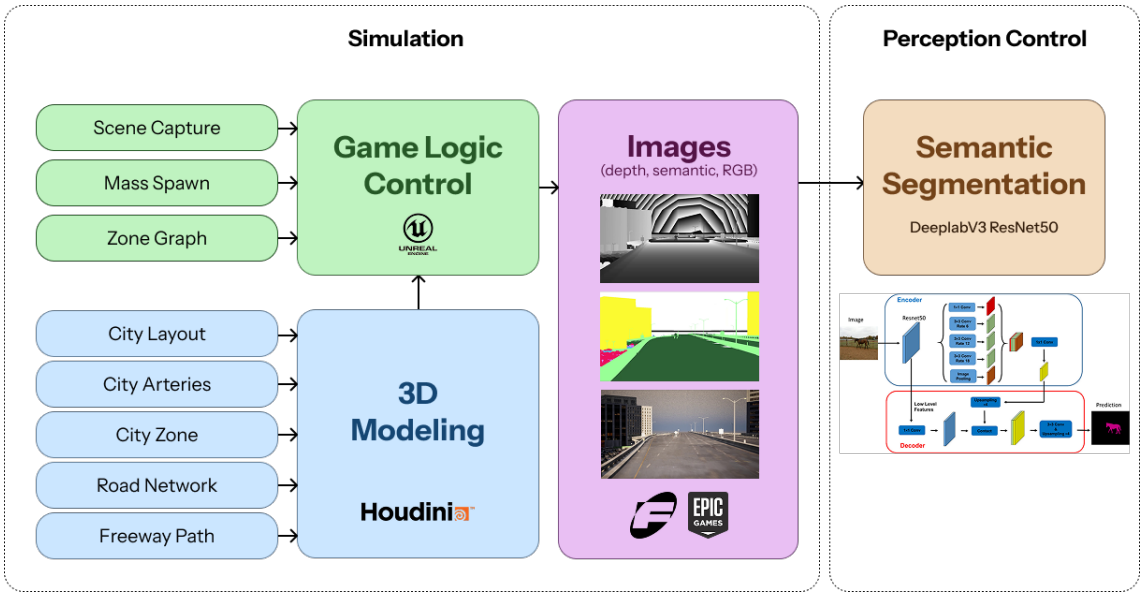


Figure 3.1: Pipeline for developing and testing autonomous vehicle control strategies within a realistic simulated environment. The simulation generates multi-modal image data, which is then processed by an AI object detection system. The detected objects inform a rule-based control system that dictates the autonomous vehicle's actions within the simulation, creating a closed-loop system for testing and validation.

The pipeline begins with 3D asset modeling in Houdini, a procedural generation tool ideal for constructing the city's procedural content in this project. Once the modeling is complete, the assets are exported and then imported into Unreal Engine 5 (UE5). Within UE5, the assets are integrated into an interactive simulation environment.

After importing, the game logic is configured in UE5 to simulate various components of an urban environment. This includes modular controls for player navigation, vehicle driving, traffic light operation, pedestrian behavior, and spawning desired numbers of transportation. These controls form the interactive foundation of the virtual environment. Following the game logic setup, the system transitions into a data generation phase using EasySynth, a plugin for synthetic image export. This module outputs multiple image types from the simulation, including: RGB images for visual training, depth maps for spatial understanding, semantic segmentation masks for class-based labeling.

These synthetic datasets are then used to train a semantic segmentation model, specifically fine-tuned on ten annotated classes relevant to urban traffic scenarios. This training step equips the

model with the ability to perform real-time segmentation of the synthetic environment. DeepLabV3, a state-of-the-art architecture for semantic segmentation, is employed to accurately classify and delineate the various objects in the scene, such as roads, vehicles, pedestrians, and traffic signs, which are critical for autonomous driving tasks.

Finally, the DeepLabV3 model is deployed to interpret and segment images in the game environment. Its outputs help the system recognize the layout of a scene, including road boundaries, lanes, and dynamic elements like moving pedestrians or other vehicles.

The structured pipeline ensures that each stage, from asset creation to AI-based control, is modular, reproducible, and adaptable to different simulation objectives. The use of procedural tools like Houdini, real-time engines like UE5, and deep learning models enables the simulation of highly dynamic environments with control over visual and behavioral parameters.

3.2 Sub-system

3.2.1 Procedural City Generation in Houdini

In this project, Houdini by SideFX was utilized to generate a procedural virtual city environment. In this project, *Houdini Apprentice version 19.5.1112 (Python 3.7)*, released on September 24, 2024, was employed to procedurally generate a virtual city environment using SideFX's node-based 3D software.

The project is set up with City Sample Fab game assets by Epic Launcher [52]. This asset library provides HDA for the city procedural, the HDA library includes:

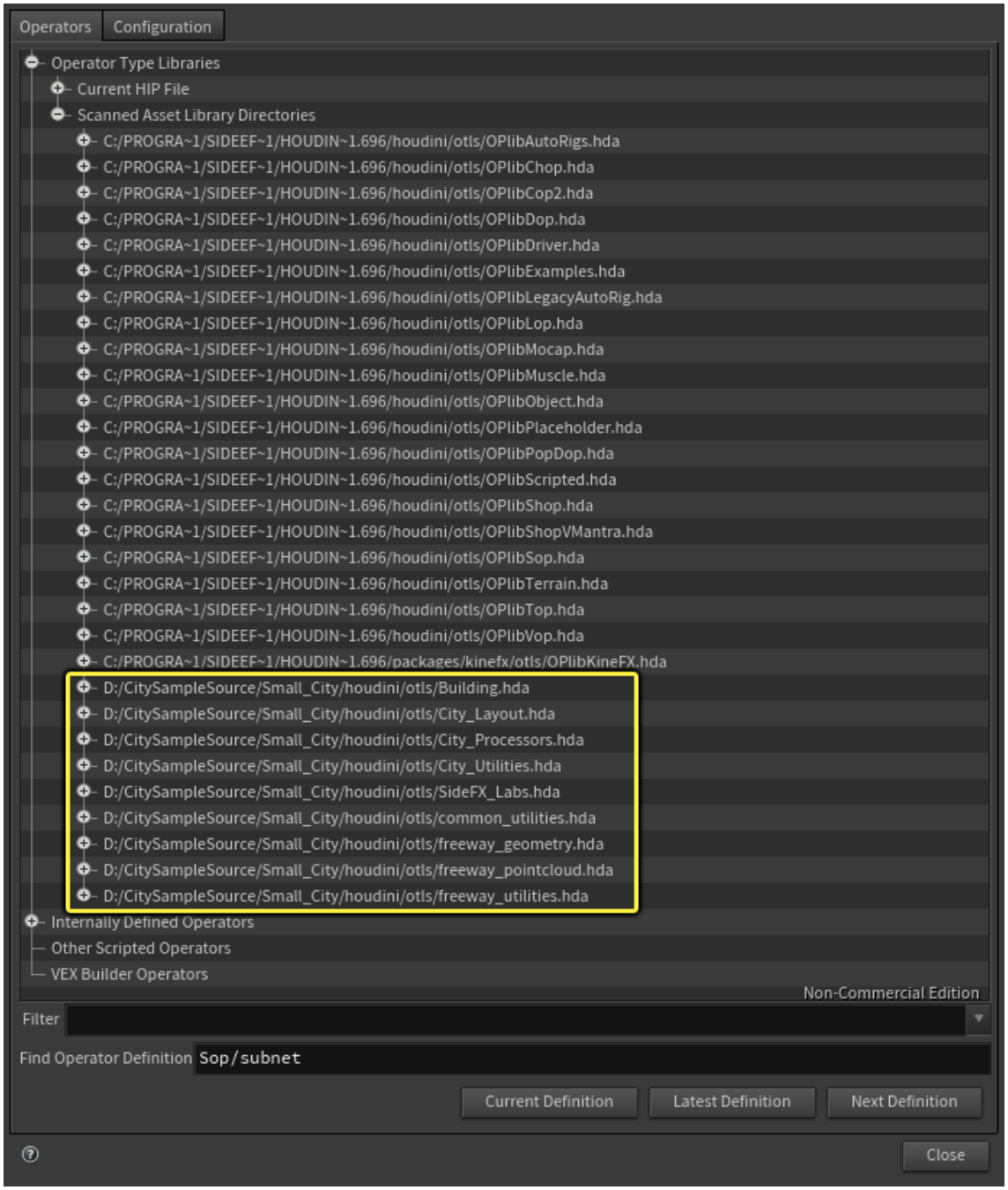


Figure 3.2: The HDA provides user-configurable parameters to control zoning, building density, block sizes, and procedural placement logic. By embedding the procedural generation rules directly into the asset, it enables real-time city layout generation and component instancing within Unreal Engine using the City Sample's existing blueprint infrastructure [53]

The procedural generation of the city began with the definition of the City Layout, using Houdini's node-based architecture. This step involved configuring the spatial structure of the urban environment through a series of parametric tools encapsulated in the City Layout Houdini Digital Asset (HDA).

Initially, the overall shape of the city was determined using a curve node, which defines the base perimeter, represented visually as the gray area in the editor viewport. The City Layout HDA enabled the procedural design of foundational urban parameters, including:

- Drawing and modifying arterial road splines, which serve as the main roads or traffic veins of the city.
- Configuring city block dimensions and zoning regions, such as commercial, residential, and industrial sectors.
- Customizing additional attributes like average building height, lot size, block depth, and zone density.

