Universität Bremen

DLR

Master Thesis

Comparing Photorealism in Game Engines for Synthetic Maritime Computer Vision Datasets

Kashish Sharma

First Examiner: Prof. Dr.-Ing. Udo Frese

Second Examiner: Dr. Rene Weller

April 2024

Urheberrechtliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Abschlussarbeit selbstständig angefertigt habe. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Sämtliche wissentlich verwendeten Textausschnitte, Zitate oder sonstige Inhalte anderer Verfasser, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche gekennzeichnet.

Bremen, 22.04.24 Kashish Sharma

Acknowledgement

I would like to express my deepest gratitude to the German Aerospace Centre (DLR), Bremerhaven, for their unwavering support and for providing me with the opportunity to conduct research at their esteemed institution. Their resources, facilities, and expertise have been invaluable in advancing my understanding and contributing to the findings of this thesis.

I am also indebted to Borja Carrillo-Perez for his invaluable guidance, and endless patience throughout the entire journey of this thesis. His insightful feedback and constructive criticism have truly shaped this work and pushed me to strive for excellence. I hope you get to add 'Doctor' to your name soon.

To my team members at the institute, for their expertise, valuable suggestions, and encouragement. Their input has significantly enriched the quality of this research.

Above all, to my family, I owe an immense debt of gratitude. Their unwavering love, encouragement, and sacrifices have been the cornerstone of my academic journey. Their belief in me has been a guiding light, and I dedicate this thesis to them.

Lastly, to Hemken Kaffee-Rösterei, your espresso classic blend is the best!

Preface

Photorealism is the term that defines the degree of reality in images that has been generated. That is, images not taken with a traditional camera but generated artificially. These methods vary from painting an image manually (back in the 1960s when the term photorealism was coined) to generating images with Artificial Intelligence (AI). Irrespective of the method that has been employed for creating the image, they are all called photorealistic, the image looks like it has been photographed. However, there is a question that most of us do not ask when someone calls an image photorealistic. The question is, *"Photorealistic to what extent?"*

In this thesis, we will start by discussing the problems associated with the creation of computer vision datasets that comprise real world photographs. Followed by a discussion of methods that others have incorporated to overcome the shortcomings with real-world dataset creation. Then, we will apply one of the method in our use case that is maritime infrastructure. Spoiler alert! We will create synthetic datasets with game engines. Yes, game engines (with an "S"), there's two of them. The synthetic images from both the game engines will replicate the scenarios from a real-world dataset. In the end, we will place the synthetic images from both the game engines next to the real dataset image and ask - *"Photorealistic to what extent?"* The difference is this time, alongside asking, we will answer it as well.

P.S. Someday I will do something similar for the term *Cinematic*.

Abstract

Computer vision for real-world applications faces data acquisition challenges, including accessibility, high costs, difficulty in obtaining diversity in scenarios or environmental conditions. Synthetic data usage has surged as a solution to these obstacles. Leveraging game engines for synthetic dataset creation effectively enriches training datasets with increased diversity and richness. The choice of game engine, pivotal for generating photorealistic simulations, significantly influences synthetic data quality.

This thesis aims to compare Unreal Engine's and Unity Engine's capabilities in generating synthetic maritime datasets to support ship recognition applications. To this end, the real-world maritime dataset ShipSG has been replicated in the corresponding game engines to recreate the same scenarios. To this end, a true-to-scale 3D model of the Doppelschleuse was crafted using Blender, aiming to reproduce the scenery of the ShipSG images. This model is further used in the corresponding game engines, along with 3D models of different vessels, to recreate these scenarios in rendered synthetic datasets.

The performance of the generated synthetic datasets is benchmarked against the real-world ShipSG dataset using the YOLOv8 model for ship segmentation. The comparison includes an assessment of performance differences between manually annotated synthetic datasets and those with auto-generated annotations by Unity Engine. Additionally, the comparison investigates effects, such as sunlight presence and lens distortion, to find the configuration that most enhances performance when using YOLOv8. The dataset generated using Unity engine with manual annotations, both with and without lens distortion, provided the best accuracy in ship recognition (mAP of 72.3%). Both were used to augment the ShipSG dataset to train YOLOv8. The configuration with lens distortion provides the highest mAP increase, of 0.4% compared with YOLOv8 performance on ShipSG when no synthetic data is used (76.5% vs 76.9%). This evidence underscores that utilizing game engines can effectively support and enhance ship recognition tasks.

This thesis demonstrates the potential of synthetic datasets to augment real-world maritime data, highlights the importance of detailed and accurate 3D modelling, and systematically provides a methodology to select the most fitting game engine for the generation of photorealistic images to enhance maritime computer vision applications.

Contents

List of Figures

Chapter 1

Introduction

This chapter presents the core motivations and goals of the thesis. We will introduce the potential of synthetic data generation in the maritime domain using game engines, set the stage for a comparative study of photorealistic synthetic images, and motivate their application to enhance real-world maritime datasets for computer vision tasks.

1.1 Motivation

The field of computer vision has witnessed significant advancements, driven by the availability of large scale datasets for training machine learning models [1]. These datasets typically consist of images captured in real-world scenarios, showcasing a variety of events and occurrences [1]. Ideally, the events depicted in these images should reflect those occurrences accurately. However, achieving this ideal scenario is not always possible. Several factors hinder the creation of such datasets:

- Data acquisition cost: obtaining real-world data can be expensive. The logistical cost associated with the equipment and infrastructure add to the overall cost of acquisition.
- Data bias: real-world data can experience data bias because of environmental conditions, sampling methods. This limits the information and can contribute to unfair results.
- Privacy and ethical considerations: datasets from the real-world can contain information that is sensitive and raise concerns for privacy and ethical usage. Complying to privacy and ethical regulations is crucial for dataset creation and usage.

In the maritime domain, these challenges are amplified by several factors. First, the environmental conditions in maritime environment are unpredictable and harsh with factors like fog and rain can affect the visibility and image quality [2]. Second, the limited viewing points from which images can be captured, restrict the perspectives from which data can be collected [3]. Further, the vastness of the maritime environment adds to the logistical cost and effort spent on data acquisition. The quality of the data that needs to be acquired is the next critical aspect to consider.

Synthetic data generation has emerged as a promising alternative that addresses the limitations of real dataset [4]. Synthetic datasets offers the ability of generating datasets on demand, including scalability, reproducibility and customisation [5]. By leveraging advanced simulation techniques like a game engine, highly detailed synthetic datasets that simulate maritime scenarios can be generated [3][4][6]. Exploration of multiple viewpoints, dynamic sequences provides for a comprehensive data for instance segmentation. Furthermore, it facilitates cost-effective data generation and eliminates the need for extensive resources.

1.2 Goals of the Thesis

The primary goal of this thesis is to conduct an empirical comparison of synthetic images generated by Unity and Unreal game engines, focusing on their quality and photorealistic attributes, and the performance when used to augment real-world maritime datasets. The breakdown of the subgoals of this research are:

- **Photorealism Simulation:** Explore how the Unity and Unreal game engines simulate photorealism in synthetic maritime datasets. This includes the recreation of a detailed 3D model of a port infrastructure and ships present. Moreover, with an analysis of rendering methodologies that contribute to image realism, such as textures, materials, lighting, and dynamic environmental effects such as sky and water.
- **Systematic Comparative Analysis of Game Engines:** Benchmark synthetic datasets rendered with the game engines against real-world maritime datasets to assess quantitatively their relative quality and usefulness. The comparison will focus on the accuracy in ship segmentation using these datasets.
- **Study of Photorealism Effects:** Identify and analyze the critical parameters that influence the photorealism of synthetic images. This goal aims to derive insights on optimizing game engine settings for enhanced realism in maritime synthetic dataset generation.
- **Augmentation of Real-World Datasets:** Investigate the potential of using synthetic datasets to augment real-world maritime datasets in improving the performance of computer vision models, specifically focusing on ship segmentation tasks.

Through these objectives, the thesis will provide a comprehensive evaluation of the capacities of both game engines to produce high-quality, photorealistic images for computer vision applications, thereby informing future uses and developments in synthetic data generation.

1.3 Structure of the document

The theoretical foundation of perspective, along with 3D modelling software Blender, Game Engines and a game engine component Skybox is discussed in Chapter 2. Chapter 3 contains discussion on prior works that have carried out in real-world ship segmentation, use of synthetic datasets and game engines of the generation of synthetic datasets. The creation of a true-to-scale model of the Doppelschleuse along with its materials and vessel models is covered in Chapter 4. Chapter 5 elaborates the methodology that has been followed for the creation of synthetic dataset in both the game engines. The experimental setup for the analysis along with the results in presented in Chapter 6. In Chapter 7 the outcome of the thesis is summarised and concluded. Finally, in Chapter 8, future work is suggested.

Chapter 2

Theoretical Background

Prior to the description of the methodologies used in this thesis, there are some theoretical foundation which discussion eases the understanding of them. In this chapter, we will start by describing a photography phenomena, the perspective, vanishing point and lens distortion. This is followed by a description of Blender a tool that supports 3D modelling. Subsequently, we will briefly discuss game engines and we will establish a baseline for game engine comparison. Lastly, an important common component of game engines, the skybox, is introduced.

2.1 Perspective, Vanishing Point and Lens

When taking a photograph of the real-world, we are trying to represent a 3D world on a 2D surface (photograph). The appearance of the objects in the 3D world and its representation on a 2D surface depend on factors like the position of the photographing camera, the focal length of the lens that is being used and the relative position and orientation of the objects to the camera. By changing any of these factor, can result in a different reproduction of the 3D world on the 2D surface. To match the real-world images to the images in the game engine, it is important that these factors coincide. In this section we will be discussing certain aspects of perspective drawing that will help us match the real-world images to the images that are being formed in the game engines.

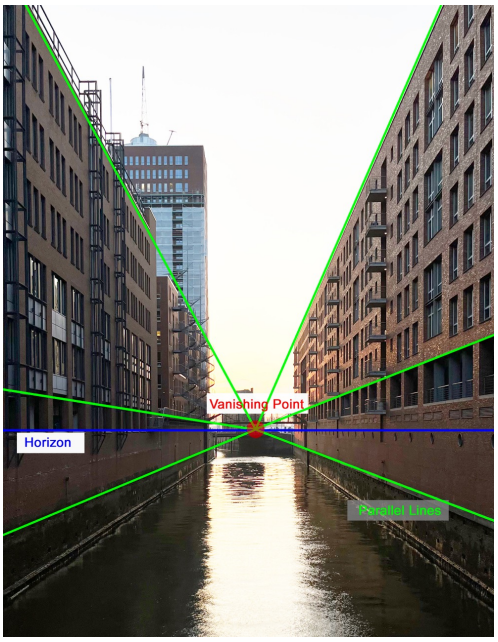


Figure 2.1: Vanishing Point on the Horizon

The rules of perspective are techniques applied in painting to give a flat surface a sense of depth, much like taking a photograph. The major rule of perspective is that parallel lines in the 3D world when (as seen in the fig.2.1) projected, all cross at the same point in the 2D space. This point is called the Vanishing Point (in red, fig.2.1). The vanishing point lies on the Horizon, which is imaginary horizontal line that lies on the eye level of the viewer. Tracing the vanishing point on real world image (fig. 2.1) helps in understanding the orientation of the camera with which it has been photographed.

Another phenomenon that images exhibit is **lens distortion**. Lens distortion is the deviation from the ideal projection that is, the image appears to be deformed or curved unnaturally. Consider this, if the straight lines traced in the fig. 2.1 for the vanishing point wouldn't have appeared straight, then there would have existed a lens distortion in the image. There are five major types of lens distortions, Barrel, Pincushion, Mustache, Chromatic Aberration and Vignetting [7].

In the scope of this thesis, Barrel distortion is important. Ideally the horizontal and vertical lines (fig. 2.2) in an image should be appear straight (also seen in fig. 2.1). Barrel distortion cause vertical and horizontal lines to bend outwards from the center of the image and occurs with wide field of view lenses [8]. Another reason for a Barrel distortion to occur is if the size of the image plane (diagonally) is larger than the focal length of the lens [9].

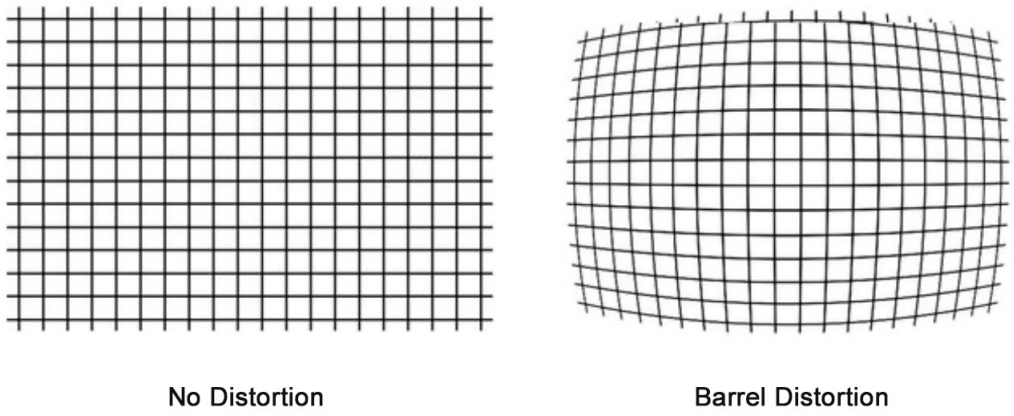


Figure 2.2: No Distortion (left) and Barrel Distortion (right). Image Source [10]

.

In the Chapter Environment setup in game engines, we will trace the vanishing point and observe the lens distortion on an image from a real dataset. Doing so will provide us with the orientation of the real camera, and help us in the mimicking the same in synthetic images.

2.2 Blender

To craft a 3D model of that should be similar to real world, we need a 3D modelling software that can create complex models in file formats compatible with the generation of synthetic images later on. Blender [11] serves as tool for that.

Fundamental Tools

Boasting a comprehensive suit of tools for modelling, animation, rendering and compositing, Blender has emerged as a powerful tool for 3D modelling and content creation. Some of the fundamental tools include

- Measure tool [12] - helps with measuring distance or angles and areas within the 3D scene. This tool is helpful in ensuring that the size of the models correlate their real-world counterpart.

- Extrude tool [13] helps in creating new geometry by extending existing faces, edges, or vertices along their normals or specified directions.
- Knife tool [14] facilitates cutting and creating new geometry within a mesh object.

Besides these tools Blender also provide, Camera [15], useful for defining the viewpoint from which the scene will be rendered, and Light [16], that helps in illuminating objects or scenes.

Blender offers the ability to assign materials to the models. The materials can be baked onto the model or assigned with the help of an external image. Baking materials [17] optimise the scenes for faster rendering or export them to external applications while preserving the visual fidelity of the materials. Shader Editor [18] provides support for assigning material with the help of external image files [19]. By default a Shader Editor initiates with a Principled BSDF (Bidirectional Scattering Distribution Function) [20] node that helps in adjusting material properties like, Base Colour, Roughness, Alpha (transparency) etc. There is also provision for generating materials procedurally [21] and materials that are transparent like glass.

These comprehensive features make Blender the ideal choice for creating realistic models while maintaining the level of detail and accuracy required for complex structure.

An in-depth use case scenario of the Blender tools and methods of creating materials is explained in Chapter 4 Model Creation. Creation of every model and materials are explained individually.

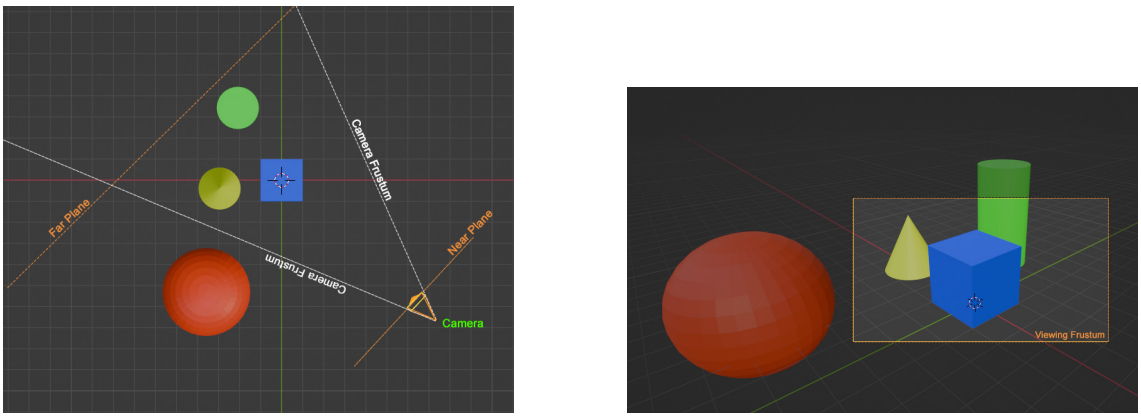
2.3 Game Engine

Game Engines are complex, multipurpose tools designed for the development of games. Though Game Engines are not 3D modelling software, they offer a development environment which can be reused for variety of games, sometimes even without the knowledge of scripting [22] [23].

Modern game engines are built on component architecture [24], where each component handles a particular task. Major components are: Audio, which handles the sound design and music of the game; Physics Engine, which helps in simulating physics; Scripting, which offers the ability to add custom codes in the game; Animation, which enables the capability of animating; and the most important component for the creation of synthetic image datasets, the Rendering Engine, which handles the graphics rendering.

2.3.1 Rendering Engine

When considering 3D games, one often first thinks of the stunning graphics. The rendering engine is where the action happens. The camera, lighting, and objects— with their materials and textures—are placed to prepare the scene for rendering. The rendering engine then compiles this scene setup and executes the rendering process.



(a) Camera Frustum with Near and Far Planes

(b) Viewing Frustum

Figure 2.3: Culling in Render Engine

In a scene (fig. 2.3a), there are objects placed at certain positions along with the camera and the light source. The camera has configuration much like the real-world camera. Based on the camera's configuration (fig. 2.3a), near and far clipping planes are assigned by the render engine, where the objects in the scene are visible. Game engine makes use of this information and calculates the matrix to transform the objects from the scene's world coordinates to the viewport of the camera. Occlusion culling [25] takes place and the objects that are outside the viewing frustum are cut off. Objects that lie inside the viewing frustum are rendered (fig. 2.3b).

In this thesis, the rendering engine holds the utmost significance in the game engine, as we will use the rendered image for experimentation.

2.3.2 Contemporary Game Engines

Companies such as Ubisoft [26], Rockstar games [27] have their own proprietary engines. Relatively smaller studios or independent developers make use of ugh third party game engines. As of December 02, 2022 (fig.2.4) the two most popular game engines are the Unity Engine and the Unreal Engine [21]. These two game engines together hold more than 50% of the market. Thus, these two game engines are chose as the contenders for comparison in this thesis.

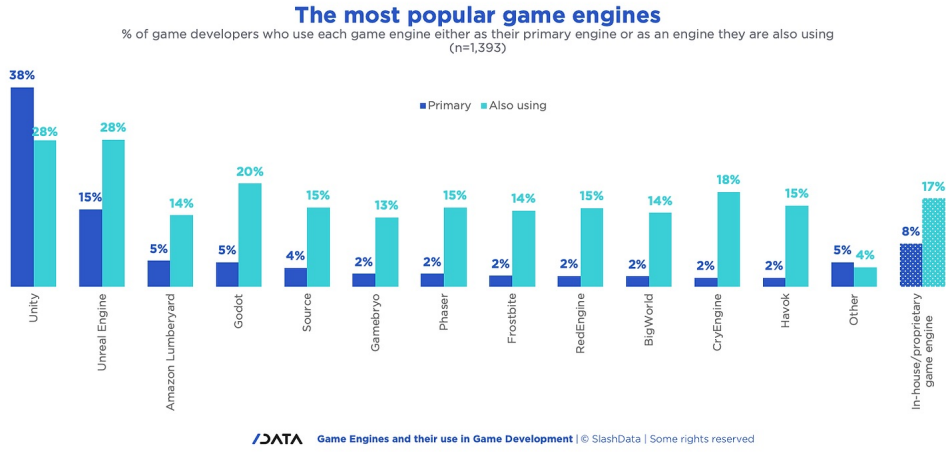


Figure 2.4: Popular Game Engines, Source: Slashdata (December 2, 2022)

2.3.3 Unity Engine

Announced at the June 2005 Apple Worldwide Developers Conference, Unity Engine [28] holds the highest number of users. Credited as the easy to use for beginners [29], Unity offers support for iOS, Android, macOS, Windows as well for XR (Extended Reality). Unity Engine offers two development pipelines; the Universal Render Pipeline (URP) [30] and High Definition Render Pipeline (HDRP)[31]. The URP is optimised for performance and supports wider range of platforms, including mobile devices. The HDRP is designed for high-end performances and prioritises visual quality over performance. Unity engine is free for students and hobbyists [32].

2.3.4 Unreal Engine

Cited as the most successful game engine [33], Unreal Engine [34] was primarily used for first person shooter games. The first game that was built using the engine is Unreal [34] in the year 1998 [35]. The game engine Unreal was developed by Epic Games and is available for free for students, educators, hobbyists and companies generating less than $1 million USD. The pricing for companies generating over $1 million USD in annual gross revenue and are not creating games is $1,850 per seat [36].

The source code is written in C++ and Unreal provides its own Blueprint Visual Scripting [22] for those who are not familiar with the C++ language. The current stable version in Unreal Engine 5.3.

2.3.5 Baseline for game engines comparison

So far we have seen the features of both the game engines. To make sure that both the game engines have the same level for a fair photorealism comparison in rendering images for maritime computer vision applications, it is important to set some baseline for consideration before we dive into the experiment.

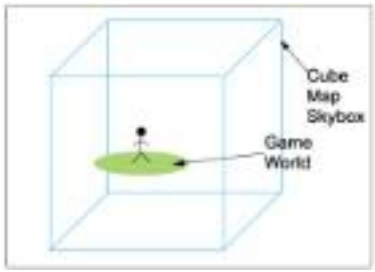
- The version of both engines should have similar rendering capabilities.
- Considering the rendering capabilities of both the engines, the most recent stable version of both the game engines should be considered.

Given this baseline, Unity Engine HDRP 2023.2 and Unreal Engine 5.3 are selected for comparison in this thesis.

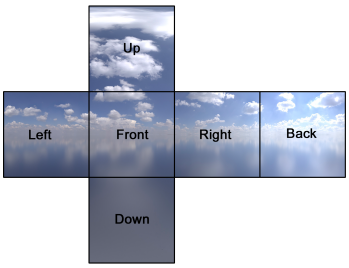
2.4 Skybox

The Skybox is a common component in game engines that helps with the creation of a visually immersive background, setting the scene mood. Skyboxes can be cube maps or panoramic images projected onto the interiors surface of the game world. This technique is used to simulate the appearance of a distant scenery or the sky itself.

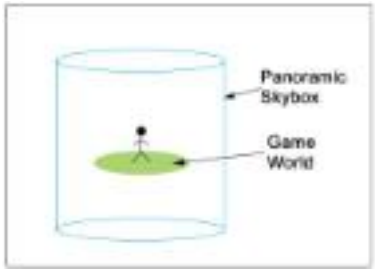
Both the game engines are capable of creating procedural sky [37][38]. But, the generated procedural sky in both the game engines will differ. In the scope of this thesis, it is important that the sky in both the game engines match each other. Therefore, the procedural skybox is replaced by a static skybox. The static skybox in both the game engines will carry the same material for the sky. Thus, the sky appears consistent in the respective game engines.



(a) Skybox - Cube Map



(b) Unfolded Cube Map Skybox



(c) Skybox - Panoramic

(d) Panoramic - Unfolded
Source [39]

Figure 2.5: Skyboxes - Cube Map and Panoramic

A **cube map skybox**, as the name implies, forms a cube when folded. A common shape that the skybox takes when unfolded is shown in fig. 2.5b It comprises six images that a player

sees when looking in that direction (front, left, right, up, down and back) (fig. 2.5b) The game world is positioned inside this cube map skybox (fig. 2.5a) A **panoramic skybox** (fig. 2.5d) is a single panoramic image that encircles the game world in a latitude-longitude (cylindrical) fashion (fig. 2.5c).

A panoramic skybox material can be used with a cube map skybox. When applying a panoramic skybox material (fig. 2.5d) to the cube map skybox, the game engine optimises the panoramic skybox material and matches it to the shape of cube map (fig. 2.5b). This is true for skyboxes whose shape differs from the two discussed as well (see Section 5.3.1).

In this project, four panoramic skybox materials (fig. 2.5c, 2.5d, A.2, A.4) are used with both the game engines. The four skyboxes are sourced from Poly Haven [40]. The chosen skybox materials facilitates the creation of skies with overcast, clear and cloudy conditions. The implementation of the skybox for both the game engines is elaborated in Sections 5.2.1 and 5.3.1. The first of the four sky boxes can be seen in fig.2.5d, the rest can be seen in the Appendix (fig. A.2, A.3, A.4).

Chapter 3

Prior Work

In Chapter 2 Theoretical Background, we have discussed important concepts that will be utilised in this thesis. Moreover, we discussed rendering in game engines and the reason behind selecting Unity and Unreal as game engines and laid the baseline of the comparison. In this chapter, we will observe existing works in the literature, which will help understand the problems associated with the creation of maritime real-world dataset. Followed by a discussion on a real-world maritime dataset that will be utilised for the creation of synthetic images and as a benchmark for the comparison. Then we will have an overview of the existing work on synthetic dataset creation and why game engines are suitable for it. In the end, some existing comparison of the two game engines will be discussed.

3.1 Real-world datasets for maritime computer vision

Instance segmentation methods, normally reliant on deep learning [1], aim to identify object instance in an image at the pixel level, delineating its shape with a mask. These methods require datasets to be trained [1]. In the scope of the maritime domain, general purpose instance segmentation datasets like COCO [41] do not fit the task of ship recognition. Maritime domain requires images from the real-world that showcases varied ship data with precise annotations including the ship class, masks and features like geographical coordinates. Existing datasets related to ship detection that are Singapore Maritime Dataset (SMD) [2], SeaShips [42] and the dataset by Chen et al. [43]. These datasets suffer from a lack of annotations for instance segmentation, limited diversity in ship types, and the absence of a favorable oblique view, all of which complicate the process of georeferencing. To over come these obstacles with ship segementation, a novel dataset ShipSG [44] was created by DLR Bremerhaven.

ShipSG is a real-world dataset for ship segmentation and georeferencing available in the public domain. The images in the dataset showcase the Fischereihafen-Doppelschleuse, port basin in Bremerhaven, Germany. The dataset (fig.3.1) comprises 3505 images using two different cameras with static and oblique view. Overall, 11625 ship masks annotated manually are grouped in seven classes. The seven classes are Cargo, Law Enforcement, Passenger/Pleasure, Special 1, Special 2, Tanker and Tug. Each image includes a geographic annotation specifying the latitude and longitude of one of the ship masks depicted in the image. With the ShipSG dataset images, we experience a Barrel distortion because the focal length of the lens is smaller than the diagonal size of the camera sensor and the lens has a wide field of view (FOV) (see section 2.1).

Figure 3.1: Images from ShipSG dataset, showcasing the two camera views and annotated masks and class

Acquiring these images of real-world datasets such as ShipSG is a difficult task in itself (as discussed in Section1.1 Motivation). In the publication [2], Dilip K. Prasad et al have explained the issues like changing illumination and weather conditions, variation in the appearance of the objects. These challenges with data acquisition led to exploration of alternate means for creating datasets, such as synthetic datasets to support real-world applications, which will be covered by Section 3.3 In the next section we will be discussing a real-world maritime dataset and its performance with segmentation model. This dataset will serve as a benchmark in our quest for choosing the best game engine for synthetic maritime applications.

3.2 Instance Segmentation on ShipSG using YOLOv8

As discussed in the previous section, deep learning algorithms depend on deep learning algorithms that require task-specific data. In the publication [45], YOLOv8 [46] was utilised for ship segmentation and georeferencing using the ShipSG dataset (see section 3.1). YOLOv8 is a state-of-the-art real-time model computer vision model built upon previous YOLO (You Only Look Once) versions. YOLOv8 architecture provides support for object detection, instance segmentation, pose estimation, tracking, and classification. YOLOv8 also offers customisation of the architecture along with five model sizes. The models range from the quickest and lightest to the deepest and most precise: YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l, and YOLOv8x.

The mean Average Precision (mAP) is a widely used metric in computer vision that evaluates the precision of a model's object detection and instance segmentation (mask of the object) capabilities. It is computed by averaging the precision across all classes, where precision is defined as the ratio of true positive detections over the sum of true positive and false positive detections. True positives are determined by whether the Intersection over Union (IoU) between the predicted and ground truth bounding boxes or instance masks exceeds a certain threshold. The formula for IoU, in the case of instance segmentation, is given by:

$$IoU = \frac{area(M_{pred} \cap M_{gt})}{area(M_{pred} \cup M_{gt})} \tag{3.1}$$

Where M_{pred} and M_{gt} represent the predicted and ground truth masks, respectively.

The Average Precision for each class i is then computed by summing the precisions at different IoU thresholds:

$$\text{AP}_i = \sum_{t \in T} p(t) \Delta t \tag{3.2}$$

In this formula, T is the set of IoU thresholds (normally from 0.5 to 0.95), $p(t)$ is the precision at threshold t, and Δt is the difference between consecutive IoU thresholds (normally 0.05).

Finally, the mAP is the mean of the AP values for all $N_{classes}$ classes:

$$mAP = \frac{1}{N_{classes}} \sum_{i=1}^{N_{classes}} \text{AP}_i \tag{3.3}$$

Instance segmentation using YOLOv8x performed on the ShipSG for the segmentation and classification of ships yields a mean Average Precision (mAP) of 76.45%. This value is used as benchmark for this thesis when evaluating the performance of game engines using synthetic datasets. The performance of the synthetic dataset should be as close as possible to the benchmark of the real-world dataset. To achieve the highest performance possible, the developments in the 3D scenes of this thesis focus on the perception of the synthetic datasets as photorealistic as possible. This will be described in detail in the following chapters.

In the following sections, we will discuss synthetic datasets as alternative to real datasets, which will allow us narrow down the methodology followed in this thesis to obtain our own synthetic datasets, that should be suitable suitable for maritime applications.

3.3 Existing Synthetic Datasets

Synthetic dataset generation has shown the potential to the issues of the real-world dataset acquisition (Section 3.1) by creating artificial images that simulate real-world scenarios, allowing to address the limitations in real data availability and quality. By generating synthetic images, researchers and practitioners have augmented existing datasets with diverse and realistic data, improving the robustness and capabilities of computer vision models[3]. It was in 2010 that Simon Baker et al. [47] proposed the first realistic and a pure synthetic computer generated dataset, the Middlebury Dataset.

Besides eliminating the obstacles of real-world data acquisition, synthetic dataset generation has showcased other advantages ([48],[49],[50],[51],[52]), including:

- Scalability: synthetic data generation techniques has provided with tools for creating large scale datasets repressing diverse scenarios
- Bias control: as the generation of dataset is within control, the extent of bias in the dataset can be controlled. Moreover, by augmenting synthetic data in real these bias can be minimised.
- Cost-effective: generating synthetic dataset can be cost effective and help in reducing the time it takes for annotations.
- Preserves privacy: The images are generated artificially, the question of sensitive information getting leaked is eliminated altogether.

Jonathan Becktor [3] et al. have regarded the use of synthetic dataset as a solution to their data collection problems in maritime setting. Inclusion of synthetic data resulted in an increase of 3-5% overall performance in their use case (discussed in depth in section 3.4). Additionally it has provided flexibility with dataset creation and optimise the entire creation process.

MPI-Sintel [49] dataset was the first dataset to make use of Blender [11], a 3D modelling-animation software for the generation of synthetic images. The open source short film Sintel [53] was utilised to study optical flow. The MPI-Sintel [49] dataset provided with longer scene sequence, along with specular reflections and atmospheric effects that are missing in the Middlebury dataset. These effects made the generated images more realistic. While the MPI-Sintel dataset showcased the use of animation software in generating synthetic dataset, there are limitations associated with the use of Blender. For instance, Blender does not offer real-time interactivity, the physics in Blender is again not real-time but baked. These limitations open the door for other simulation methods, like Game Engines. In the following section, we will discuss at existing works where game engines proved their capabilities in generating synthetic datasets.

3.4 Game Engines for Synthetic Data Generation

Using a 3D animation software for the generation of synthetic dataset paved a way for simulation software. Simulation softwares where the real world can be recreated photo-realistically, rules of the physics applies and can be tweaked to match the dataset needs, like playing a 3D game. Reason why, Stephan Richter et al. [54] made use of the game GTA5 (Grand Theft Auto 5) to create large

datasets with pixel-level labels. Thus reducing the human effort and accelerating the development of computer vision. DeepGTAV [4] is an open source framework to produce ground truth training data. DeepGTAV has been utilised for object detection on Unmanned Aerial Vehicles (UAV) [4].

With game engines, a developer or practitioner is not bounded to the assets that are available within a game. The CARLA [55] project is a open source simulator for autonomous driving in urban layout built with Unreal Engine. CARLA is also used for the Virtual Traffic Simulator by the students at the Universität Bremen to replicate Borgfeld district in Bremen to improve traffic handling [56].

Microsoft made use of game engine to build simulators for drones and cars in their project AirSim [57].The AirSim project is available with both the Unreal and Unity game engine. Unreal Optical Flow Demo showcases a patch that facilitates the implementation of the features on an Unreal-based robot simulator, Pavilion [58]. The project also combines the use of Unreal Game engine and the Robot Operating System (ROS) [59].