The City Arteries—depicted as an overlaid network of red, blue, and green lines above the gray base—were defined on top of the layout. This system represents the skeletal infrastructure of the city, comprising both roadways and designated building regions. Artery density, road hierarchy, merging behavior of adjacent blocks, and other parameters can be interactively modified through the $city_network$ and $road_network$ panels in the right-hand sidebar.

Subsequently, the City Zone node was introduced to further refine the functional distribution of space. These zones are visually identified as two white segmented areas within the editor. Using additional curve nodes, the shape and extent of each zone were established. Their height and shape were then adjusted using the $zone_attributes$ panel.

After the layout and zoning were finalized, the Road Network HDA was employed to procedurally generate a connected system of roads. This included secondary roads, intersections, and branching paths based on the arterial framework. To enhance realism and complexity, the Freeway Generator HDA was then used to create elevated highways that seamlessly integrate with the surrounding terrain and existing roadways, including proper curvature, exits, and structural spacing.

Finally, all the generated components—arterial splines, zones, road networks, and freeways—were integrated into a single procedural model. This was achieved by connecting all output nodes into a merge node (specifically, $merge1$ in the node network), which consolidated them into a unified structure. The result was then passed into the $layout1$ node, which serves as the input for downstream generation processes, including exporting to Unreal Engine for simulation purposes.

Once the city model was finalized, it was exported using Houdini's Procedural Dependency

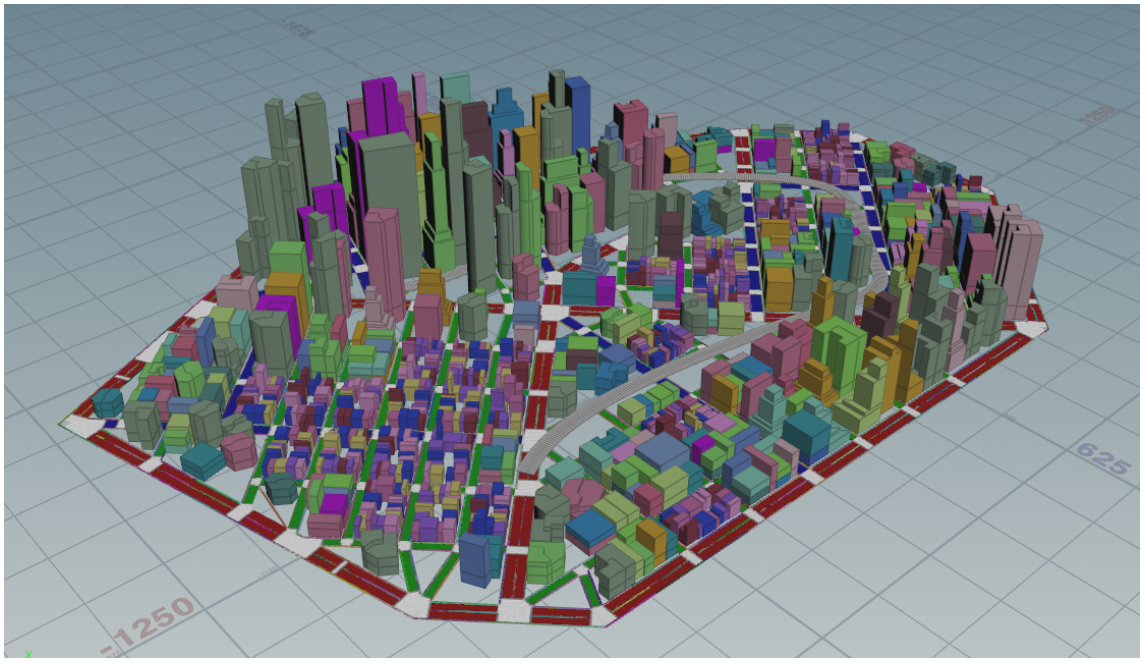


Figure 3.3: City procedural generation system developed in Houdini. The system uses rule-based modeling and parameter-driven workflows to generate complex urban environments with customizable road networks, building layouts, and city infrastructure. The output reflects a scalable and modular design suitable for real-time applications such as games or simulations. [52]

Graph (PDG) to streamline and accelerate the data processing workflow. The PDG framework enables efficient task scheduling and parallel execution of procedural tasks, significantly improving the overall export speed for complex scenes.

The export process was initiated through the *task graph interface* located in the left panel of the Houdini workspace. The sequence of steps included:

- *Processing the City Base* – establishing the foundational geometry and zoning structure.
- *Executing PDG with Building Generation* – generating and populating city blocks with procedural buildings.
- *Processing City Furniture* – adding secondary assets such as streetlights, traffic signs, and environmental props.
- *Exporting PBC* (Point Cloud Alembics) – exporting the final scene elements as point cloud data and geometry files.

The result of this pipeline was a structured set of output files and directories containing point cloud representations and the full structural layout of the city.

Following the export, the scene was imported into Unreal Engine 5 (UE5). This step required additional debugging and optimization, including adjustments to graphical fidelity and resolution of compatibility issues between the specific versions of Houdini Engine and UE5. These interventions ensured that the procedural assets were rendered correctly and performed efficiently within the game engine environment.

Figure 3.4: Final 3D product being imported from Houdini to Unreal Engine 5. The export process ensures that the high-quality procedural models, including detailed environments and assets, maintain their integrity when transitioning from Houdini's dynamic simulation environment to Unreal Engine 5's powerful real-time rendering capabilities [52]

3.2.2 Game Components in Unreal Engine 5

The Houdini asset offers a preconfigured foundation for traffic simulation, incorporating key elements such as defined driving lanes, collision-aware geometry, urban structures, lighting systems, and non-player characters (NPCs), including vehicles and pedestrians. This robust setup significantly reduces the need for complex custom logic within Unreal Engine.

The simulation logic applied in this project is relatively straightforward, relying on a modular set of components—such as *player control*, *AI-driven vehicle behavior*, *traffic signal management*, and *pedestrian navigation*. These components integrate seamlessly with the Houdini-generated environment and Epic Games' *City Sample* framework to create realistic and responsive traffic scenarios.

The traffic behaviors can be further customized using MassEntity, which allows control over the number of vehicles in the scene, enabling either sparse or dense traffic, and ZoneGraph, which generates parked vehicles along city streets based on procedural placement data.

For this thesis, only the traffic behavior components are customized. Additional features, such as smart objects and unpredictable pedestrian behaviors, can be explored through the CitySample project from this Epic Games documentation [53].

MassEntity

Unreal Engine's MassEntity [53] framework enables efficient simulation of large-scale AI agents, making it ideal for managing dynamic traffic and crowd behavior in complex urban environments. In the City Sample project, the **Mass Spawner** is used to introduce different types of entities, such as pedestrians, vehicles, and parked cars, based on predefined Mass Entity Definitions that control traits like visuals, behavior, and level of detail.