RobotriX [50] is a dataset by Alberto Garcia-Garcia et al that helps in solving robotic vision problems. The dataset comprises of hyperrealistic indoor scenes explored by robot agents. To record and generate the dataset, they built a tool called UnrealROX in Unreal Engine 4. An improvised version of UnrealROX, UnrealROX+ [51] was presented in 2021. UnrealROX+ added new features like Python API for interacting in virtual environment and generating albedo.

In the maritime domain, Jonathan Becktor [3] et al (also in section 3.1 Real-world datasets for maritime computer vision) made use of Unreal Engine and created an environment similar to Limfjorden in Denmark. Various ship models that are seen in Limfjorden were added in Unreal Engine for the creation of synthetic datasets. This way, the complications with data collection that they faced in their earlier project [48] was resolved. As a result the overall performance with object detection improved by 3-5%.

As seen in all these projects, the use of game engines for the generation of synthetic dataset has shown its potential. However, so far these works do not discuss the reason for which they selected or developed their game engine for their computer vision application. Therefore, the next section will review existing literature where game engines have been compared so that a scientific selection of the correct game engine can be done.

3.5 Comparison of the Game Engines

Regarding the comparison of the two game engines under consideration for this thesis, Unity Engine vs Unreal Engine, there is no extensive research on the topic.

In their publication [60], Ramiz Salama et al have only discussed the history, supported platforms and marketing of both the game engines. Hussain Ali Juma Al Lawati [61] has presented a similar comparison of the two game engines and stated that both the game engines are similar.

Paul E. Dickson et al [62] provided a subjective comparison where students were first taught game development with both the game engines and were asked to pick which one of the two is easier to learn. The students credited Unreal engine as frustrating to learn. From faculty's point of view, they cite Unity engine was easier but teaching Unreal engine provided more job satisfaction.

Antonín Šmíd [24] presented a comparison of the two game engines by building the same Pac Man game. The comparison showcases the hardware profiling of both the game engines on various platforms such as a PC, laptop, mobile device and VR glasses. The visual comparison is subjective and compared with Blender's visual fidelity. Alongside the author has also provided personal opinion of the comparison.

Therefore, a scientific and quantitative comparison of both engines when generating synthetic datasets, analysed for their specific task, has not been found in the literature. In this thesis, we tackle this comparison in the specific task of ship segmentation from synthetically generated images to find a reasoning behind the selection of the game engine.

Chapter 4

Model Creation

In Chapters 2 and 3 we explored the foundations of photorealism, the use of game engines for synthetic image generation, and existing applications that could be improved using them, namely maritime computer vision applications like ship segmentation. Expanding upon the motivated need to use game engines for such an application, this chapter focuses on the description of the 3D models crafted and developed in this thesis. These models will take part in the complete 3D scene that will be used later in the game engines to render synthetic images for ship segmentation, and will allow the comparison of the engines.

We start by identifying structures from the real dataset, ShipSG, that are crucial for the reconstruction. Following that, the modelling and the steps taken to ensure their scale matches the real world are explained. Lastly, the section 4.3 highlights the vessel models that will be used in the scene.

4.1 Breakdown of Real Image - Key Structures

As discussed in Section 3.1, the ShipSG dataset was captured with views to the Doppelschleuse in Bremerhaven. From, Section 2.2 we know that game engines are not 3D modelling software. Therefore, to create a true-to-scale 3D model replica of the Doppelschleuse, we make use of Blender. Later, the handcrafted 3D model of the Doppelschleuse will be utilised in both the games engines.

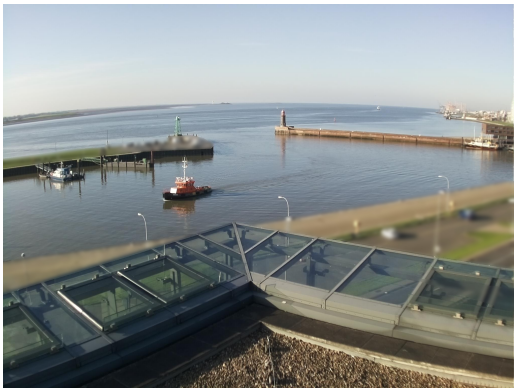
To ensure that the 3D model of the Doppelschleuse is true to scale, Google Maps [63], along with its measure tool, was employed to verify the model's adherence to real-world proportions. Google Maps is a web mapping service developed by Google that provides detailed satellite imagery, street maps, 360-degree panoramic views of streets (Street View), real-time traffic conditions, and route planning for traveling by foot, car, bicycle, or public transportation. The Doppelschleuse location was visited and visually inspected on multiple occasions and documented for any details that have been missed out on.

We begin by identifying the key structures that are visible in the ShipSG dataset. This helps us narrow down the structures that are of prime importance and should exist in the 3D model.

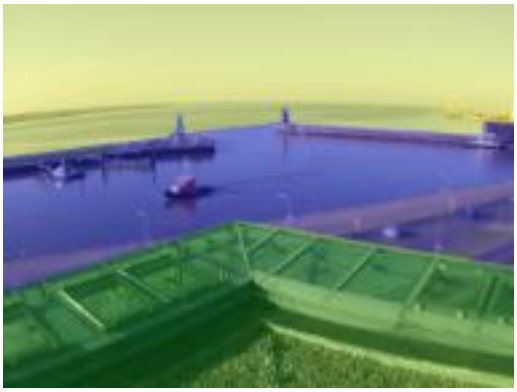
Identifying the key structures

The images in the ShipSG dataset (fig. 4.1a) have been photographed from the rooftop of the Alfred Wegener Institute (AWI) at the Doppelschleuse, Bremerhaven, Germany. The AWI building has a glass exoskeleton visible in the foreground (fig. 4.1b). The section of the rooftop where the two sides of the exoskeleton meet faces the Doppelschleuse port basin, as can be seen in fig. 4.1a. The mid-ground (fig. 4.1b) comprises the street connecting the Doppelschleuse with the city and the port basin with the two lighthouses. In the background (yellow, fig. 4.1b) is the Weser river dividing Nordenham to the left and Bremerhaven to the right.

Most relevant structures for the 3D scene recreation are in the foreground and the mid-ground. The background comprises of the water body and distant land which are not that significant. Thus, the Bremerhaven land (orange in fig. 4.2) in the background has been omitted from the 3D model to simplify the modelling. The key structures (fig. 4.2) that are pivotal for the the 3D-model and



(a) Real Image from ShipSG dataset



(b) Breakdown: Foreground - Green, Midground - Blue, Background - Yellow

Figure 4.1: Breakdown of the depth in image

should exhibit a high level of detail (LOD) are the AWI building (green), the connecting street (blue), the two lighthouses viz., Molenfeuer Süd (red) and Nordmole (yellow) and the Lotsenstation (magenta).

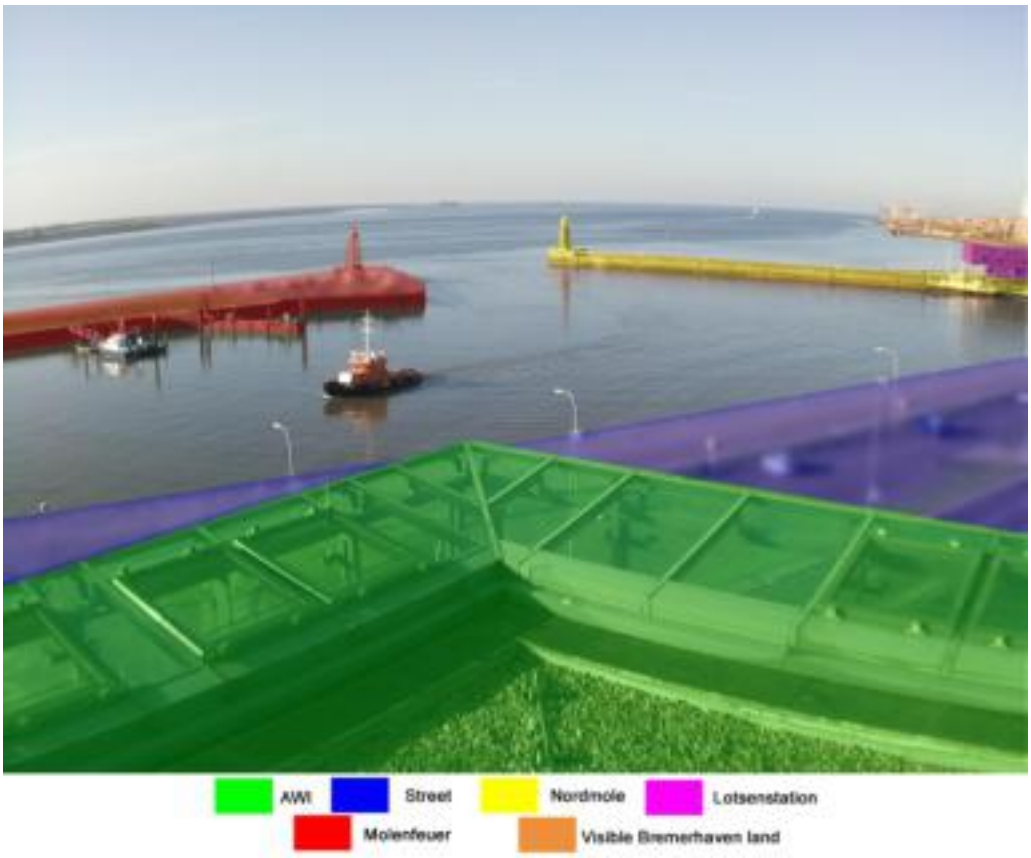


Figure 4.2: Key Structures for 3D model creation

The Nordmole lighthouse (yellow in fig. 4.2) existed at the time of creation of the ShipSG dataset (2020). Unfortunately, in August 2022, its pier sank and subsequently it was demolished to avoid any mishaps. Thus, it does not appear in the satellite view of the Doppelschleuse (fig. 4.3). However, since the Nordmole is part of the ShipSG dataset, its 3D recreation has been included in this thesis and is explained in section 4.2.

4.2 Creation of the models

The following Steps (1 to 5) explain the development flow of the 3D scene undertaken in this thesis.

Step 1: The Land

Before modelling the relevant structures using Blender, it is important to model and scale the land on which they stand. This helps in identifying their locations and orientation. A satellite top view image of the Doppelschleuse (fig. 4.3) was taken from Google Maps [63]. The area under consideration has a width of 1000m and a length of 500m. This image serves as a reference for the shape of the land and its dimensions and shapes are used in Blender.

Figure 4.3: Satellite View of Doppelschleuse from Google Maps

The reference image is scaled to measure (1000x500m) in Blender to match the real world scale. The dimensions of the land features in the reference image is compared with the measurements in Google Maps (fig. 4.4). A contour of the land under consideration is traced and a plane mesh is introduced in Blender. The plane mesh is modelled to match the traced contour. This creates the ground map of the Doppelschleuse. Because a plane mesh was utilised for the tracing, the map traced is a planar mesh, therefore it's scale on Z-axis is 1. To give this planar mesh a volume, it is extruded (see Blender tools in section 2.2) vertically (Z-axis) to 10m to provide ample depth, so that after adding water in the respective game engines(see Sections 5.2.1, 5.3.1), the water does not flood the land surface.

(a) On reference image

(b) Crafted in Blender

Figure 4.4: Measurement of the features and Marking the key structures

The terrain of the Doppelschleuse is treated as a flat land, except for the Weser-Strandbad, located next to the Lotsenstation. If the Strandbad is treated as a flat land, then it comes in the line of sight with more prominence. This is not the case in the real images as seen in fig. 4.4a, where the Strandbad presents a slight slope. Hence, the Strandbad tapers towards the end and appears different in the Blender top-view (fig. 4.4b).

After superimposing the reference image on the created ground in Blender, the position of the key structures, along with the respective length, width, and orientation, is marked (fig. 4.4). The tracing of the street is also aided by the satellite view. The top view does not provide the height of the structures and their shapes. In the following steps to this one, we derive the height for each structure and consider its shape.

The shape of the traced street (fig. 4.4b) was subdivided (cut up) from the main land model with the Blender knife tool[14] (see 2.2). This subdivision helps in changing the material of the street from the land materials of the land.

Materials for the Land

Based on the appearance of the land, four major materials can be inferred. The first one is for the grass, second for the gravel parts, a third one for the Strandbad and lastly as asphalt for the street and Molenfeuer Süd land. The materials ([64],[65],[66],[67]) for these parts were obtained from Poly Haven [40] and Ambient CG[68]. To apply the material at their respective positions, four new materials were created in Shader Editor (see section 2.2) of Blender. Each new material has a Principled BSDF Node [20] (see section 2.2). The obtained materials have their own, Base Color, Roughness and Normal which connects to the Principled BSDF Node of the created materials. By combining all the properties of the materials, a composite material is obtained, which is applied to their respective position (fig.4.19).

Step 2: Molenfeuer Süd & Lotsenstation

The Molenfeuer Süd (fig. 4.5) measures 14m in height[69] and is supported by a base that stands 1m tall(fig. 4.5a). It comprises 6 sections(fig. 4.5b), with the bottom 5 sections serving as pilings with a crosshatch pattern. The light source is housed in the service room on the top. The service room makes up one-fourth of the entire structure, the remaining three-fourth are its pilings. Based on these observations, the model for the lighthouse was crafted. The ladder has been exempted from the model.

(a) Molenfeuer Süd (b) Molenfeuer Süd Proportions

Figure 4.5: Molenfeuer Süd and its structures proportions

Lotsenstation is a four-story structure. As seen in fig. 4.6, the base floor is elevated compared to the three floors above it(fig. A.1b). The width of the floor above the base is slightly small(fig. A.1a). The top two floors are wider than the bottom floors by 25% and the extension is supported by exposed pillars(fig. A.1a). The base floor has an entrance for parking along with a warning for the permissible maximum height of the vehicle to be 2.1m (fig. 4.7). Using this the height of the structure can be estimated by stacking rectangles of the same height as the entrance to the parking. The seventh rectangle exceeds the height by half. Thus, the overall height of the Lotsenstation is

approximated to 13.65m (height per floor = 3.4m)((fig. 4.7)). The approximate length and breadth of the Lotsenstation is known from Step 1.

Figure 4.6: Lotsenstation

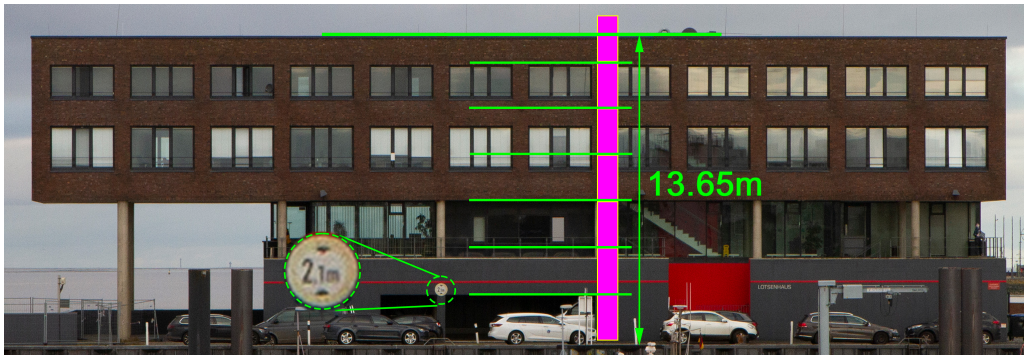


Figure 4.7: Tracing the height of the Lotsenstation

Materials for the Molenfeuer and the Lotsenstation

The Molenfeuer Süd has three materials for the main structure, the base and the glass on the at the light source. Material for the main structure and the base is Baked whereas for the glass a new material with Glass BSDF[70] is created and assigned to it.

The Lotsenstation comprises five materials each for the base floor, the pillars, the top two floors, the glass windows and the glass floor. To create glass materials, Glass BSDF[70] is used. Rest of the materials are baked in the structure.