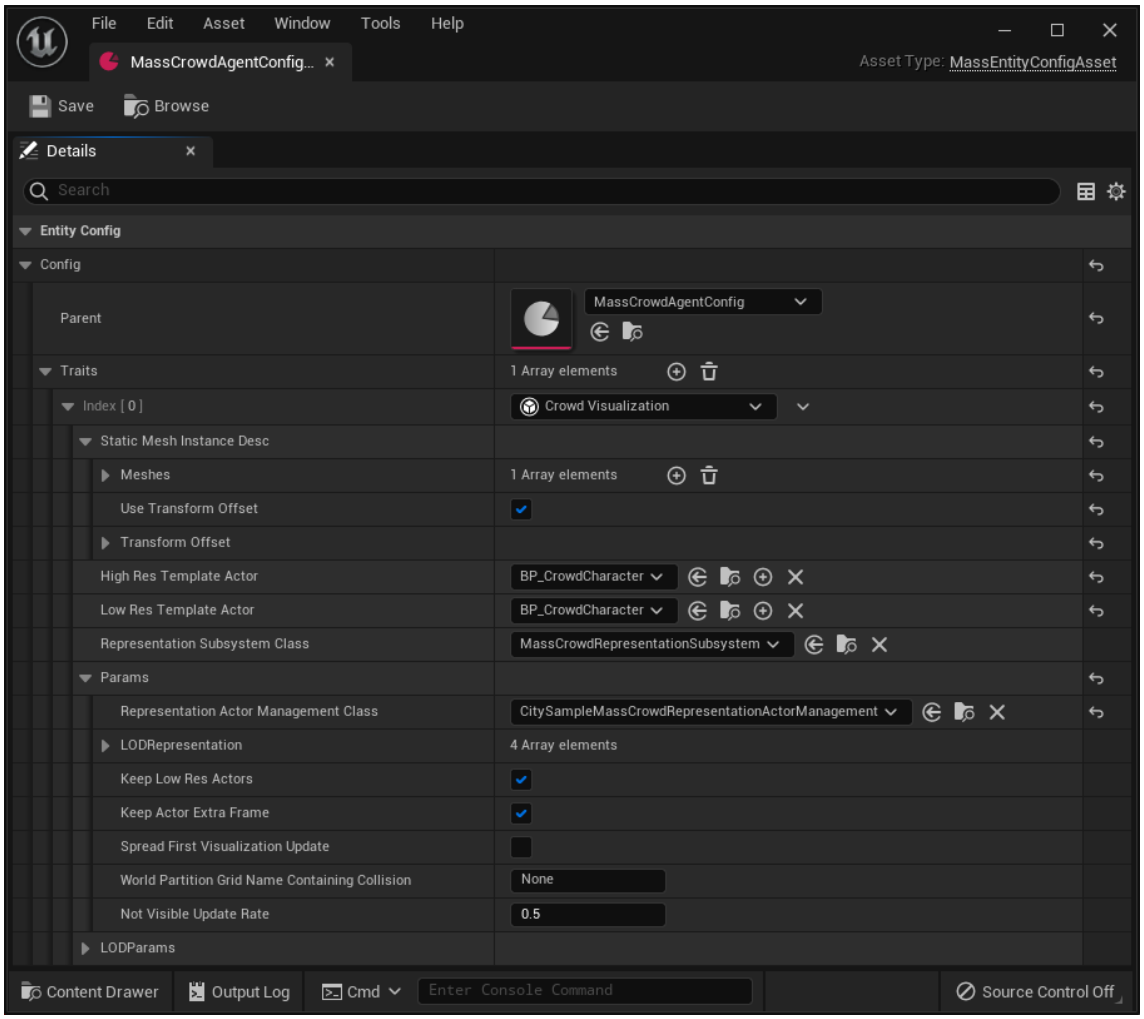


Figure 3.5: Configuration of a MassEntity blueprint called "MassCrowd" in Unreal Engine. The blueprint is designed to efficiently spawn and manage a large number of pedestrians in the scene, utilizing the MassAI framework for high-performance crowd simulation. By configuring the entity parameters and spawning logic, the blueprint enables realistic crowd behavior and interaction, enhancing the dynamic realism of the environment [53]

These entities are distributed using procedural data from Houdini, including ZoneGraphs for moving agents and point clouds for static elements such as parked vehicles. By configuring different Mass Spawner blueprints and assigning appropriate definitions and spawn locations, a wide variety

of traffic scenarios can be simulated. These include congested intersections, pedestrian-heavy areas, and varied driving patterns across different city zones.

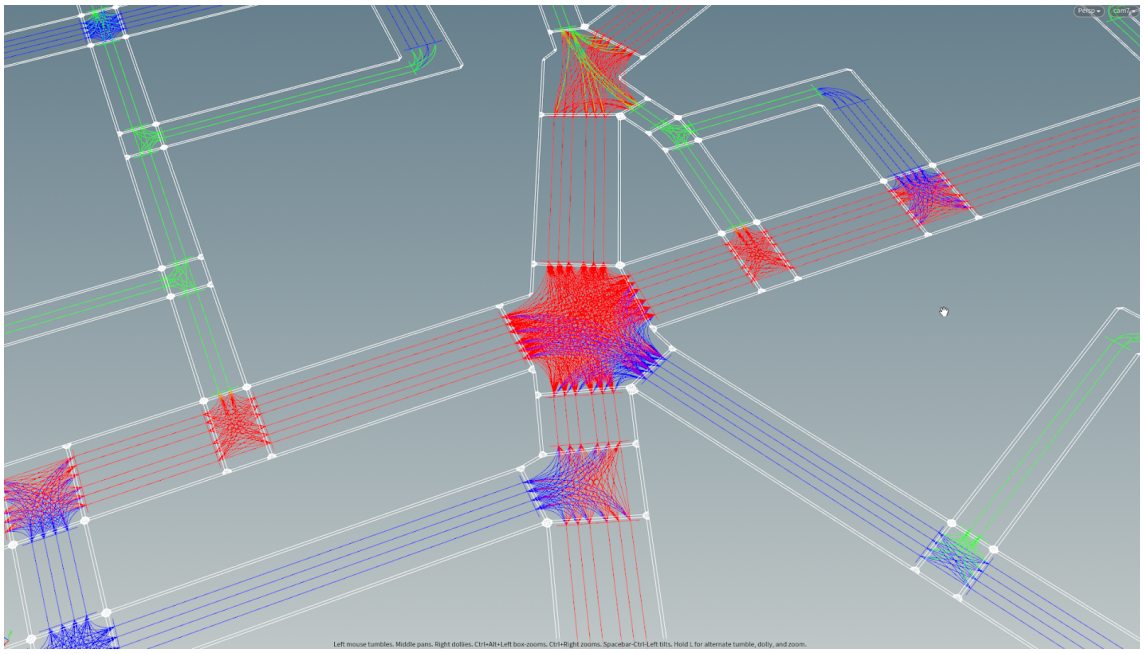


Figure 3.6: Illustrates the generated data used to determine optimal spawn locations for both crowd and traffic entities within the city. By analyzing key factors such as road networks, pedestrian pathways, and traffic flow, the data helps to intelligently position entities in the scene [53]

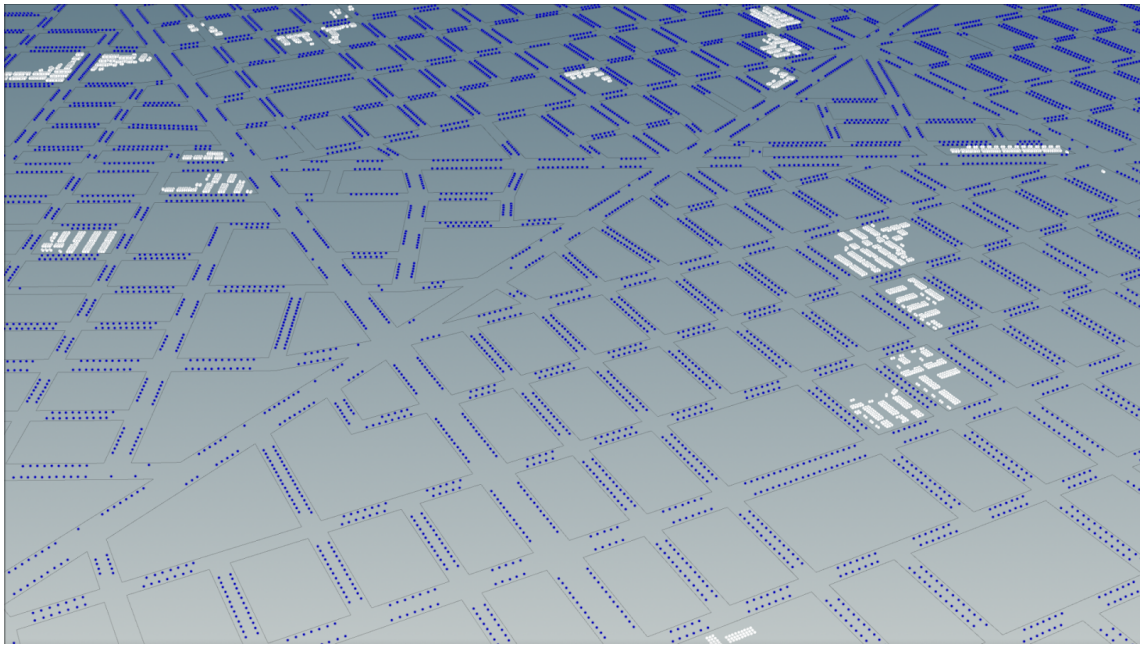


Figure 3.7: The use of point cloud data to identify suitable locations along city streets for spawning parked vehicles. The point cloud analysis provides precise spatial information, allowing for accurate placement of vehicles in parking spots, along curbs, or in designated areas. This technique ensures that vehicles are placed naturally within the urban environment, enhancing the realism and detail of the city simulation [53]

ZoneGraph

The **ZoneGraph system** is a lightweight, design-driven framework used to guide AI behavior through a structured, corridor-based network. It defines navigable paths as point-by-point sequences and supports the assignment of both static and dynamic tags that can influence AI decision-making and movement logic.

Within UE5, Zone Shapes can be added to the scene using the *Place Actors* panel. At the *Big City* level, these components are organized under the *Zone Shapes* folder in the *World Outliner*, demonstrating their role in shaping AI navigation across the environment. For this capstone project, I implement the *Small City Level* to reduce computational cost.

Figure 3.8: ZoneGraph of a street in the Big City Level scene. The ZoneGraph is used to define different zones within the city, such as pedestrian areas, traffic routes, and points of interest. By mapping the street layout into a graph structure, the system allows for optimized navigation, traffic flow management, and AI pathfinding [53]

3.2.3 Data Collection and Processing

Data Collection with Scene Capture 2D and EasySynth Fab

EasySynth [54] is an Unreal Engine plugin designed to streamline the generation of high-quality image datasets using a moving multi-camera rig. It is particularly well-suited for machine learning applications and is accessible to users without requiring any knowledge of C++ or Blueprint

scripting. EasySynth relies solely on native Unreal Engine components, eliminating the need for third-party tools while maintaining compatibility and performance.

The plugin offers a user-friendly interface that facilitates the semantic labeling of in-scene actors. Dataset generation is triggered automatically by rendering a predefined level sequence, with support for a variety of camera post-processing configurations. The outputs in Figure 3.9 generated by EasySynth include:

- A CSV file containing detailed camera poses, including position, rotation, and intrinsic calibration parameters.
- A JSON file that describes the camera rig configuration.
- RGB color images are rendered by default.
- Grayscale depth images.
- Per-pixel surface normal images.
- Optical flow images between consecutive frames.
- Semantic segmentation images with actor-specific labels.

This tool enables efficient and automated creation of annotated datasets, which are essential for training and evaluating computer vision models in tasks such as object detection, segmentation, and depth estimation.

In Unreal Engine 5, the `Scene Capture 2D` component is utilized to simulate camera vision by capturing rendered images from a specified viewpoint. In this project, a `Scene Capture 2D` blueprint was implemented to emulate the perspective of a driver within the virtual environment. A predefined camera path was created, enabling the blueprint to move systematically through the map. As the camera travels along this trajectory, it captures and exports images at regular intervals, effectively replicating the visual experience of a vehicle navigating through the scene. This setup is essential for generating viewpoint-accurate image datasets for training and evaluating computer vision models.

The output of the image capture process consists of three distinct types of images: RGB images, depth images, and semantic segmentation images. These images are exported in a synchronized format, ensuring consistent temporal alignment across all modalities. For each captured frame, the images adhere to a unified naming convention. For example, an RGB image named

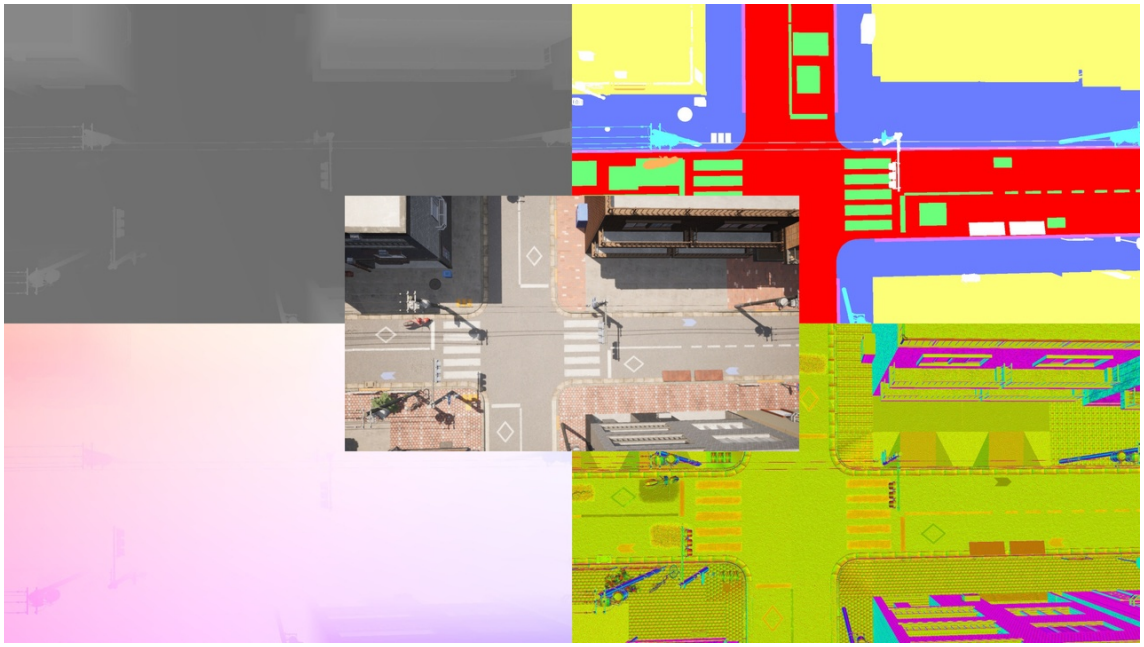


Figure 3.9: Output of the EasySynth plugins, which generate five distinct types of images: Depth, Optical Flow, RGB, Semantic, and Normal images. Each image type serves a specific purpose in enhancing scene realism and aiding in tasks such as object recognition, motion tracking, and environmental depth analysis. The combination of these images provides a comprehensive dataset for training AI models and improving the accuracy of simulations, enabling more immersive and dynamic interactions within virtual environments [54]

`RGB_frame_001.jpg` is accompanied by its corresponding depth image `depth_frame_001.jpg` and semantic segmentation image `semantic_frame_001.jpg`. This structured naming convention facilitates efficient data organization and enables straightforward mapping between modalities during the training and evaluation of computer vision models. The image output of the simulation platform will later be discussed in Chapter 4

Semantic Segmentation Mapping

To enable semantic segmentation of the rendered images, each object class in the synthetic environment was assigned a unique RGB color. This mapping allows the conversion of pixel values in semantic images into corresponding class labels, enabling supervised learning for object detection and segmentation tasks.

The color codes were manually defined using EasySynth [54], a tool integrated within Unreal Engine (UE5) that supports semantic image generation. These RGB values were determined based on the visual distinctiveness and clarity in the rendered outputs and were manually selected by the author to ensure consistent class identification across the dataset.

The following Python dictionary illustrates the mapping from RGB values to semantic class labels:


```
color_to_class = {
    (0, 81, 34): 0,          \# Freeway deck
    (84, 21, 0): 1,          \# Obstacles
    (0, 51, 206): 2,         \# Traffic light
    (142, 255, 145): 3,      \# Decor
    (223, 118, 255): 4,      \# Margin
    (0, 88, 255): 5,         \# Road light sign
    (255, 239, 0): 6,        \# Building
    (255, 0, 86): 7,         \# Car
    (0, 184, 255): 8,        \# Road
    (255, 255, 255): 9       \# Sky
}
```

Each color in this dictionary corresponds to a specific object category present in the virtual city environment. During the segmentation process, EasySynth generates semantic images where each pixel's color matches one of the pre-defined RGB codes. These images are then processed to convert the pixel colors into integer class labels, which are used as ground truth for training semantic segmentation models in the next step of this project.

Dataset Analysis

The dataset exhibits a significant class imbalance. The "*building*" class dominates with 182,598,134 pixels, followed by "*road*" with 113,028,506 pixels. In contrast, "*traffic_light*" and "*obstacles*" are severely underrepresented, with only 1,533,715 and 251,561 pixels, respectively.

The total pixel count across all classes is 425,172,668. This imbalance suggests that machine learning models may be biased toward the overrepresented classes like "*building*" and "*road*," which could degrade performance on minority classes. The large disparity—where "*obstacles*" account for less than 0.06% of the total pixels—highlights the need for corrective measures, such as class weighting, to promote balanced learning across all classes.

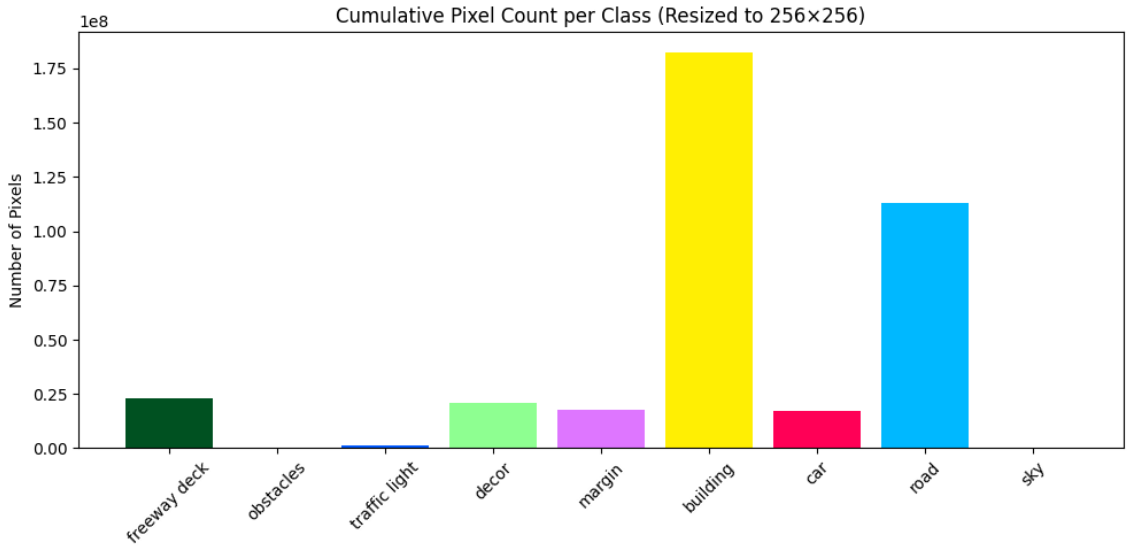


Figure 3.10: Illustration of class imbalance in the dataset, with overrepresented classes like $'building'$ and $'road'$ dominating the pixel count, while underrepresented classes such as $'traffic\_light'$ and $'obstacles'$ contribute a minimal proportion of the total pixels.

Handling Class Imbalance with Cost-Sensitive Learning

To address the class imbalance in the dataset, *Cost-Sensitive Learning* was applied using inverse frequency weights. The weight for each class was calculated based on its relative frequency in the dataset, with underrepresented classes receiving higher weights to give them more attention during training. The inverse frequency for each class was computed using the formula:

$$\text{Weight for class } c = \frac{1}{\text{Frequency of class } c + \epsilon}$$

Where ϵ is a small constant added to prevent division by zero. These inverse frequencies were then normalized so that the highest weight would be set to 1.