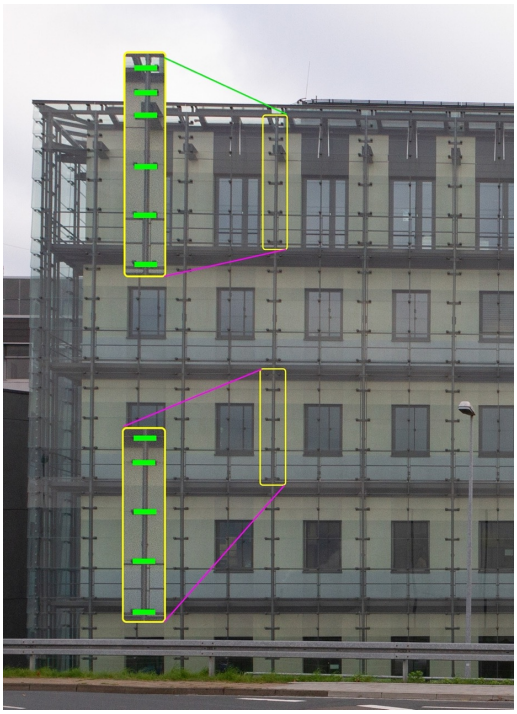
Step 3: The AWI Building

(a) AWI - front view

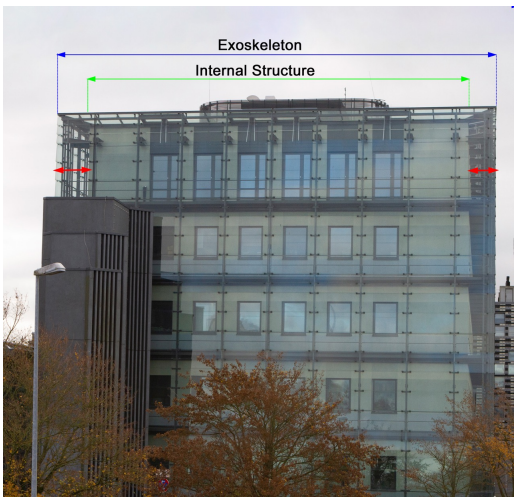
(b) AWI - side view

Figure 4.8: Alfred Wegener Institute (AWI), Bremerhaven

From Step 1, the position and orientation of the AWI building is known. The Lotsenstation measures approximately 13.65m in height and comprises four storeys and serves as a reference. The inspection of the structure in person, revealed the height of each floor to be over 3.4m. Additionally, the top floor of the AWI is taller than the rest. The number of clamps (fig. 4.9a) securing the glass exoskeleton showcases this difference in height. Therefore, using the Lotsenstation as reference and the visual inspections in person, the height of the five-story AWI can be approximated to 20m.



(a) Number of clamps (in green) on the top floor and rest of the floors



(b) Difference in size of the exoskeleton and internal structure

Figure 4.9: Estimation of the dimensions - AWI

The exoskeleton and interior structure of the AWI building are at different heights. The inward slope on the rooftop (fig. 4.1a) showcases this difference in height. Plus, the interior structure is approximately 1m shorter (fig. 4.9b) from all sides of the exoskeleton. The longer side of the AWI has 29 vertical bars in its exoskeleton. The bars are organised in a pattern that alternates between thick and thin. The shorter side is composed of 9 bars, all with equal thickness. The awning windows on the rooftop (seen in fig. A.7a) are positioned between 2 consecutive thick bars. These observations of the AWI provide a clear understanding of the model's shape and are crucial because the dataset images were captured from its rooftop and the structure lies in the foreground.

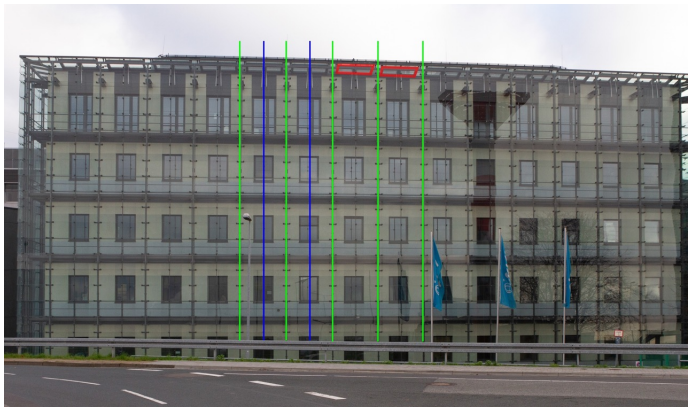


Figure 4.10: Arrangement of the bars (thick - blue, thin - green) and the position of the awning windows(in red)

Materials for the AWI Building

The AWI has four materials altogether for the glass exoskeleton, the frame of the exoskeleton, floor at the rooftop and for the step that surrounds the floor. The glass exoskeleton material is created like the glass material of the Lotsenstation 4.2. For the exoskeleton frame, the material is baked. Same material that was used for the gravel in the Land (section4.2), is used on the rooftop of the AWI.

Step 4: Nordmole

(a) Nordmole

(b) Nordmole height distribution

Figure 4.11: Proportions of the Lotsenstation. Source: www.deutsche-leuchtfeuer.de

The reference image (fig. 4.3) from the Google Maps do not provide a precise position of the Nordmole or of the land on which it stood. For the reconstruction of the model, it is important that the position of the Nordmole matches with the dataset images. To ensure this, a camera [15] is added in Blender. The resolution of the camera matches the dataset image resolution. This camera is positioned on top of the AWI and overlooks the Doppelschleuse basin. The obtained view from the camera matches the view of the dataset image. This facilitates in matching the position of the Nordmole to the dataset image.

The height of the Nordmole is known to be 14m [69]. By dividing the entire structure into 14 equal parts (fig. 4.11b), the position of each feature is estimated. In the division, the lighting rod is in the 1st and 2nd sections. The dome covers the 3rd and 4th sections while the astragal glass for the light belongs to the 5th section. In the 6th section lies the balcony, while the three sets of windows are positioned in sections 8, 10, and 12-13 respectively. The entrance goes from section 12 to 14.

(a) Concentric platform - green, Half hexadecagon - pink

(b) Cubes at the bottom - pink, Width of the boundary - green

(c) Position of the pathway

Figure 4.12: Aerial and side view of the Nordmole
Image Source: (fig. 4.12a, 4.12b) www.nordsee-zeitung.de, (fig. 4.12c) www.butenunbinnen.de

The pathway leading from the mainland to the Nordmole has a boundary on both the sides. The length of the pathway is adjusted by comparing the image from the Blender camera with the real image. One of the boundary is thicker than the other (fig. 4.12b) and the pathway does not meet the lighthouse at its centre (fig. 4.12c), but is positioned to one side[1]. The edge of the land on which the Nordmole stood was not a perfect circle but had 8 sides or half of a hexadecagon (16 sided polygon)(fig. 4.12a). The Nordmole's centre and the hexadecagon coincided. The Nordmole stood on a platform with three concentric, elongated circles of varying sizes(fig. 4.12a). This should have also served as steps to access the door of the Nordmole. The base has alternating

[1]Some of the images in this section were photographed while dismantling the Nordmole and are used to explain the structure's shape and form. The purpose of these images is not to cause any offense in any manner.

brick cubes(fig. 4.12b) surrounding the entire circumference of the Nordmole. Finer details such are the year "1914" on the lightning rod, the bent ladder on the dome is added(fig. 4.18).

Materials for the Nordmole

The Nordmole has four materials, a bricked material that extends to its pathway, another material for its dome, a material for the astragal glass and a material for the base. The bricked texture of the Nordmole and its pathway is created by making use of procedural brick texture node in Blender [71]. The dome's colour is taken from the Nordmole image, and is baked in it. The glass material is created like the glass material of the Lostsenstaion 4.2. The colour of the base is baked as well.

Step 5: Other Details

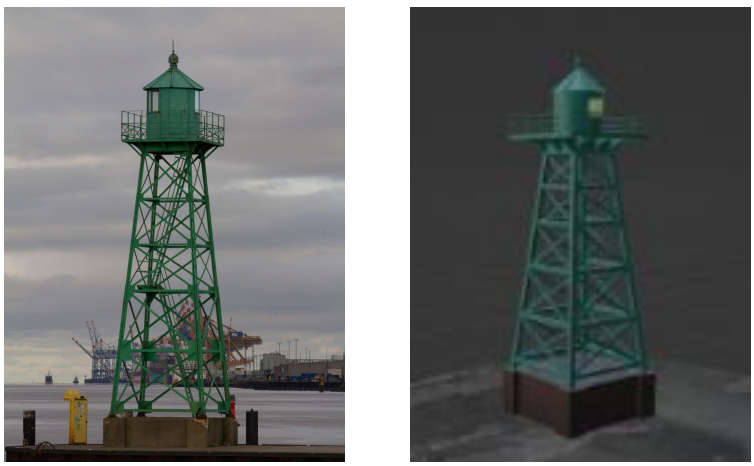
The piers visible in the dataset images (fig. 4.1a) are added on land on which the Molenfeuer Süd stands. Their orientation is traced from the satellite image (fig. 4.3) as well. The ribs on the vertical surface of the Molenfeuer Süd land is created. The streetlights are added as well. The respective **materials** are baked (see 2.2)in each structure.

4.2.1 Results of the Complete 3D Model

The integration of the key structures presented in Steps 1 to 5 complete the Doppelschleuse model with the rest of structures. The completed model is exported in .fbx (filmbox) format[72] with +Y axis Up (right-hand coordinate system)[73]. This format is supported by both the game engines, allowing a seamless import. The materials created with Principled BSDF are exported as .png image files. The baked materials are retained in the fbx model. The glass materials are only supported within Blender. Therefore the fbx model does not retains the glass material. The material for the glass is created within the respective game engines and explained in their respective sections (section 5.2.1, 5.3.1). This model is exported in both the game engines to maintain consistency with the game objects and allows a fair comparison. The following subsections will present the resulting crafted 3D models described in the previous section, and that have been crafted in this thesis.

Molenfeuer Süd

Fig. 4.13 shows the visual comparison between the real and crafted Molenfeuer Süd. The colour of the baked (see 2.2) materials were extracted from the real image. The damage to the corner of the base is excluded as it is not that significant.



(a) Molenfeuer Süd - Real

(b) Crafted Molenfeuer Süd - Blender

Figure 4.13: Molenfeuer Süd

Lotsenstation

In Fig. 4.14 the visual comparison between the real and crafted Lotsenstation is shown. It is important to note that the windows in the Blender model are considered as a single piece because their division is not visible in the real dataset images. This reduces the complexity of the model as redundant meshes are removed.

(a) Lotsenstation - Real

(b) Crafted Lotsenstation - Blender

Figure 4.14: Comparative Visualization of Lotsenstation

Alfred Wegener Institute (AWI) Building

(a) AWI Building - Real

(b) Crafted AWI Building - Blender

Figure 4.15: Alfred Wegener Institute (AWI)

The visual comparison between the real and crafted AWI building is shown Fig. 4.15. It is noticeable that as the images of the real dataset ShipSG are captured from the rooftop of the AWI, the windows are not visible. Thus, they are omitted in the crafted 3D model.

Nordmole

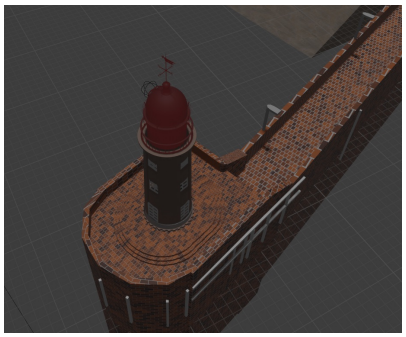
The comparison of the real and 3D crafted Nordmole can be seen in fig. 4.16 and 4.17. We can observe that the support structures (in white) are added as final details.

(a) Nordmole - Real

(b) Crafted Nordmole - Blender

Figure 4.16: Nordmole - Side View

(a) Nordmole - Real



(b) Crafted Nordmole - Blender

Figure 4.17: Nordmole - Top View

As can be seen in fig. 4.18, the dome is shaded uniformly in the model as the faded part is not significantly visible in the dataset image. This also reduces the time and labour that would have been spent on shading the dome manually.

Doppelschleuse

The top-view of the completed model (fig. 4.19) showcases all the structures with their materials. The parts that do not have a colour on them are not important for the project, since they do not take part in the views of ShipSG. Thus, they have the default white material.

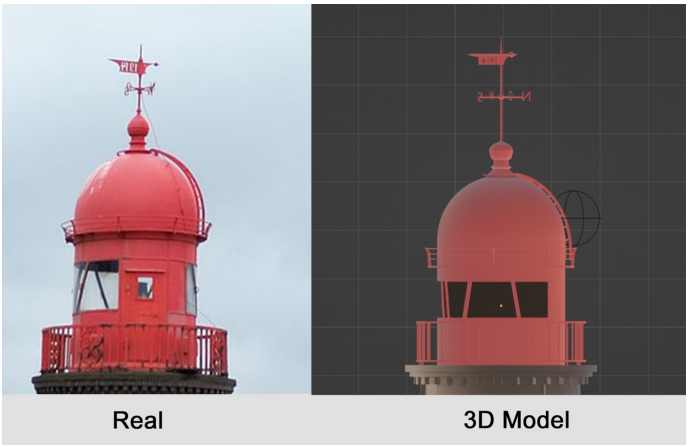


Figure 4.18: Nordmole dome (left - real, right - 3D model)

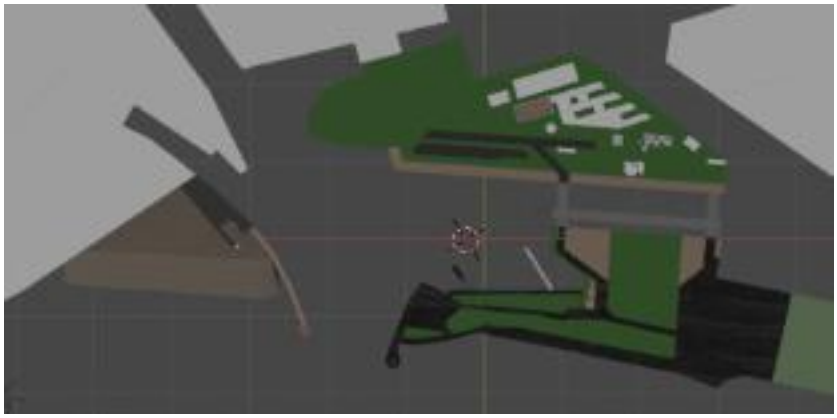
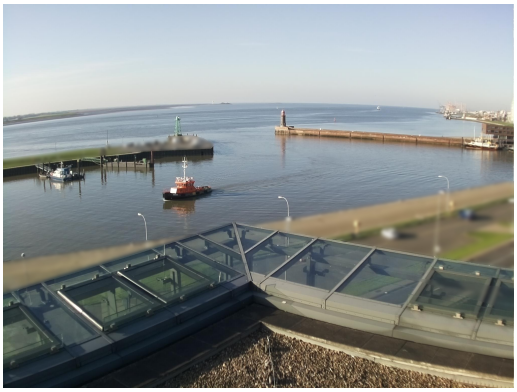
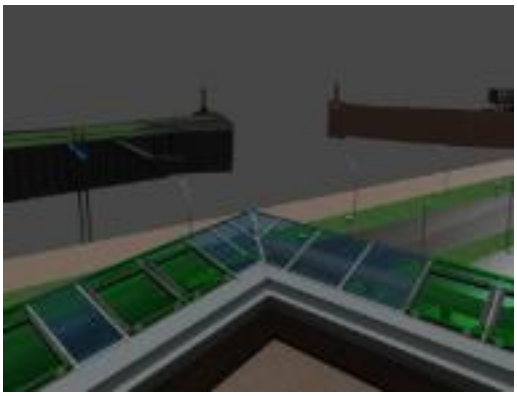


Figure 4.19: Top view of the crafted Doppelschleuse using Blender.

4.2.2 View from the AWI in Blender



(a) ShipSG - Real



(b) Crafted ShipSG ShipSG view - Blender

Figure 4.20: Visual comparison of the real and crafted Doppelschleuse, including all key structures.

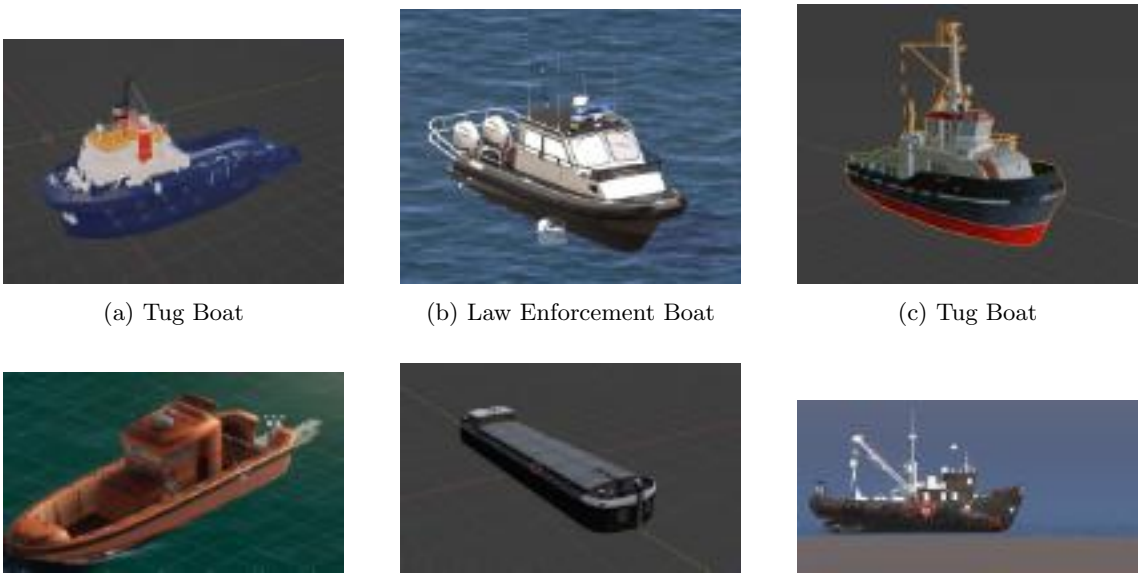
A view similar to the real view provided by the ShipSG dataset can be observed in fig. 4.20. To the left the image from the real dataset ShipSG and to the right is the image rendered from the Blender camera 4.2 showcasing the created model. The depth of the land underneath the surfaces in the model is 10m (see 4.2) which provides sufficient depth to add the water body within the

game engines.