The normalized inverse frequency weights for each class are as follows:

Class	**Normalized Weight**
Freeway Deck	0.0109
Obstacles	1.0000
Traffic Light	0.1643
Decor	0.0120
Margin	0.0142
Building	0.0014
Car	0.0145
Road	0.0022
Sky	0.0050

Table 3.1: Normalized Inverse Frequency Weights for Each Class

The *Obstacles* class has the highest weight (1.0000), reflecting its underrepresentation in the dataset. Conversely, the *Building* class, which is the most dominant, has the lowest weight (0.0014), indicating its overrepresentation. By assigning these weights, the model is encouraged to focus more on the minority classes, such as *Obstacles* and *Traffic Light*, which helps improve segmentation accuracy for these underrepresented categories.

Model Evaluation

The performance of the model was assessed using several key evaluation metrics, including accuracy, loss, Intersection over Union*(IoU)*, and *Dice coefficient*, for both the training and validation datasets. These metrics provide insight into the model's ability to correctly segment the images and its generalization capability on unseen data.

During each epoch, the model was trained using the training dataset, and the average loss and accuracy were computed. The training loss indicates how well the model fits the training data, while the training accuracy measures the percentage of correctly predicted pixels. Similarly, after each training epoch, the model was evaluated in the validation data set. The validation loss and accuracy help assess the model's performance on unseen data, providing an indication of how well the model generalizes to new scenarios.

IoU measures the overlap between the predicted segmentation mask and the ground truth mask, serving as an important metric for evaluating the quality of the segmentation. A higher IoU score indicates that the model's predictions are more aligned with the actual objects in the image.

While the Dice coefficient is another metric used to evaluate the similarity between the predicted segmentation and the ground truth. Like *IoU*, higher values of *Dice* indicate better performance, with a maximum value of 1.0 representing perfect overlap between the predicted and true masks.

The metrics were computed and recorded for each epoch, and the model's performance was visualized using loss, accuracy, *IoU*, and *Dice* scores over time. The evaluation is based on the following formulas:

- Formula to evaluate model accuracy:

$$\text{Accuracy} = \frac{\text{Correct Pixels}}{\text{Total Pixels}} \times 100$$

- Formula to calculate IoU:

$$\text{IoU} = \frac{\text{Intersection}}{\text{Union}}$$

- Formula to calculate Dice coefficient:

$$\text{Dice} = \frac{2 \times \text{Intersection}}{\text{Sum of the Sizes of Both Sets}}$$

3

4

Result and Discussion

Contents

4.1 Results

4.1.1 Simulation Platform

The simulation generates high-resolution, detailed images with realistic graphics that closely mimic real-life traffic scenarios. These images are used to train and test the semantic segmentation model, providing a comprehensive view of the urban environment, including roads, vehicles, pedestrians, and other objects of interest.

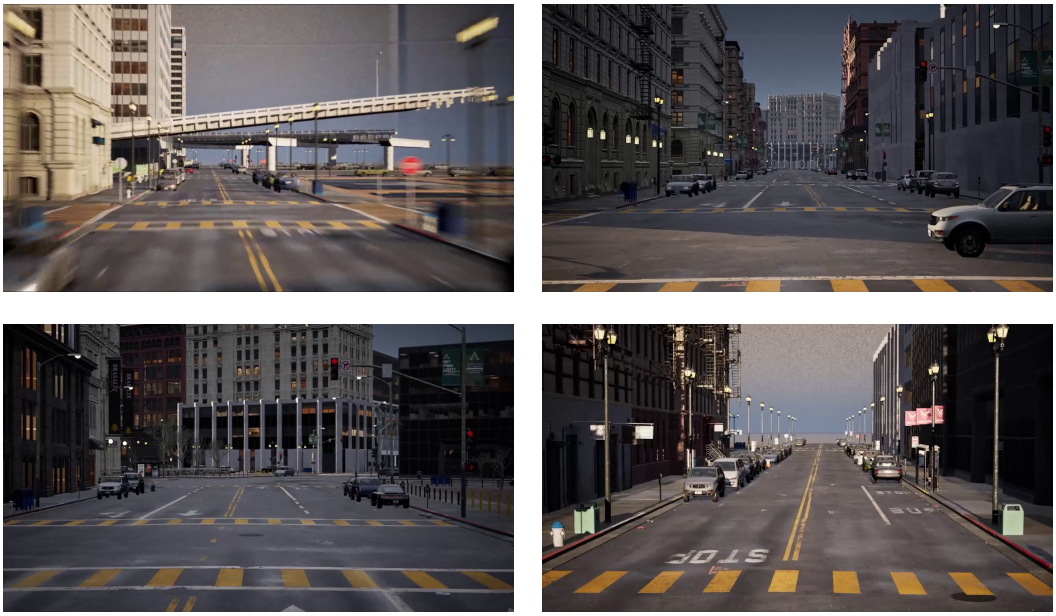
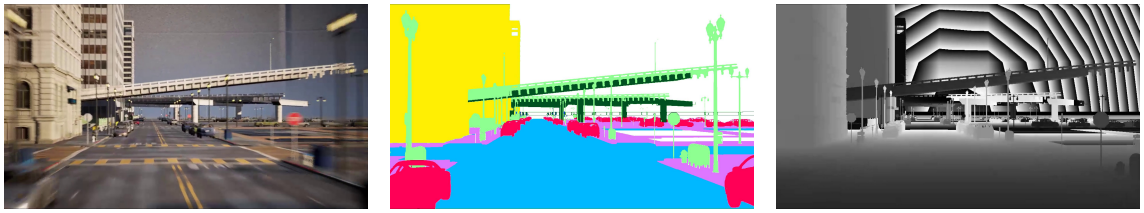


Figure 4.1: High-resolution simulation image showing the detailed urban landscape, capturing the realistic texture and lighting of the environment. This image illustrates the accuracy of the graphics and how the simulation mimics real-life scenarios, providing a suitable setting for autonomous vehicle testing.

From the simulation, I generated three distinct types of synthetic images as part of the autonomous driving pipeline:

- RGB Images: High-resolution, photorealistic frames rendered from the Unreal Engine 5 (UE5) environment, representing real-world road scenes under varying conditions such as lighting, weather, and traffic density.
- Segmentation Masks: Ground-truth semantic masks corresponding to the RGB images, where each pixel is labeled with one of ten predefined urban scene classes (e.g., road, sidewalk, car, pedestrian, building).
- Depth Maps: Per-pixel depth information used for potential future integration with 3D perception and scene reconstruction tasks.



(a) RGB Image (b) Segmentation Mask (c) Depth Map

Figure 4.2: Visualization of synthetic data types generated in the simulation environment: (a) raw RGB images, (b) ground truth segmentation masks, and (c) corresponding depth maps.

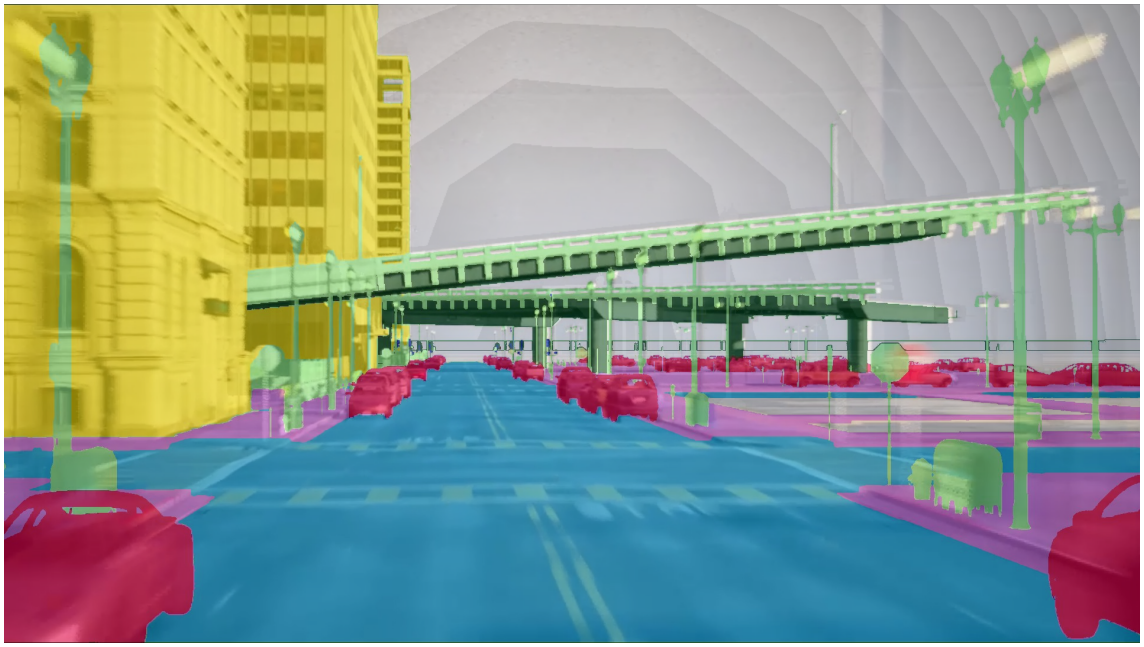


Figure 4.3: Overlapping triplet images showing the RGB image, semantic segmentation mask, and depth map. This visualization demonstrates the relationship between the raw input image, its segmented regions, and depth information in the context of autonomous driving simulations.

The dataset contains a total of 7,374 images, along with a CSV file specifying class color codes in RGB format. Each data sample is represented by a triplet of aligned images: an RGB image, a semantic segmentation mask, and a depth map. Consequently, each image type (RGB, semantic, and depth) includes 2,458 samples. For the purpose of semantic segmentation, the dataset is partitioned into training, validation, and test sets using a 75% / 15% / 15% split.

The dataset can be found at Kaggle Dataset

4.1.2 Semantic Segmentation Result

The performance of the semantic segmentation model was evaluated over 24 epochs using multiple metrics including training and validation accuracy, loss, Intersection over Union (IoU),

and Dice coefficient. The quantitative results from each epoch are summarized in Table 4.1. These metrics provide a comprehensive view of the model's learning behavior and generalization ability.

Epoch	**Train Loss**	**Train Acc (%)**	**Val Loss**	**Val Acc (%)**	**IoU**	**Dice**
1	0.6477	82.28	0.3931	88.78	0.5647	0.6569
2	0.3254	88.78	0.3226	88.75	0.5798	0.6714
3	0.2621	90.39	0.2808	90.42	0.6148	0.7057
4	0.2326	91.28	0.2892	89.82	0.6043	0.6944
5	0.2269	91.38	0.2725	91.10	0.6208	0.7089
10	0.1622	93.72	0.2544	92.02	0.6512	0.7371
15	0.1234	95.27	0.2607	92.85	0.6676	0.7503
20	0.1094	95.80	0.2681	93.77	0.6827	0.7615
24	0.0966	96.34	0.2986	93.93	0.6876	0.7660

Table 4.1: Training and Validation Metrics Across 24 Epochs. This table summarizes the training loss, training accuracy, validation loss, validation accuracy, Intersection over Union (IoU), and Dice coefficient for each selected epoch during model training. The metrics show steady improvement in performance, with notable increases in accuracy and segmentation quality (IoU and Dice) up to around epoch 20. A slight increase in validation loss after epoch 20 may indicate early signs of overfitting, highlighting the need for regularization or early stopping strategies. These results demonstrate the model's learning progression and segmentation performance over time.

As shown in Table 4.1, the model's training accuracy improved steadily from 82.28% in the first epoch to 96.34% by the final epoch, while the validation accuracy improved from 88.78% to 93.93%. Correspondingly, the IoU increased from 0.5647 to 0.6876, and the Dice coefficient improved from 0.6569 to 0.7660.

The trends observed confirm that the model not only learned the segmentation task effectively but also generalized well on the validation dataset. The convergence of training and validation losses indicates minimal overfitting, and the consistent improvement in IoU and Dice scores reflects improved mask prediction quality over time.

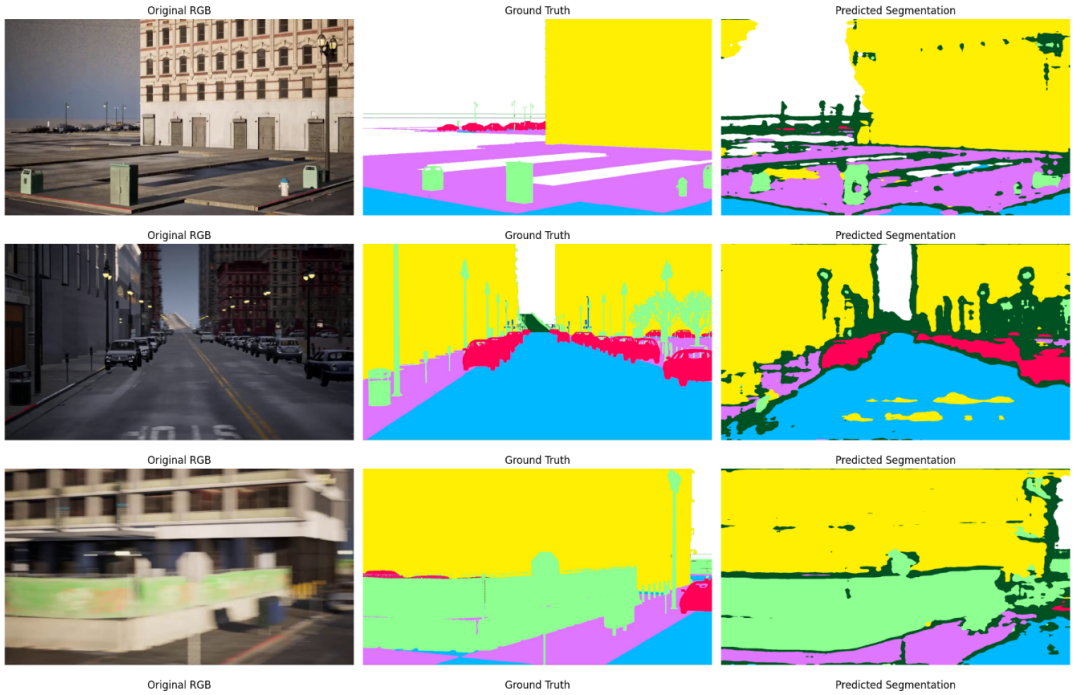


Figure 4.4: Visual comparison between ground truth and predicted segmentation masks. The first column shows the original input images, the second column displays the ground truth segmentation masks, and the third column presents the corresponding predicted masks generated by the model. The predictions closely resemble the ground truth, indicating the model's effectiveness in learning spatial features and semantic boundaries.

Based on the comparison between the predicted segmentation results and the ground truth images, the model demonstrates a moderate level of accuracy. The predictions showrange of colored regions thathat correspondto different semantic classes, such as buildings, roads, and vehicles. However, several discrepancies are evident. In particular, some predicted boundaries and class labels diverge from the ground truth, suggesting instances of over-segmentation or misclassification.

In the first set of images, a large yellow region in the ground truth is only partially reproduced in the prediction, which also introduces noise and incorrect class assignments. The second set highlights inconsistencies in delineating roads and vehicles, with the prediction including artifacts that are absent in the ground truth. Similarly, the third set shows significant mismatches in the green and purple regions, indicating difficulties in accurately identifying certain object classes.

To conclude, while the model is capable of capturing general scene structure and class distribution, the segmentation results reflect the need for further refinement. Enhancements are required to improve boundary accuracy, reduce false classifications, and achieve better alignment with ground truth data.

4.2 Discussion

The results of this study provide valuable insights into the capabilities and limitations of using synthetic environments for semantic segmentation in autonomous driving applications.

The use of Unreal Engine 5 as a simulation platform successfully enabled the creation of diverse, photorealistic scenes with consistent ground truth annotations. This approach offers clear advantages over manual labeling of real-world data, both in terms of scalability and annotation accuracy. The performance metrics indicate that the segmentation model was able to learn and generalize well within the synthetic environment, showing strong classification accuracy and spatial alignment in most classes.

However, closer analysis of the visual outputs reveals that while large, distinct classes such as roads, buildings, and skies were segmented with high precision, smaller or less frequent classes—like poles and pedestrians—were prone to misclassification or blending with neighboring categories. This suggests that class imbalance in the training data and limited visual complexity in certain object regions may affect the model's fine-grained understanding.

The consistency in the learning curves reflects stable training dynamics, yet a slight increase in validation loss towards the later epochs hints at potential overfitting. This might be mitigated with techniques like early stopping, data augmentation, or more diverse scenario generation within the simulation.

Importantly, although the model achieved high performance on the synthetic dataset, its generalization to real-world conditions remains untested. The synthetic-to-real gap is a well-known challenge in computer vision; despite the realism of simulated scenes, variations in lighting, weather, motion blur, and unpredictable object behavior in real environments may reduce model reliability when deployed outside the virtual setting.

4.3 Limitations of the study

Despite the promising results, several limitations of the current study must be acknowledged. One of the primary concerns is the synthetic-to-real domain gap. While the simulation environment using Unreal Engine 5 offers high levels of photorealism, it still cannot fully replicate the complexity of real-world imagery. Factors such as texture variations, unpredictable lighting conditions, weather

changes, and natural occlusions in real-world scenes present challenges that are difficult to model synthetically. This domain gap raises concerns about how well the trained model would perform when applied outside the simulated environment.

Another significant limitation is class imbalance in the dataset. Although the synthetic dataset includes multiple urban object categories, less frequent or smaller classes—such as poles and pedestrians—were underrepresented. This imbalance led to weaker segmentation performance for those classes, as the model prioritized learning from the more dominant categories like roads and buildings. As a result, detection and segmentation of critical, less-common objects in safety-critical applications may be unreliable.

There is also evidence of an overfitting risk, particularly visible after the 20th training epoch, where validation loss began to increase despite continued improvement in training accuracy. This suggests that the model may have begun to memorize training data rather than generalize from it. Without proper regularization or the inclusion of more diverse and varied scenarios, model performance may degrade in unseen environments.

Furthermore, boundary precision in the predicted masks poses a practical challenge. While the model accurately segmented broad object areas, it often failed to maintain sharp boundaries between adjacent objects or introduced visual artifacts. In real-time autonomous systems where accurate object delineation is critical for tasks like lane following or obstacle avoidance, such imprecision could lead to operational risks.