This completes handcrafting the modelling of the true-to-scale Doppelschleuse 3D model. As discussed, consideration has been given to the Level of detail (LOD) that the model exhibits is high. Special attention has been given to the materials as well.

4.3 Vessel Models

As presented in section 3.1, the vessels in the ShipSG dataset are categorised into 7 classes. Twenty-one vessel models that replicate those present in the ShipSG and span over the 7 classes were acquired from Turbosquid[74] to be used within the crafted 3D model to simulate real ShipSG scenarios using the game engines. The resizing of the vessels was performed in Blender to match the scale of the vessels present in ShipSG. The real dimensions was obtained from the AIS[44] messages of the ShipSG dataset, which contain ship length information. The materials of certain vessels (fig. A.7) were changed in Blender as well, to match the vessel colours present in the dataset. All the vessels acquired and resized can be found in the Appendix. (fig. A.5, A.6, A.7).



(a) Tug Boat (b) Law Enforcement Boat (c) Tug Boat

(d) Law Enforcement Boat (e) Cargo Vessel (f) Fishing Boat

Figure 4.21: Some of the Vessels

Chapter 5

Environment setup in game engines

In the previous Chapter 4, we have discussed the creation of a true-to-scale 3D model of Doppelschleuse using Blender and the vessel models. In this chapter we elaborate on the methodology followed in both the game engines for the creation of synthetic datasets, which precedes the experimental evaluation presented in Chapter 6. We begin by tracing the vanishing point (see the theory in section 2.1) on a sample image from the ShipSG dataset. With the groundwork established, we dive into the methodology followed in the respective game engines and discuss the components like sun, water that will help in creating synthetic images that match the real dataset.

5.1 Breakdown of Real Image - Perspective

Figure 5.1: Vanishing Point in the ShipSG image vanishing point in red, horizon in blue, reference lines in green

Unlike the example discussed in the Section 2.1, tracing the Vanishing Point sometimes is a challenge because of lack of parallel lines in the image. The ShipSG dataset was created with a Raspberry Pi camera module that measures of 7.9mm diagonally [75]. This sensor was equipped with a 6mm wide-angle lens [76]. As a result, the images of the dataset exhibit a Barrel lens distortion (section 2.1). This distortion is evident based on the the appearance of the horizon in

fig. 5.1 . Moreover, it can be inferred from the figure 5.1 that the camera is not held parallel to the ground but tilted downwards.

The simplest vanishing point (red) (fig.5.1) that can be traced is made possible by drawing a line from the rooftop of the Lotsenstation, and that can be observed in green. Another line from the platform that exists on top of the Molenfeuer Süd and another one on land on which it stands. All three of them, appear to emerge from the same point on the horizon (blue).

The tracing of the vanishing point is critical, as it helps to align the camera in respective game engines to match the real images. The parameters (like focal length) of the real camera can be assigned to the camera in the respective game engine. But the orientation of the game engine camera should match the orientation of the real camera. Only then, the synthetic images will match the real one. Now that we know the position of the vanishing point in the real image, we can orient the camera in the respective game engine to match the vanishing point of the synthetic image with the real one. Thus, creating a synthetic image that is equivalent to the real image.

With the vanishing point of the real image now known, we are ready to setup the game engines for the creation of synthetic dataset, starting with the Unity Engine.

5.2 Unity Engine

This section elaborates on the steps to create the synthetic scene with the 3D Doppelschleuse model. Before diving into the methodology, it is important to become familiar with the terminologies [77] used by Unity. Understanding these terms helps clarify the operations involved and described in this chapter. These terminologies are listed as follows:

1. Scene: Scene contains the environment of the game like camera, characters, etc.
2. GameObject: Fundamental object in a scene that represents characters, scenery, camera, etc.
3. Component: The functional part of a GameObject. For instance, if a GameObject carries a component “Rigid Body” it will then obey rigid body physics.
4. Prefab: A GameObject that has been saved as a template. It facilitates in easy usage if the same GameObject carrying the same components is to be used multiple time.

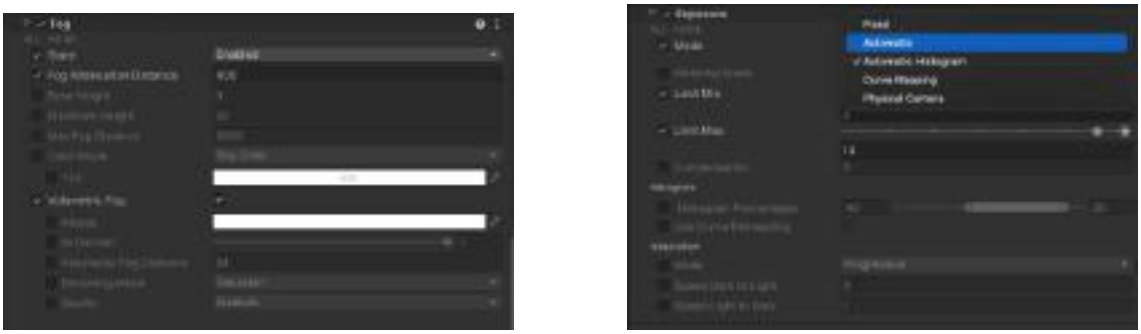
With the important terms now known, we start by creating a Master Scene. The Master Scene serves as a template for all the Scenes that are to be recreated.

5.2.1 Creating the Master Scene

Upon initialising a new HDRP (see section 2.3.3) Scene in a project, Unity Engine provides default Sky and Fog Volume, Sun and a Main Camera in the Scene. This Scene serves as a Master Scene. GameObjects that are common to all the Scenes that are to be recreated are added to the Master Scene. Then, copies of this Master Scene is created. Modifications are made in each Scene based on the respective reference image. This fosters in easy development of multiple Scenes with less error.

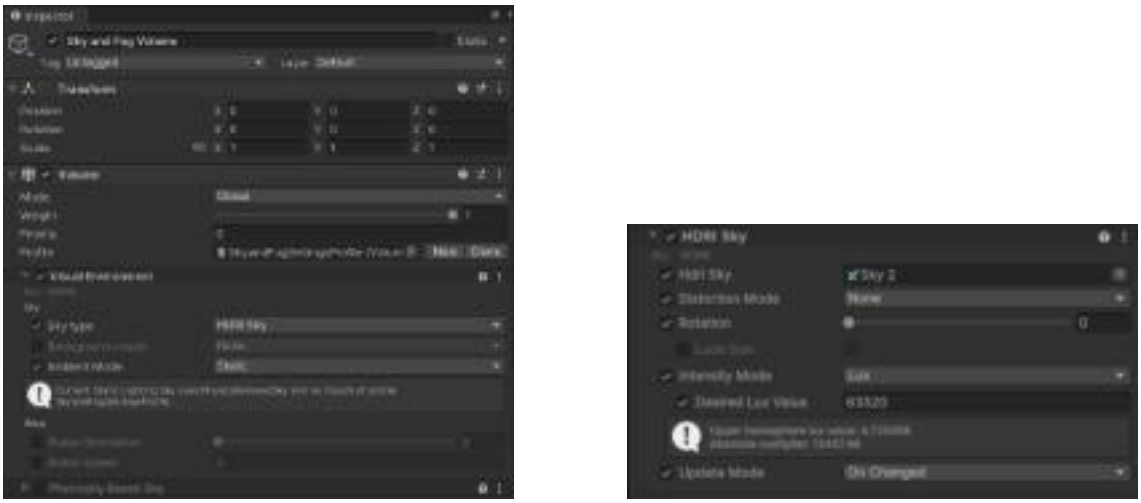
Sky & Fog Volume

The Sky and Fog Volume (fig. 5.2a) helps in creating volumetric effects of the Sky and Fog.



(a) Fog Component (b) Exposure Component

Figure 5.3: Fog and Exposure Components



(a) Sky & Fog Volume Component (b) HDRI Sky Component

Figure 5.2: Sky & Fog Volume Component

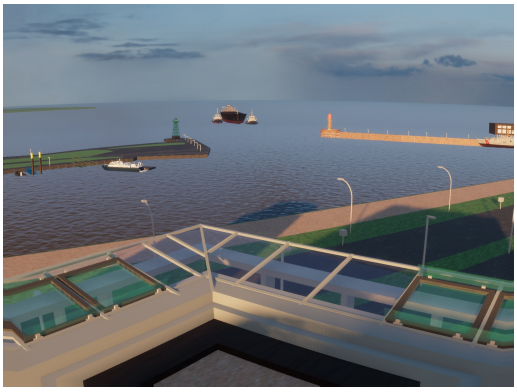
The Volume mode is Global so that the changes are rendered globally and not bounded to a finite volume. Under the Visual Environment component, the Sky type is changed form Physically Based Sky to HDRI (High Dynamic Range Image) Sky. This enables the use of skybox (see section 2.4). Along with this change, the Physical Based Sky component is turned off and HDRI Sky override (fig. 5.2b) is added to the Sky and Fog Volume. Custom cubemap skybox material is assigned under the HDRI Sky. This component provides with the rotation of the skybox as well. Thus, the same skybox material can be used to create different appearance of the sky as required. The Intensity Mode is changed to Lux, this enables setting of light intensity for the diffused skylight.

The Fog component (fig. 5.3a) facilitates the activation of fog in a scene. The component provides Fog Attenuation Distance, which controls the density of the fog and visibility. The default value of the Fog Attenuation Distance is 400. By reducing the default value, heavy fog is created. Fog Attenuation Distance is changed as per the Scene's requirement.

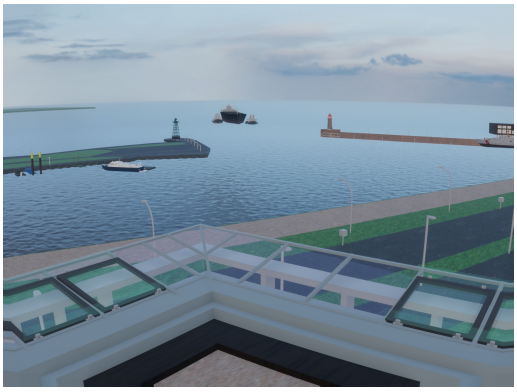
Exposure component (fig. 5.3b) facilitates in setting the rendering exposure of the Scene for the camera. The menu provides with options like Fixed, Automatic and Physical Camera to control the exposure [78]. In this use case, as the camera properties are custom, the Exposure is set to Physical Camera, thereby making the exposure dependent on the camera settings and parameters.

Sun

The Sun GameObject facilities in simulating a directional sunlight (fig. 5.4a). In real world, the objects cast shadows based on the position of the sun. In Unity engine, casting of the shadow is governed by rotating the Sun.



(a) With a directional Sun light



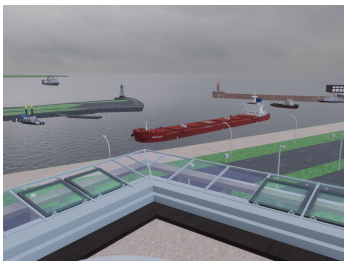
(b) Without a directional sunlight

Figure 5.4: Effect of a directional sunlight

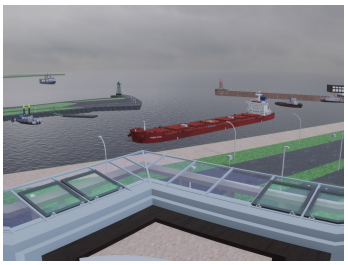
Under the Emission component, the appearance of the light can be tweaked. The two modes, Filter & Temperature and Color provide control over the light's appearance. In Filter & Temperature, the colour temperature of the light can be calibrated from 1500°K to 20000°K along with its intensity in Lux. The intensity unit Lux is similar to the real-world [79]. Therefore, by increasing the Lux value, a sunny day can be simulated (fig. 5.4a). In the absence of a directional sunlight (fig. 5.4b), the HDRI Sky component becomes the only source of light. Thus, a diffused ambient light is spread throughout the Scene. This is crucial for the investigation in parameters for photorealism.

Camera

The Camera (Main Camera) components help with the setup of the rendering parameters. The Physical Camera is enabled to change the camera parameter. The sensor size of the camera used for capturing ShipSG images measures 7.9mm diagonally and the focal length of the lens is 6mm. By enabling the Physical Camera, these parameters are applied to the Camera.



(a) No lens distortion



(b) with lens distortion and colours

(c) Monochrome Image

Figure 5.5: Effects of Post-Processing volume on the images

The curvilinear wide angle distortion exhibited in the real images (fig. 5.1) is introduced in synthetic images (fig. 5.5b) by adding post processing volume [80] in the Scene. Post processing volume provides with Lens Distortion component that is tweaked to create the curvilinear horizon (fig. 5.5a). Under the post processing volume, we also get the option to change the colour saturation of the image. By reducing the values to 0, the images can be made monochromatic (fig. 5.5c) .

These two parameters, lens distortion and monochromatic image are also part of the photorealism experiment.

Water

The Water system package was introduced by Unity at the Tech Stream 2022.2 [81]. It provides an out-of-the-box tool to create water bodies [82].

Figure 5.6: Ocean Component

A new GameObject Ocean is added from the Water GameObject list. Under the Project Setting HDRP, Water is enabled. Upon doing so a water body will spawn. Scripts are provided by the developers to simulate buoyancy [83]. Script interaction needs to be enabled for this. The waves in the water body are procedurally generated with Fast Fourier Transform (FFT) [81]. These waves are controlled by changing the wind parameter under the swell tab (fig. 5.6).

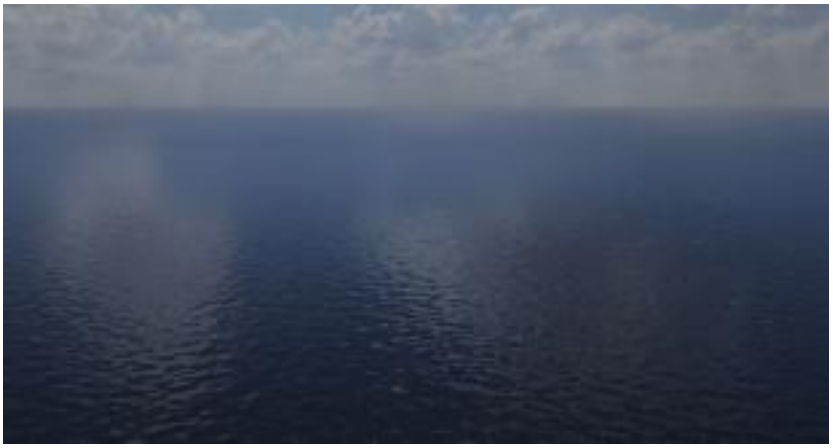


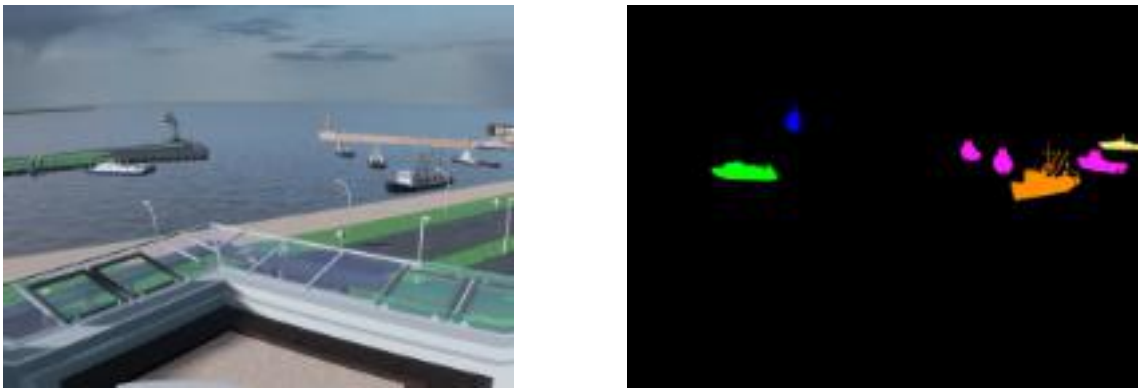
Figure 5.7: Simulated Ocean - Unity Engine

The appearance of the sky correlates with the appearance of the water's surface (fig. 5.7) i.e. no additional changes are required to have the reflection of the sky on water.

Perception Package

The Perception package [84] offers a set of tools for creating synthetic datasets intended for computer vision purposes. The toolkit is equipped with a Perception camera component that captures RGB images with the ground truth. By adding this component to the Camera GameObject,

images can be captured. The frame rate at which the images are captured is controlled by this component. The Labelling component assigns the labels to the GameObject. Thus, every time the labelled GameObject appears in the Scene, it will be annotated according to its label. Label Config component carries the taxonomy of all the labels. Under the Label Config, the annotation type like Bounding Box, Instance Segmentation, Semantic Segmentation are assigned along with the label colour.