Lastly, a key limitation lies in model generalization. The absence of evaluation on real-world datasets means the model's robustness in actual urban traffic conditions remains unverified. Without testing the model against real images, it is difficult to determine whether the observed performance in the synthetic domain will translate effectively to practical applications.

4

Conclusions and future directions

4.3.1 Conclusion

This thesis presents a novel, modular pipeline for generating synthetic traffic data through a simulation platform, developed using Unreal Engine, to address the pervasive problem of data scarcity in autonomous driving systems. The platform simulates diverse, photorealistic traffic scenarios—including various weather conditions, lighting states, and dynamic behaviors—to generate rich, labeled datasets. These synthetic datasets are used to train semantic segmentation models capable of detecting and classifying objects within urban environments.

A key contribution of this work is the open-source nature of the platform, which is designed not only for scalable data generation but also to serve as a research and educational tool in the autonomous driving community. By eliminating the dependency on manually annotated real-world data, the platform accelerates model development while lowering the barrier to entry for researchers and educators.

However, during experimentation, the project encountered challenges, particularly with class imbalance in the synthetic dataset. For instance, traffic lights, being less frequently represented, led to misclassifications and reduced model performance in critical safety scenarios. Moreover, while the model achieved high training accuracy, generalization to real-world scenarios remains limited due to the absence of real-data benchmarks and potential domain gaps.

In summary, this thesis demonstrates that synthetic simulation platforms can significantly support the development of perception models for autonomous driving. Yet, it emphasizes the importance of improving data diversity, representation, and validation to ensure real-world robustness.

4.3.2 Further Research Recommendations

Future research should focus on several key areas to enhance the performance and applicability of the proposed simulation platform. One of the most pressing issues is class imbalance within the

synthetic dataset, particularly the underrepresentation of critical objects like traffic lights. Addressing this could involve techniques such as targeted data augmentation, synthetic oversampling of minority classes, the use of class-weighted loss functions during training, or leveraging generative models like GANs to create more balanced datasets. Improving the simulation environment itself is also crucial; incorporating more diverse and complex traffic scenarios, such as dynamic agent interactions, varying weather conditions, and different times of day, would contribute to better generalization of models to real-world conditions.

Additionally, experimenting with alternative model architectures like DeepLabV3+, U-Net, or transformer-based segmentation models, and incorporating multi-task learning could further boost detection accuracy. Evaluation methods should also be expanded beyond accuracy to include metrics like Intersection over Union (IoU), Dice coefficient, and Precision-Recall curves, which provide a more nuanced view of model performance in imbalanced settings.

Finally, integrating the trained models into real-world autonomous systems for testing and deploying them in simulation-in-the-loop frameworks would be essential for validating their practical effectiveness. Continued development of the platform as an open-source educational and research tool would also enable broader collaboration, support reproducibility, and accelerate advancements in autonomous driving technology.

A
Appendix

The dataset used in this project is synthetic traffic scenarios generated from the simulation platform. The dataset can be accessed at the following Kaggle link: `https://www.kaggle.com/datasets/pdphanh/ue-city-dataset`

All data processing scripts, analysis codes, and additional materials related to this study can be accessed on GitHub. The repository is publicly available at the following link: `https://github.com/PhuongAnh2212/capstone-project`. This repository includes:

- **Archive/**: Contains archived versions or old files of the project.
- **Controller/**: Contains Python code that interacts with the simulation in Unreal Engine 5 (UE5).
- **Data Preprocessing/**: Includes scripts for loading images and processing uneven pixel values of different classes.
- **Dataset/**: Contains the raw dataset used for training and testing.
- **Model/**: Contains the final model and several notebooks that outline and document the process of model development and testing.
- **README.md**: Provides an overview of the project, setup instructions, and other relevant information.

For questions or issues, please refer to the repository's README file or open an issue directly on GitHub.

.1 Minimum System Requirements for Houdini 19.0

To ensure smooth performance and a seamless experience with Houdini 19.0, the system must meet or exceed the following minimum requirements. These specifications cover the operating system, hardware, and software components required for optimal functionality [55].

Operating System (64-bit only):

- **Windows:** 8.1, 10, or 11 (Windows 7 and Windows Server not supported)
- **macOS:** 10.13+ on Intel or Apple Silicon (via Rosetta; native M1/M2 not fully supported)
- **Linux:** Ubuntu 18.04+, Debian 10+, CentOS 7+, Fedora 28+, RHEL 7+, OpenSUSE 15+, Mint 19.3+, Pop!_OS 20.04 LTS *Note: glibc 2.34 distros (e.g. Ubuntu 21.10+, Fedora 35+) are not supported.*

Memory:

- Minimum: 4 GB
- Recommended: 12 GB or more
- Strongly recommended for fluid simulation: 64 GB

Processor:

- Intel or AMD x64 CPU with **SSE 4.2** instruction set (required)

Disk Space:

- 4.5 GB required for installation

Input Devices:

- 3-button mouse required
- Wacom tablet recommended for brush-based tools

Graphics Card:

- Minimum: OpenGL 4.0 compliant GPU with **4GB VRAM**
- Required for high-res displays or dual monitors
- Must support **OpenCL 1.2**
- **NVIDIA Kepler or newer** required for OptiX Denoiser
- Latest **proprietary drivers** required (not open source)
  - NVIDIA: 511.09+ Studio Driver
  - AMD: 21.Q4 or higher
  - Intel: 30.0.100.9955 or higher
- *Intel GPUs are not supported on macOS.*

Monitor:

- Minimum resolution: **1920x1080**
- 4GB+ VRAM required for high-DPI or dual displays
- True color (32-bit) and sRGB/gamma 2.2 profile recommended

GPU Acceleration:

- Supports GPU-accelerated simulations (Vellum, Pyro FX, etc.) on supported OpenCL devices
- VRAM limits simulation size
- Multi-GPU support limited to Karma XPU (alpha)

Other Requirements:

- Sound card, speakers/headset
- Network access for batch jobs, offline simulation, and licensing
- Internet browser for accessing SideFX services
- Video encoder for outputting video files

.2 Minimum System Requirements for Unreal Engine 5

The advanced technologies offered by Unreal Engine 5 enabled me to develop a highly realistic scene that closely resembles real-world environments. However, achieving this level of visual fidelity comes with significant hardware demands. To ensure optimal performance, Epic Games outlines the following minimum hardware and software specifications for developing with Unreal Engine 5 [56]:

Windows Requirements

- **Operating System:** Windows 10 (64-bit), version 1909 or later
- **Processor:** Quad-core Intel or AMD, 2.5 GHz or faster
- **Memory:** 8 GB RAM (minimum)
- **Graphics:** DirectX 11 or 12 compatible GPU with the latest drivers
- **Software:** DirectX Runtime, Visual Studio 2019 or 2022 (for C++ development)

macOS Requirements

- **Operating System:** Latest macOS Ventura
- **Processor:** Quad-core Intel or AMD, 2.5 GHz or faster
- **Memory:** 8 GB RAM
- **Graphics:** Metal 1.2-compatible GPU
- **Software:** Xcode 13.4.1 or newer

Linux Requirements

- **Distribution:** Ubuntu 22.04 LTS
- **Processor:** Quad-core Intel or AMD, 2.5 GHz or faster
- **Memory:** 32 GB RAM (minimum)

- **Graphics:** NVIDIA GeForce GTX 960 or better with Vulkan support (AMD 21.11.3+ or NVIDIA 515.48+)
- **Video RAM:** 8 GB or more

Bibliography

[1] Y. Zhang *et al.*, "Perception and sensing for autonomous vehicles under adverse weather conditions: A survey," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 198, pp. 24–46, 2023. DOI: 10.1016/j.isprsjprs.2022.12.021.

[2] Bosch Global, *The impact of self-driving cars on society*, https://www.bosch.com/stories/impact-of-self-driving-cars-on-society/, Accessed: 2025-05-11, 2023.

[3] M. H. Alsharif, Y. H. Alsharif, H. H. Alsharif, and A. H. Alsharif, "Deep learning-based semantic segmentation for autonomous driving: A comprehensive review," *Heliyon*, vol. 10, no. 5, e26157, 2024. DOI: 10.1016/j.heliyon.2024.e26157. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S131915782400315X.

[4] M. Daily, S. Medasani, R. Behringer, and M. Trivedi, "Self-driving cars," *Computer*, vol. 50, no. 12, pp. 18–23, 2017. DOI: 10.1109/MC.2017.4451204.

[5] S. Cakir, M. Gauß, K. Häppeler, Y. Ounajjar, F. Heinle, and R. Marchthaler, "Semantic segmentation for autonomous driving: Model evaluation, dataset generation, perspective comparison, and real-time capability," *arXiv preprint arXiv:2207.12939*, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2207.12939.

[6] T. A. M. V. Samak and C. A. M. V. Samak, "AutoDRIVE – An Integrated Platform for Autonomous Driving Research and Education," B.Tech Project Report, SRM Institute of Science and Technology, Department of Mechatronics Engineering, Kattankulathur, Chengalpattu District, India, 2021.

[7] H. B. Sghaier, *Application of unreal engine in the automotive industry*, Bachelor's Thesis, Course: Engineering and Management, First Examiner: Prof. Dr. Max Ruppert, Second Examiner: Daniel Niederberger, Ingolstadt, Germany, 2021.