(a) Generated Image (b) Generated Annotation Mask

Figure 5.8: Generated Image and its generated annotation masks. Colours represent different object categories.

The Perception package captures images as per the defined frame rate, simultaneously it also generates annotation masks (fig. 5.8b) and a JSON file for the masks. As the package is still experimental, the generated JSON file provides support for only Bounding Boxes in Unity Engine's own format Synthetic Optimized Labeled Objects (SOLO) [85]. In this project, the vessels are annotated under Semantic Segmentation. Thus, the generated JSON file is redundant, as it carries no information about the segmentation masks. To make use of the generated segmentation masks with the YOLOv8, the annotation masks are converted from an image file to YOLOv8 annotation format. The YOLOv8 format consists of one text file per image. Each row in the text file corresponds to a class and contains the polygon coordinates of an instance of the object in the image [86].

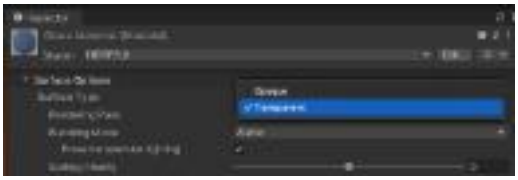
Setup of the Doppelschleuse model

The Doppelschleuse model created in the section 4.1 is added in the Scene along with the material files. The baked materials are created by the Engine upon import. The materials are manually assigned to their respective locations.

As discussed in section 4.2.1, the material for glass needs to created in the Unity Engine. To do so, a new material is created in the Scene. The Surface type of the material is changed from Opaque to Transparent (fig. 5.9a). Under the Surface Inputs the Alpha (A) of the Base Map (fig. 5.9c) is reduced to a 5. Setting the Alpha to 5, makes the glass transparent but retain 5% of opaqueness. Lastly, the Refraction Model (fig. 5.9b) under Transparency Input is changed to Planar with the Index of Refraction (IOR) set to 1.52. This value of the IOR for glass is provided by the developers [87]. The glass material is then applied to the exoskeleton of the AWI building. Similar glass materials are created and assigned to the glass of the lighthouses and the Lotsenstation.

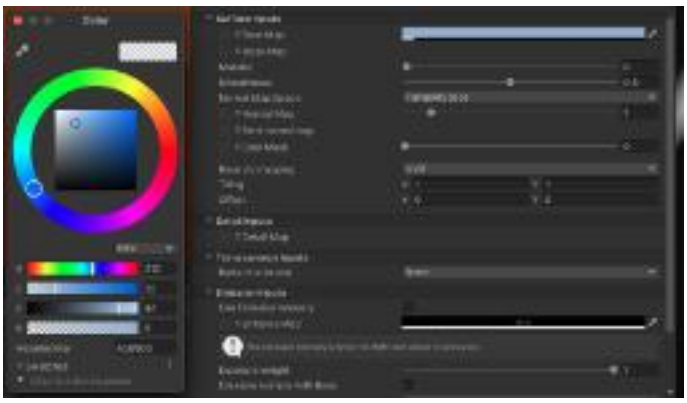
All the vessel models are added into the Scene. Their materials and scale are inspected and the respective segmentation labels are assigned. These vessel models are then converted as Prefabs. This reduces the hassle of adding a vessel in the Scene and ensuring that it exhibits the properties consistently.

The addition of the Doppelschleuse model completes the setup of the Master Scene. In the Section (6.2.1), we will discuss the creation of the datasets with the help of this Master Scene, to be used in the experimental evaluation of this thesis. In the next section 5.3 we will setup Unreal Engine for synthetic dataset creation.



(a) Surface Type - Transparent

(b) Setting the IOR



(c) Alpha (A) Setup

Figure 5.9: Properties of the Glass Material

5.3 Unreal Engine (UE5)

In this section, we will be setting up the Master Map of the Doppelschleuse model for Unreal Engine. Same as in Unity Engine, there are certain terminologies native to Unreal Engine [88]. The explanation of these terms eases with the understanding of this section. These terminologies are listed as follows:

1. Map: Is the gameplay area containing the environment that a player interacts with. It is also referred as Level.
2. Actor: Fundamental objects that is placed in a Level such as the camera
3. Component: functionality that can be added to an Actor

5.3.1 Master Map

Similar to the Master Scene (section 5.2.1) that was created in Unity Engine, a Master Map is created to accelerate the creation of dataset in Unreal Engine.

Upon starting a new Project in UE5 it offers templates for various game genres and a Blank Template. Blank template is chose to ensure that there aren't any particular game settings in the Project. The Project begins with a new Map consisting of Lighting, a PlayerStart (starting position of the Player) and HLOD (Hierarchical Level of Detail) [89] components in the Map. The Hierarchical Level of Detail (HLOD) system helps manage many Static Mesh Actors by merging them into a single proxy mesh and Material when viewed from afar. This decreases the number of Actors needing rendering, reducing draw calls per frame and improving performance.

Under Lighting we get DirectionalLight useful for creating sunlight; ExponentialHeightFog for creating fog volume, SkyAtmosphere a physically based sky rendering technique; SkyLight creates ambient light, SM_SkySphere another method for creating physically based sky and VolumetricCloud for the creation of clouds. Together they help in creating sun, sky, clouds and fog.

HDRI Backdrop Tool

Under the Lighting components group SM_SkySphere and VolumetricCloud are removed and a HDRI Backdrop tool [90] is added. The HDRI Backdrop is where the skybox (section 2.4) will be assigned. The default shape of the HDRI Backdrop mesh is of a dome called EnviroDome (as show in figure 5.10). The shape of the default skybox is not like the one discussed earlier in section 2.4. But it functions in the same manner as discussed. The same panoramic skybox material which takes the shape of a cubemap (fig. 2.5a)for Unity Engine, is optimised by Unreal Engine to take this EnviroDome shape. The skybox can be rotated within the HDRI Backdrop.

Figure 5.10: Shape of the HDRI Backdrop mesh

Fog

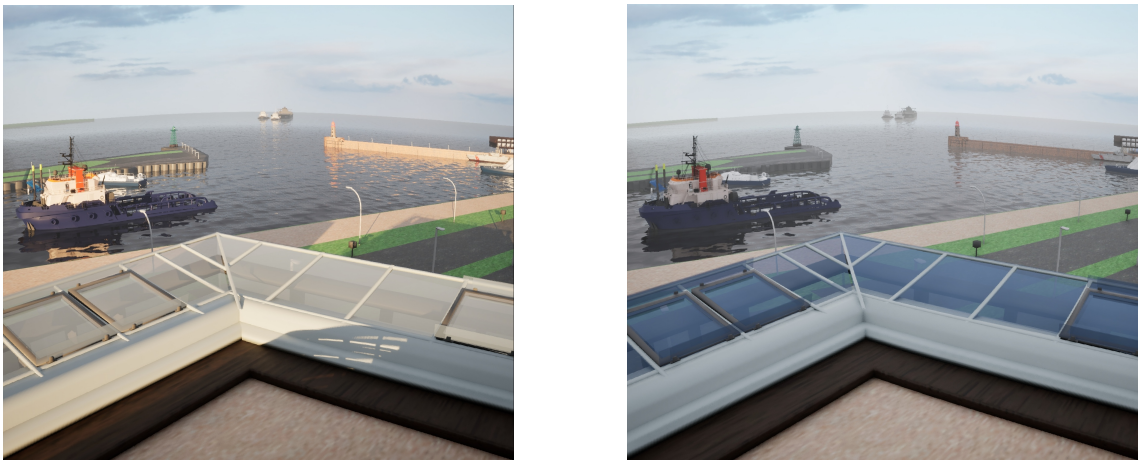
The Exponential Height Fog actor controls the density of the fog and it's falloff. For scenarios with high fog density, a value close to 0 is selected and the falloff is reduced as well.

Figure 5.11: Exponential Height Fog in UE5

DirectionalLight

The directional light component mimics the sunlight (fig. 5.12a). The rotation of the directional light facilities with the formation of shadows. The unit of its intensity is Lux [91]. However, the value does not correspond to the real-world light intensity unit Lux. For instance, the default value is 6 Lux, but 6 Lux in the real-world is the intensity of a light bulb. Colour temperature of the light can also be adjusted with the Temperature tab.

Upon removal of the DirectionaLight (fig. 5.12b), the SkyLight becomes the only source of light. This a diffused light illuminates the entire Map. This is how the images for the photorealism parameter experiment is created.



(a) with directional light (b) without directional light

Figure 5.12: Effect of directional light

Water System

Water system in Unreal Engine needs to be activated from the plugins panel. The Water body cannot be added directly into the Map, it requires a landscape. Thus, a landscape is added into the Map. It is made sure that the landscape is flat surface as the Doppelschleuse model will be placed on top of this landscape. A Water Body Ocean is introduced in the project outliner.

UE5 makes use of Gerstner waves [92] for the simulation of the water body. Their randomness, amplitude and the wavelength can be controlled under the properties of the Gerstner waves.

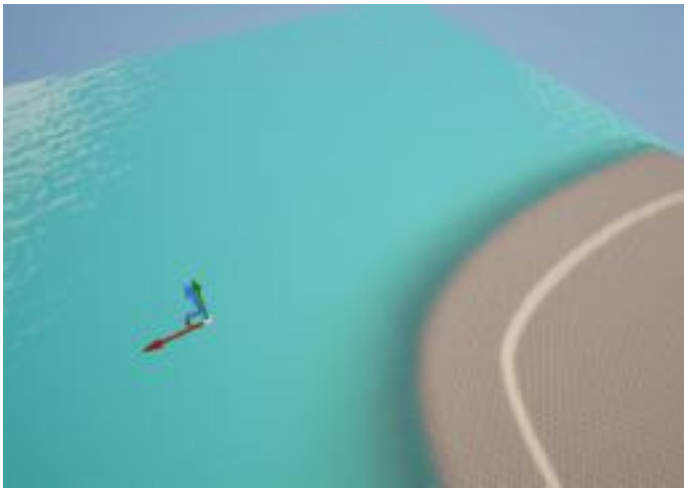


Figure 5.13: Water Body Ocean (left) and the Landscape(right)

Camera

In UE5, there is a camera assigned to the PlayerStart and spawns when the game is initialised. But in this use case, where images are needed to be recorded, UE5's Sequencer is more apt.

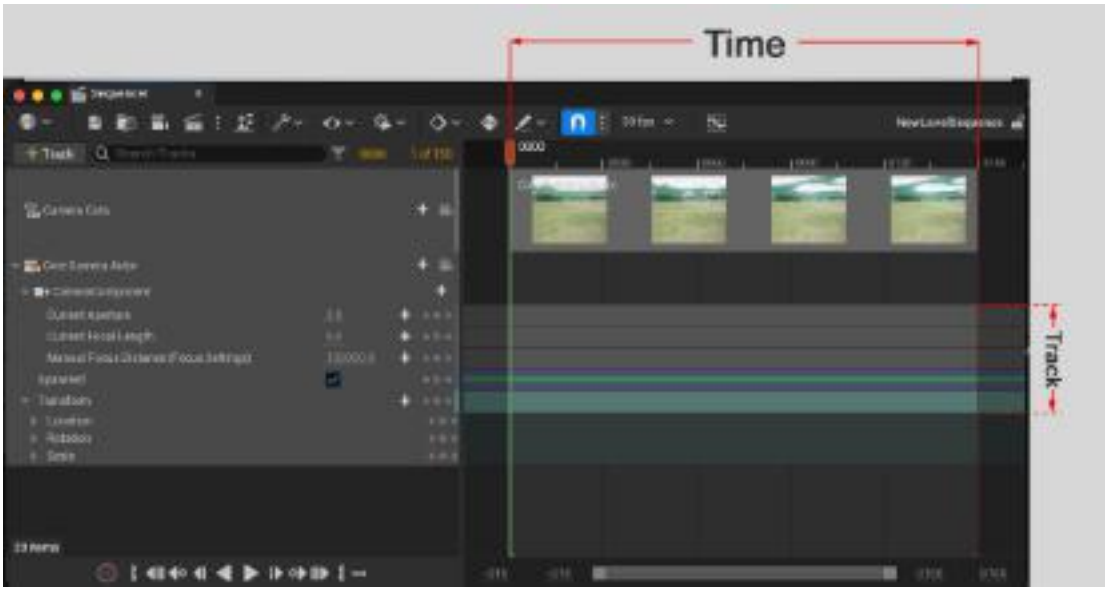


Figure 5.14: Tracks and Time in a Sequencer

The Sequencer (fig. 5.14) has tracks distributed over time like a video editing software. The Cine Camera Actor is added to the Sequencer and positioned on top of the AWI building. A Cine Camera Actor is replicated the real-world camera. Thus, the camera parameters (fig. 5.15) like the recording resolution and focal length, are assigned to the Cine Camera Actor. The output format, like the video file or image sequence, is set in the Sequencer and the recording time period is assigned in the Sequencer. (Note: Unity Engine has a similar tool called Recorder [93]. But, the Perception package captures the images along with the annotation masks. So the need for the Recorder becomes obsolete. Conversely, the Sequencer does not provide annotation like the Perception Package. More about this is explained in the section 6.2.2)

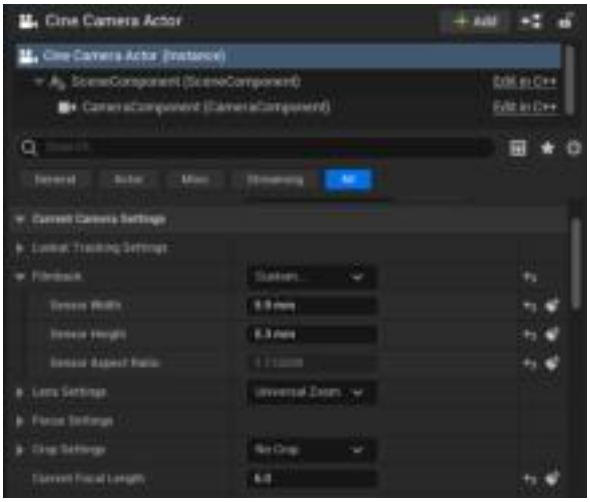
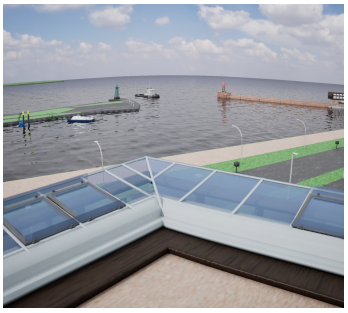


Figure 5.15: Cine Camera Actor

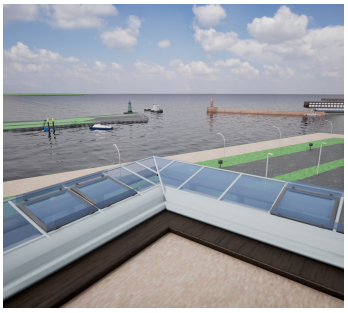
Colour grading (fig. 5.16a) is possible with the Cine Camera Actor. Thus, by reducing the saturation to 0, the recorded images become monochromatic. It is useful for the experimental dataset.

Lens distortion is not provided by the Cine Camera Actor. The distortion is added separately by creating a new material. The Material Domain of the lens distortion material is changed to Post Processing. Screen Position node is added and its variables are assigned. The output value

(a) Monochrome Image



(b) with lens distortion and colour



(c) No lens distortion

Figure 5.16: Effects of post-process volume

is connected to the Radial Falloff [94], which eventually links to the Emissive Color of the Lens Distortion material. An instance of this material is created. A post-process volume is added in the outliner and its extent is set to infinite so that the post processing works across the entire Map. The Lens Distortion Instance is assigned to the post processing volume resulting in an image with lens distortion (fig. 5.16b). For the photorealism effects dataset, the Lens Distortion is removed to form a planer image (fig. 5.16c) .

Models Setup

The Doppelschleuse model is added in the project. The option to generate materials while importing is disabled as it causes some issue with the scale of the materials. Instead after importing the model, the materials are added individually and assigned their respective positions.

As explained earlier (section 4.2.1), a new material for glass needs to be created. We start by creating a new material. The Blending Mode is changed from Opaque to Translucent, by doing so the Opacity option is activated. The Lighting Mode for Translucency is changed to Surface TranslucencyVolume to enable the Metallic and Roughness option.

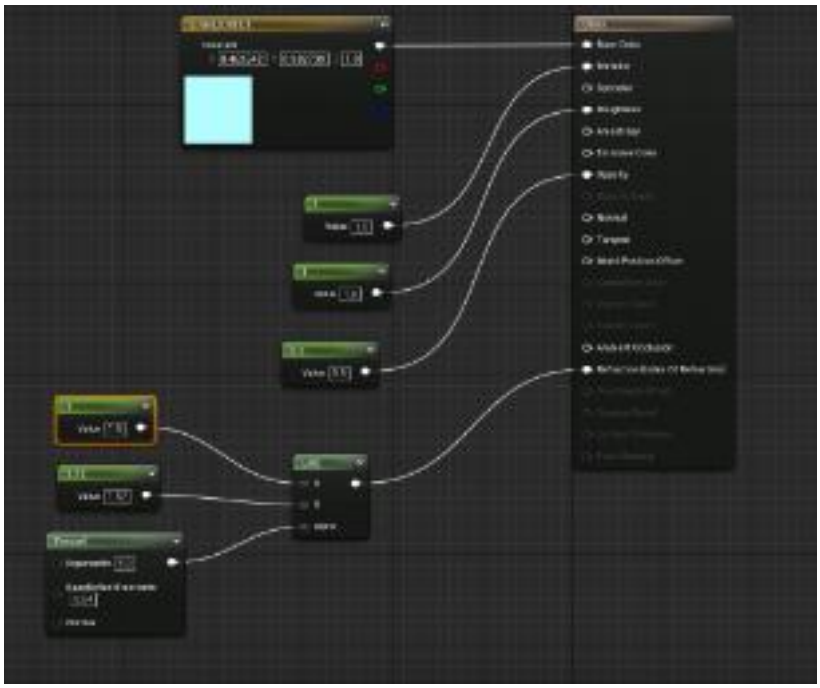


Figure 5.17: Material Node for glass material

In the Material Graph(fig. 5.17), a constant 3 Vector node is connected to the Base Color. The constant 3 Vector node holds the RGB value of the colour. Thus, the colour of the glass can be assigned from this. A variable constant node is assigned to each Metallic, Roughness and Opacity. A Fresnel node [95] that goes through the Lerp (Linear Interpolation) [96] node and into the Refraction (Index of Refraction). The A and B value of the Lerp node is assigned to 1 and 1.52 respectively. The value of the Index of Refraction for glass is provided by the developers [97]. The Fresnel node helps in adjusting the reflections on the material. An instance of the glass material is created and the instance is assigned to the exoskeleton of the AWI. Similarly, glass materials are created and assigned to the glass of the lighthouses and the Lotsenstation.

Vessel Models are added in the project like the Doppelschleuse model. Their materials and scale are implemented.

Chapter 6

Experimental Setup and Results

In Chapters 4 and 5, we have seen the creation of the Doppelschleuse 3D model and the methodology employed in both the game engines for the generation of synthetic datasets, respectively. This chapter aims to empirically compare Unity and Unreal engine and determine which one supports maritime computer vision task, like ship segmentation using ShipSG.

First, the pipeline for experimental evaluation is described. Second, we present the synthetic datasets generated with both engines, comprising images and segmentation annotations. To analyse the effects of photorealism, these synthetic validation datasets were generated multiple times, each time omitting different photorealistic effects. Creating varied validation sets allow to determine their contribution numerically for the task of ship segmentation. Third, we showcase the validation of the datasets using the instance segmentation model YOLOv8. Finally, the synthetic datasets that provide the best segmentation accuracy are used to augment the real ShipSG dataset and trained using YOLOv8, with the aim to demonstrate whether utilizing game engines can effectively support and enhance ship recognition tasks.

6.1 Experimental Evaluation Pipeline

Figure 6.1 illustrates the experimental evaluation pipeline designed in this thesis. A summary of the evaluation pipeline is as follows:

3D Model Creation

In Section 4.1 we have discussed the creation a true-to-scale model of the Doppelschleuse. This model is imported in both the games engines respectively, together with the 3D vessel models presented in Section 4.3.

Generation of Synthetic Datasets with Game Engines

In Section 5.2 and 5.3 a master template for both the game engines has been created, to facilitate the simulation process. With the help of the master template for both game engines, synthetic images that replicate several ShipSG images are rendered with varying photorealistic effects. Annotations for ship segmentation, meaning the mask contour of the ships and their, are also rendered for each synthetic image.

Validation with YOLOv8x

We use the synthetic rendered datasets and mask annotations to evaluate the performance of segmenting ships. The instance segmentation model YOLOv8x (largest version) was employed to validate these synthetic validation sets. The goal of this experiment is to which effects and which engine provide the best mAP, which is used to augment the real ShipSG training set for further evaluation.

Data Augmentation Experiment

From the previous step, the dataset of the game engine that performs the best in the comparison is selected for data augmentation. The synthetic images are further added to the training set of the ShipSG dataset and YOLOv8x is trained from scratch and evaluated on the purely real ShipSG validation set. This experiment allows to understand if utilising game engines can effectively support and enhance ship recognition tasks.

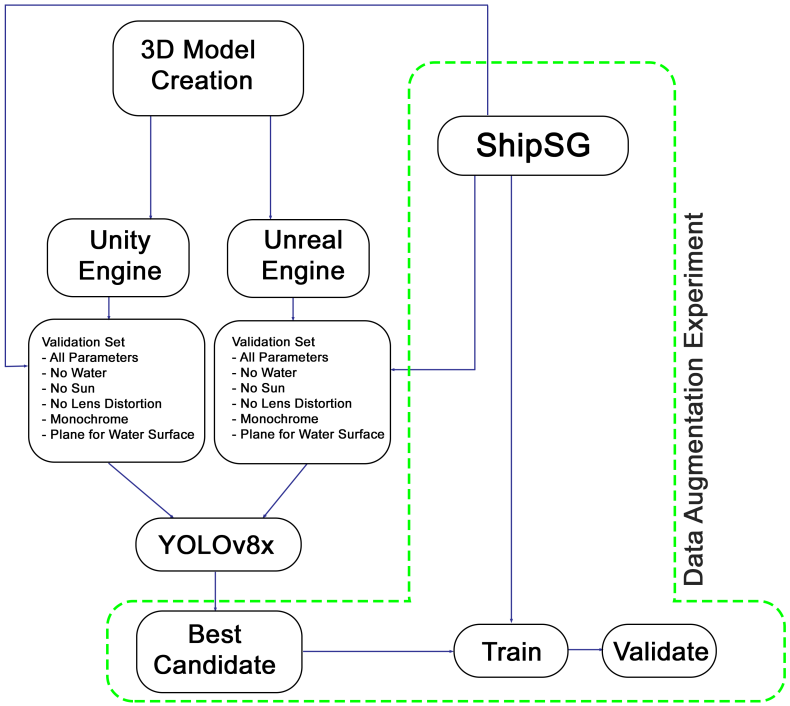


Figure 6.1: Experimental Evaluation Pipeline

6.2 Generation of Synthetic Datasets with Game Engines

Revisiting Chapter 3 Prior Work, the ShipSG dataset contains 3505 images of the Doppelschleuse. The dataset is split into two sets: training (80%, 2804 images) and validation (20%, 701 images). For the experimental evaluation of this thesis, 52 images of the real validation set of ShipSG were replicated, replacing the corresponding 52 real images out of the 701 images of the validation set of ShipSG. Figure 6.2 shows the distribution of the dataset with and without synthetic data.

Figure 6.2: Dataset Distribution for ShipSG and Synthetic datasets

To analyse effects for photorealism, the augmented validation set of the ShipSG with the 52 recreated images was generated 6 times per game engine. Each synthetic validation dataset, per game engine, attains different photorealistic effects, creating varied validation sets that will allow to determine numerically the effects that matter for the task of ship segmentation. The configurations for these effects are:

1. All effects present (sun light, water body, lens distortion and colours).
2. Absence of directional light (sun light).
3. Absence of the water body.

4. Absence of lens distortion (lens correction).
5. Absence of colour (monochromatic images).
6. Replacement the water body with a plane surface.

The first configuration contains all possible effects that would make the image photorealistic to the human eye. The subsequent configurations remove or reconfigure the corresponding effect using the first configuration as a baseline. The study of the absences of sun light, water body, lens distortion, and colour, aim to present their significance in the datasets for our maritime computer vision application, since their modelling involves increased manual modelling time by the practitioner creating the model. The last configuration aims to study the impact of water properties and the importance of using simulated water body, which includes physics (waves and reflections), in the implementation of this thesis. Using these configurations, the goal is to identify which configuration helps supporting ship segmentation tasks more effectively, which will be demonstrated along this chapter. The following subsections focus on the generation of each synthetic validation set per game engine and configuration.

6.2.1 Image and Annotations Rendering using Unity

The Master Scene presented in 5.2.1 serves as the starting point for the 52 images that are to be recreated. The Master Scene holds all the required GameObjects, which have been explained in Chapter 5. The modifications that allow to obtain the datasets per effect configuration are made in the Sky and Fog Volume, Sun, and the Water GameObjects based on the reference image from the ShipSG dataset. With the help of the vanishing point traced in section 5.1, we orient the camera in the Scene so that the vanishing point of the synthetic image lies at the same position as seen in the reference image (5.1).

For each of the 6 configurations, the vessels corresponding to the selected images from the validation set of ShipSG to be replicated are added in every scene. The position of the vessel in the Scene matches the reference image. Once each setup is complete, the scene is rendered with the Perception package (see section 5.2.1). This package supports both image and instance segmentation annotations. The first frame is captured 3 seconds after the simulation has started, allowing ample time for the game engine to refine the simulation. Consecutive image captures are performed after every 10 frames. The simulation is stopped manually after 10-15 seconds. Providing with sufficient fame captures along its segmentation mask. From the captured frames, the most suitable one and its corresponding segmentation mask is selected manually after visual inspection, based on similarity with the corresponding real image. Figure 6.3 summarizes the rendering process using Unity, of both images and automatic ship segmentation annotations.

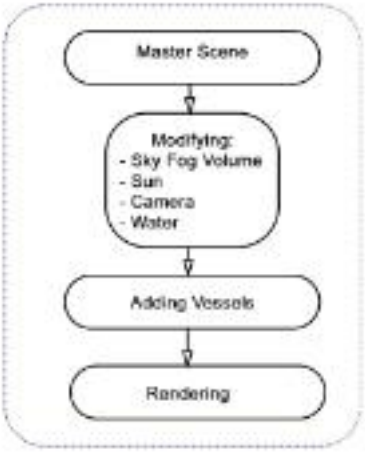


Figure 6.3: Unity rendering flow (images and mask annotations).

The generation process is repeated for all the 52 selected images and with the 6 photorealistic effects. Beyond the automatically generated ship mask annotations, manual mask annotations were created using CVAT (Computer Vision Annotation Tool)[98] for a more extensive evaluation. The ship categories on both the automatic and manual annotations are matched to the original corresponding class on each reference ShipSG image. The annotation mask for all synthetic images are converted into YOLOv8 format.

The process results on 6 datasets with auto generated annotations, and an additional 6 datasets with manual annotations. Both share the same images but the annotations differ, which will be used later with YOLOv8 to evaluate ship segmentation performance. Figure 6.6 shows a visual comparison of the Unity rendered images per configuration, together with each corresponding real ShipSG image, and automatic and manual annotations.

6.2.2 Image and Annotations Rendering using Unreal

The Master Map (section 5.3.1) presented in Section 5.3 serves as the starting point for the 52 images that are to be recreated. The Master Map holds all the required Actors with the defined components, which have been explained in Chapter 5. The modifications that allow to obtain the datasets per effect configuration are made in the HDRI Backdrop, Directional Light, Water System and Cine Camera Actor on the basis of the reference image from the ShipSG dataset. The orientation of the Cine Camera Actor is matched with the corresponding real image by ensuring that the position of the vanishing point in the synthetic image matches the vanishing point position in the real image(fig.5.1). Once these modifications are done, the corresponding vessel models are added to the Master Map. The vessels are positioned in the Master Map to match the reference images.

The Map is rendered after completion of the setup. The rendering is performed with the Sequencer. The first 3 seconds are dead frames, which means that the rendering will result in blacked out images, a default feature of Unreal. This is done to provide the UE5 with sufficient time to refine the simulation. The simulation continues for a total of 10 seconds. After the simulation ends, the most suitable image is selected manually after visual inspection, based on similarity with each corresponding real image. Figure 6.4 summarizes the rendering process using Unreal.

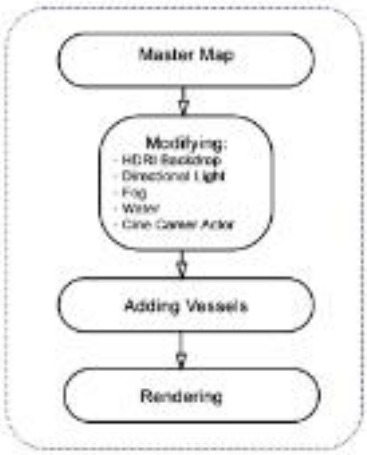


Figure 6.4: Unreal rendering flow (only images, automatic annotations not supported).

Unlike the Unity Engine, Unreal Engine 5 does not offer an out-of-the-box tool for automatic mask segmentation annotations. NVIDIA provides a Deep learning Dataset Synthesizer (NDDS)[99] plugin for annotations. However, support for the NDDS plugin ends at Unreal Engine 4.22. To ensure that the comparison utilises the latest versions of both game engines, Unreal Engine 5.3 is preferred over 4.22. Therefore, the ship masks are annotated manually using CVAT for the rendered Unreal images. The ship categories on the manual annotations are matched to the original corresponding class on each reference ShipSG image. The annotation mask for all

synthetic images are converted into YOLOv8 format. Figure 6.7 shows a visual comparison of the Unreal rendered images per configuration, together with each corresponding real ShipSG image, and manual annotations.

The result of the rendering with Unreal engine is 6 datasets, one per effect configuration, all with manual ship annotations.

6.2.3 Summary of Generated Datasets

As discussed in the beginning of Section 6.2, the 18 recreated synthetic datasets (12 for Unity and 6 for Unreal) (section 6.2.1, 6.2.2) are substituted for the real images in the ShipSG validation set. Thus, each of the new validation sets still contain 701 images, of which 52 of the total set are synthetic. The annotation mask for all synthetic images are converted into YOLOv8 format. The 18 validation sets are summarized in Table 6.1. For Unity Engine, datasets are categorised based on the annotation method into two groups: manually annotated and auto-annotated. Each of these two sets of the Unity Engine carries the same variation in the photorealistic effect.

Unreal Engine	**Unity Engine - auto annotations**	**Unity Engine - manual annotations**
All effects	All effects	All effects
No directional light	No directional light	No directional light
No water body	No water body	No water body
No lens distortion	No lens distortion	No lens distortion
Monochromatic	Monochromatic	Monochromatic
Plane surface for water body	Plane surface for water body	Plane surface for water body

Table 6.1: Rendered synthetic datasets for the experimental evaluation.

6.3 Validation Results

The instance segmentation method YOLOv8 is selected for ship segmentation validation of the generated synthetic datasets. Among the YOLOv8 possible models, YOLOv8x is chosen given its high performance on the real ShipSG dataset [45]. We used the YOLOv8x weights resulting of training ShipSG (only real images) as described in [45]. These weights, when used for validation of the real validation set, provide a mAP of 76.5%, which is used as the benchmark value for the synthetic datasets. The weights trained on the real ShipSG training set are now used to validate the generated sets that contain the above described synthetic images. A summary of the validation can be seen in Table 6.2. And examples of inferred ship masks on the synthetic images can be found in figures 6.5, 6.6, 6.7.

Configuration	**mAP (%)**		
	Unreal (manual ann.)	**Unity (auto ann.)**	**Unity (manual ann.)**
All effects	71.7	62.7	**72.3**
No directional light	71.5	62.0	69.3
No water body	71.5	62.7	69.3
No lens distortion	71.3	63.7	**72.3**
Monochromatic	71.0	62.3	69.6
Plane surface for water body	72.0	62.2	69.6

Table 6.2: Validation results of YOLOv8x trained on real ShipSG and validated on the synthetic datasets. Unity contains both auto and manual annotations, while Unreal only manual.