[8] Waymo, *Waymo: Self-driving technology*, https://waymo.com/, Accessed: 2023-05-03, 2023.

[9] Tesla, *Tesla: Electric cars, solar clean energy*, https://www.tesla.com/, Accessed: 2023-05-03, 2023.

[10] General Motors, *Path to autonomous driving*, https://www.gm.com/innovation/path-to-autonomous, Accessed: 2023-05-03, 2023.

[11] Apollo, *Apollo self-driving*, https://www.apollo.auto/apollo-self-driving, Accessed: 2023-05-03, 2023.

[12] Audi, *Automated driving*, https://media.audiusa.com/models/automated-driving, Accessed: 2023-05-03, 2023.

[13] BMW Group, *Automated driving - news 2024*, https://www.bmwgroup.com/en/news/general/2024/automated-driving.html, Accessed: 2023-05-03, 2024.

[14] Toyota Europe, *Automated driving: Zero accidents*, https://www.toyota-europe.com/innovation/zero-accidents/automated-driving, Accessed: 2023-05-03, 2023.

[15] MIT Racecar, *Mit racecar: Autonomous driving research*, https://racecar.mit.edu/, Accessed: 2023-05-03, 2023.

[16] Ansys, *Ansys avxcelerate: Autonomous driving simulation*, https://www.ansys.com/products/av-simulation/ansys-avxcelerate-autonomy, Accessed: 2025-05-03, 2025.

[17] IPG Automotive, *Carmaker: Simulation software for automotive development*, https://www.ipg-automotive.com/en/products-solutions/software/carmaker/, Accessed: 2025-05-03, 2025.

[18] NVIDIA, *Nvidia drive constellation: Autonomous vehicle simulation*, https://www.nvidia.com/content/dam/en-zz/Solutions/self-driving-cars/drive-constellation/nvidia-drive-constellation-datasheet-2019-oct.pdf, Accessed: 2025-05-03, 2025.

[19] Open Source Robotics Foundation, *Gazebo: A robot simulation toolkit*, http://gazebosim.org/, Accessed: 2025-05-03, 2025.

[20] CARLA, *Carla: Open-source autonomous driving simulator*, https://carla.org/, Accessed: 2025-05-03, 2025.

[21] Microsoft, *Airsim: Open-source simulator for autonomous systems*, https://github.com/microsoft/AirSim, Accessed: 2025-05-03, 2025.

[22] Sim4CV, *Sim4cv: Simulation framework for computer vision*, https://sim4cv.org/, Accessed: 2025-05-03, 2025.

[23] TORCS, *Torcs: Open racing car simulator*, https://sourceforge.net/projects/torcs/, Accessed: 2025-05-03, 2025.

[24] LGSVL Simulator, *Lgsvl simulator: Autonomous driving simulation*, https://hidetoshi-furukawa.github.io/post/lgsvl-simulator/, Accessed: 2025-05-03, 2025.

[25] BeamNG, *Beamng.tech: Realistic vehicle dynamics simulation*, https://beamng.tech/, Accessed: 2025-05-03, 2025.

[26] SideFX, *Sidefx official website*, Accessed: 2025-05-03, 2024. [Online]. Available: https://www.sidefx.com/.

[27] G. Kelly and H. McCabe, "A survey of procedural techniques for city generation," *The ITB Journal*, vol. 7, no. 2, p. 5, 2006. DOI: 10.21427/d76m9p. [Online]. Available: https://doi.org/10.21427/d76m9p.

[28] G. Poyck, "Procedural city generation with combined architectures for real-time visualization," All Theses. 3982, Master's Thesis, Clemson University, 2023. [Online]. Available: https://tigerprints.clemson.edu/all_theses/3982.

[29] B. Foundation, *Blender*, Accessed: 2025-05-03, 2025. [Online]. Available: https://www.blender.org/.

[30] Autodesk, *Autodesk maya*, Accessed: 2025-05-03, 2025. [Online]. Available: https://www.autodesk.com/eu/products/maya/overview.

[31] Maxon, *Cinema 4d*, Accessed: 2025-05-03, 2025. [Online]. Available: https://www.maxon.net/en/cinema-4d.

[32] Epic Games, *Understanding the basics of unreal engine*, https://dev.epicgames.com/documentation/en-us/unreal-engine/understanding-the-basics-of-unreal-engine, Epic Games Developer Documentation, 2023.

[33] E. Games, *Unreal editor interface overview*, Accessed: 2025-05-03, 2023. [Online]. Available: https://dev.epicgames.com/community/learning/tutorials/PnXj/unreal-engine-unreal-editor-interface-overview.

[34] E. Games, *Quick start guide for blueprints visual scripting in unreal engine*, https://dev.epicgames.com/documentation/en-us/unreal-engine/quick-start-guide-for-blueprints-visual-scripting-in-unreal-engine, Accessed: 2025-05-03, 2022.

[35] ResetEra Community, *Fortnite chapter 4 to utilise unreal engine 5.1 (nanite, lumen, virtual shadow maps, global illumination, ray tracing, tsr)*, https://www.resetera.com/threads/fortnite-chapter-4-to-utilise-unreal-engine-5-1-nanite-lumen-virtual-shadow-maps-global-illumination-ray-tracing-tsr.660787/page-9, Accessed: 2025-05-03, 2022.

[36] E. Games, *Unreal engine*, Accessed: 2025-05-03, 2025. [Online]. Available: https://www.unrealengine.com/en-US.

[37] U. Technologies, *Unity*, Accessed: 2025-05-03, 2025. [Online]. Available: https://unity.com/.

[38] G. Engine, *Godot engine*, Accessed: 2025-05-03, 2025. [Online]. Available: https://godotengine.org/.

[39] E. Games, *Get started with ue4*, Accessed: 2025-05-03, 2022. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/get-started-with-ue4?application_version=4.27.

[40] EJaw. "Unreal engine 4 and unreal engine 5: What's the difference?" Accessed: 2025-05-03. (Apr. 2023), [Online]. Available: https://ejaw.net/unreal-engine-4-and-unreal-engine-5-whats-the-difference/.

[41] M. A. Elhassan, C. Zhou, A. Khan, *et al.*, "Real-time semantic segmentation for autonomous driving: A review of cnns, transformers, and beyond," *Journal of King Saud University - Computer and Information Sciences*, 2024. DOI: 10.1016/j.jksuci.2024.102226. [Online]. Available: https://doi.org/10.1016/j.jksuci.2024.102226.

[42] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.

[43] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pp. 234–241, 2015.

[44] K. Muhammad, T. Hussain, H. Ullah, *et al.*, "Vision-based semantic segmentation in scene understanding for autonomous driving: Recent achievements, challenges, and outlooks," *IEEE Transactions on Intelligent Transportation Systems*, vol. PP, Sep. 2022. DOI: 10.1109/TITS.2022.3207665.

[45] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1706.05587.

[46] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[47] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *arXiv preprint arXiv:1412.7062*, 2014. DOI: 10.48550/arXiv.1412.7062. [Online]. Available: https://arxiv.org/abs/1412.7062.

[48] A. Chaudhari, *Witnessing the progression in semantic segmentation: Deeplab series from v1 to v3*, Medium, Jun. 2020. [Online]. Available: https://medium.com/data-science/witnessing-the-progression-in-semantic-segmentation-deeplab-series-from-v1-to-v3-4f1dd0899e6e.

[49] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *arXiv preprint arXiv:1606.00915*, 2016, Accepted by TPAMI. DOI: 10.48550/arXiv.1606.00915. [Online]. Available: https://arxiv.org/abs/1606.00915.

[50] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," *arXiv preprint arXiv:1802.02611*, 2018, ECCV 2018 camera ready. DOI: 10.48550/arXiv.1802.02611. [Online]. Available: https://arxiv.org/abs/1802.02611.

[51] R. Crane, N. Anantrasirichai, G. R. Bigg, and D. R. Bull, "Deep learning on airborne radar echograms for tracing snow accumulation layers of the greenland ice sheet," *Remote Sensing*, vol. 13, no. 14, p. 2707, 2021. DOI: 10.3390/rs13142707. [Online]. Available: https://doi.org/10.3390/rs13142707.

[52] Epic Games, *City sample quick start - generating a city and freeway using houdini*, https://dev.epicgames.com/documentation/en-us/unreal-engine/city-sample-quick-start-for-generating-a-city-and-freeway-using-houdini?application_version=5.4, Accessed: 2025-05-03, 2025.

[53] Epic Games, *City sample project - unreal engine demonstration*, https://dev.epicgames.com/documentation/en-us/unreal-engine/city-sample-project-unreal-engine-demonstration, Accessed: 2025-05-03, 2025.

[54] Y. Zhang, *EasySynth*, https://github.com/ydrive/EasySynth, Online; accessed 2025-05-03, 2023.

[55] SideFX, *Houdini 19.0 system requirements*, Accessed: 2025-05-11, 2023. [Online]. Available: https://www.sidefx.com/Support/system-requirements/19.0/.

[56] Epic Games, *Hardware and software specifications for unreal engine*, https://dev.epicgames.com/documentation/en-us/unreal-engine/hardware-and-software-specifications-for-unreal-engine, Epic Games Developer Documentation, 2023.