None of the game engines surpass the mAP 76.5% of the real dataset, which was expected, since the YOLOv8x was both trained and validated on real images. The goal of this experiment, however, is to identify which game engine achieves the closest mAP to the mAP of the real dataset, when using synthetic images. In this regard, mAP on Unity Engine with manual annotations (all effects) outperforms Unreal Engine with manual annotations (all effects) by 0.5%. Despite exhibiting a higher mAP consistently, Unreal Engine could not surpass the highest mAP of the Unity Engine. This suggests that Unity Engine's synthetic images with manual annotations with all effects and no lens distortion are much closer to the real-world images and annotations. In the case of Unity with manual and auto-annotations, there is a nearly 10% decrease in mAP between the auto-annotated

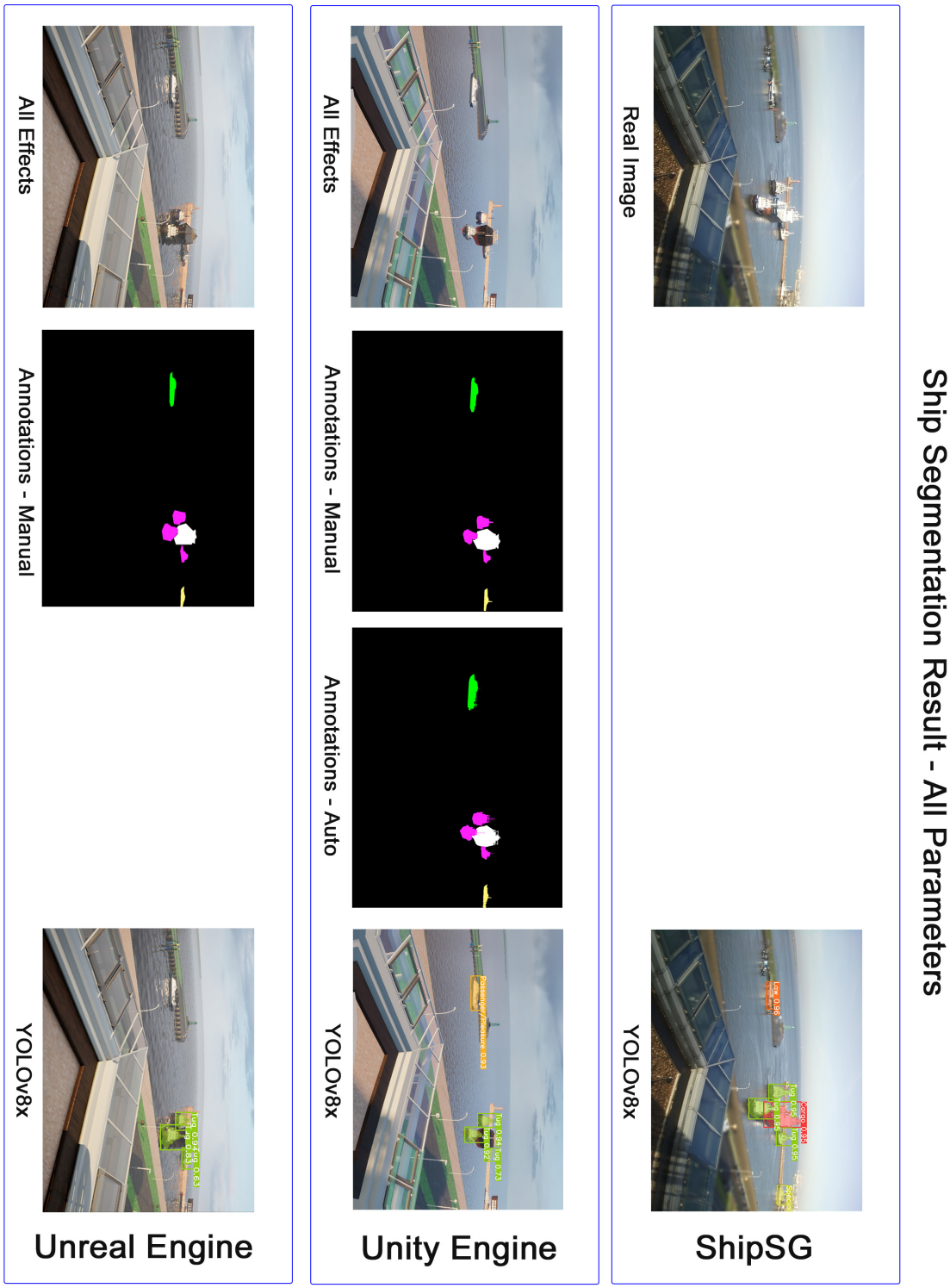


Figure 6.5: Segmentation Results - Unity and Unreal - All Parameters

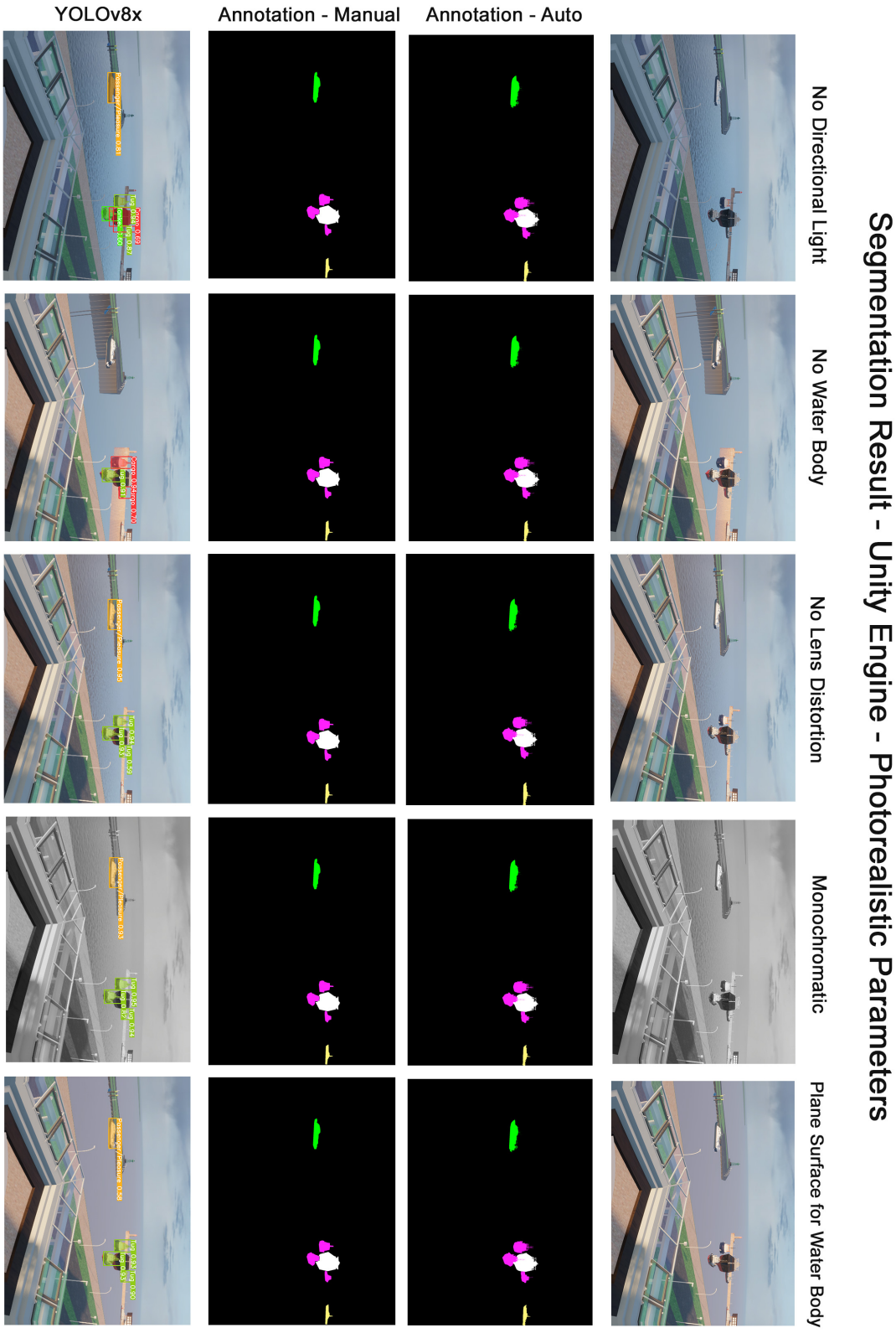


Figure 6.6: Segmentation Results - Unity - Photorealism Parameters

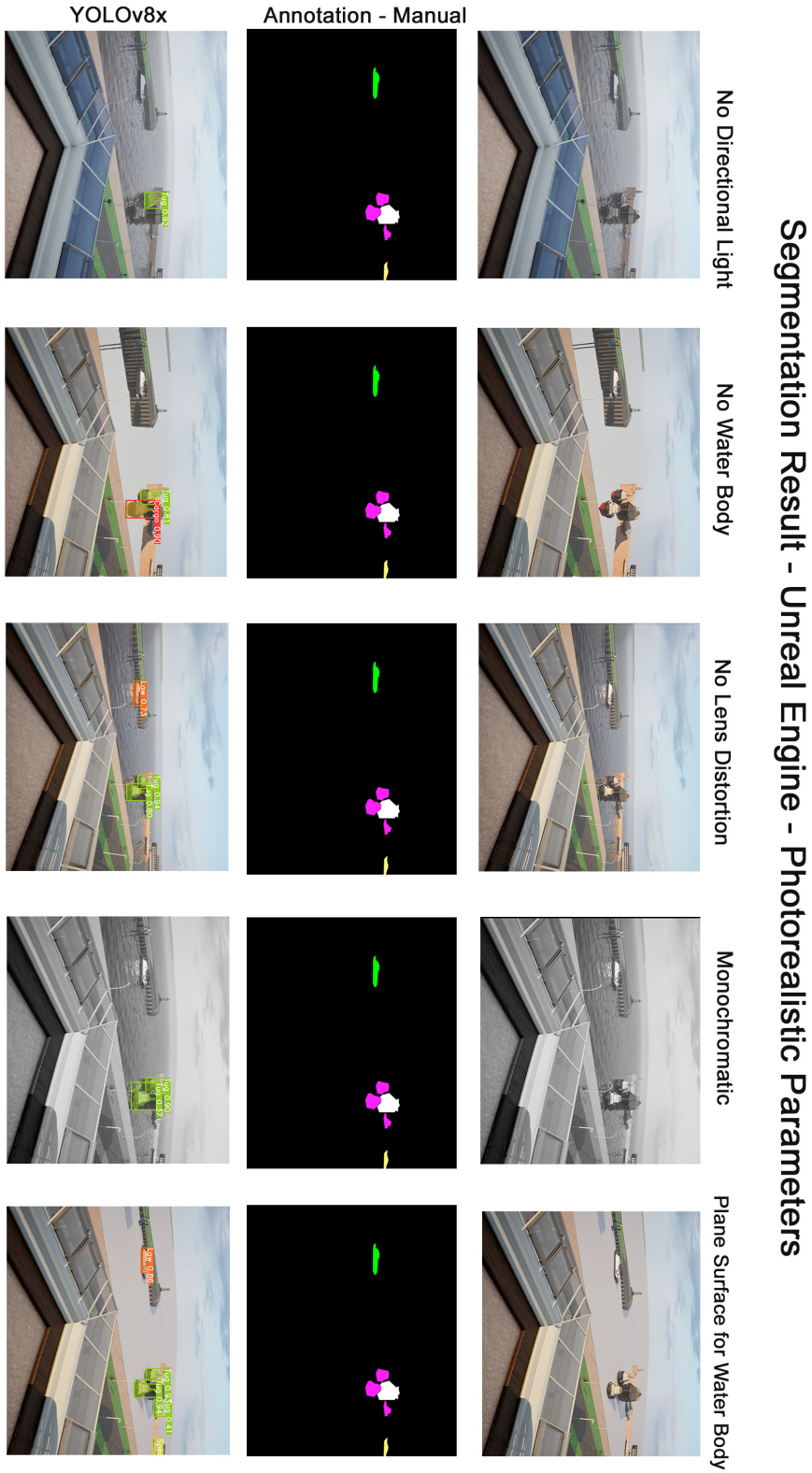


Figure 6.7: Segmentation Results - Unreal - Photorealism Parameters

and manually annotated synthetic images, which highlights the significant role annotations play in influencing both quality and subsequent model performance. Figure 6.8 depicts an example of a synthetic image rendered with Unity and the auto and manual annotations.

(a) Auto-Annotations (b) Synthetic Image (c) Manual Annotations

Figure 6.8: Visual comparison of manual and automatic annotation of ship masks on the Synthetic image. The colors of the masks represent different ship categories. - Unity Engine

We can note from fig. 6.8 that the manual annotations for synthetic images match better with the real-world annotations. This is because, specifically, only the visible portion of the vessel above the water surface is annotated in the manual annotation (fig. 6.8b, 6.8c). Conversely, Unity Engine's auto annotations overlooked the submerged vessel hull and annotates the entire vessel, disregarding water (fig. 6.8a). As a result, the generated annotation masks of both the scenarios (with and without the water body) are same. This is evident from the results obtained in both scenarios, with and without the water body (Unity auto-annotations) remain the same. This is an aspect that needs to be considered by the Unity Engine practitioners when generating similar datasets and annotations, as both the auto annotations toolkit [84] and water generation package [82] are still experimental. The presence of the ship mask annotation under water is not segmented by YOLOv8, and when compared to the ground truth, a decrease in mAP is produced since the mask would be considered as incomplete.

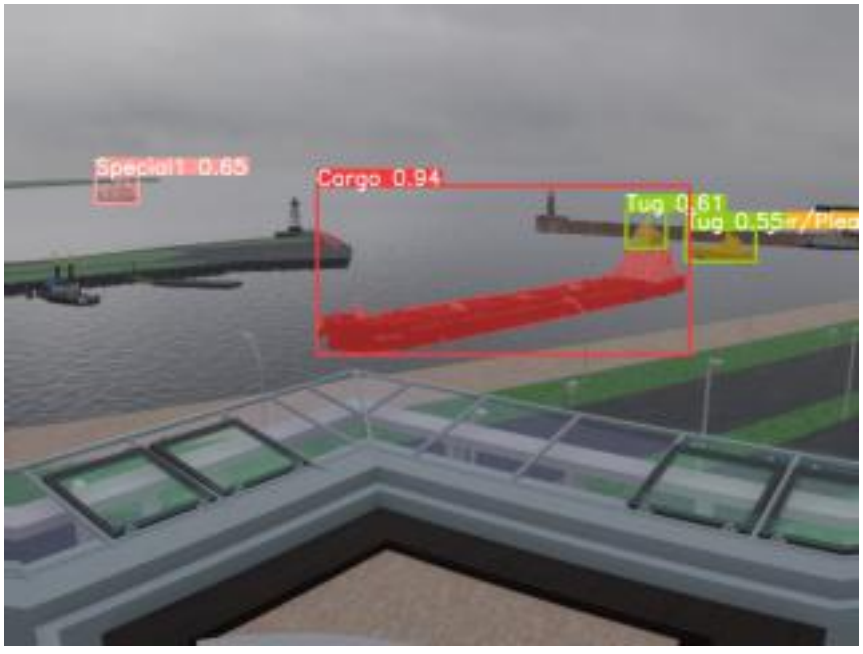


Figure 6.9: YOLOv8x on Unity generated image. False Positive appears without directional light (Bridge of the cargo is identified as a Tug boat).

The resulting mAPs per removal of individual effects from the scenes also show that these effects contribute to the realism of the synthetic images. The monochrome dataset for both the

engines shows a decline in the mAP. This decline is substantial (2.7%) for Unity Engine (manual annotation) than Unreal Engine (0.7%) and persists by 0.4% with Unity Engine's auto annotations. Thus, this highlights the importance of coloured images in instance segmentation. For Unity Engine (with both the annotations), the decline continues further by 0.3% in the absences of sun. But for Unreal Engine without sun, the mAP drops by 0.2% compared to sun's presence. The absence of a strong directional light from the sun eliminates the casting of shadows by the objects in the scene. This causes false positives, as seen in (fig. 6.9). Thus, the sun is significantly important.

(a) No Water Body (b) With Water Body (c) Plane Surface

Figure 6.10: Synthetic Images - Unreal Engine

As discussed earlier in this section, the dataset is trained with real images where the vessels are partially submerged in water. Thus, it is hard to see an entire vessel when it is in water (fig. 6.10b). Upon removing the water body (fig. 6.10a), the vessels are visible entirely and confuses the segmentation algorithm. As a result, performance of Unreal Engine and Unity Engine (manual annotations) the falls than in the presences of water. When a plane surface substitutes for the water surface (fig. 6.10c), the mAP of the Unity Engine with auto annotations decreases by 0.5% from 62.7% than in the absence of water. On the contrary, the mAP for Unity Engine (manual annotations) and Unreal Engine increases by 0.3% and 0.5% with respect to the results obtained in the absence of water. Unreal Engine achieves its highest performance (of 72% mAP) when a plane surface replaces the water. Eliminating the visual properties (such as reflection) exhibited by the water, eases the segmentation for images generated in Unreal Engine. In the absence of lens distortion, Unreal Engine experiences a drop of 0.4% than in the presence of lens distortion. Unity Engine (manual annotation) retains its peak performance of 72.3%. The mAP performance of Unity Engine (auto annotations) reaches 63.7% without lens distortion.

Taking into account the results mentioned earlier, the two best candidates for the data augmentation experiment (fig. 6.1), are Unity Engine with all effects and Unity Engine without lens distortion, both with manual annotations. Their performance (mAP 72.3%) with ship segmentation stands the closest to the benchmark of 76.5% set by the ShipSG dataset.

6.4 Data Augmentation Experiment

Expanding upon the previous section's results, the mAP performance of the manually annotated Unity Engine dataset with all the photo-real effects and without lens distortion stands the highest (72.3%). Therefore, these two datasets are selected for Data Augmentation Experiment (fig.6.1).

The experiment consists in augmenting the training set of ShipSG by adding synthetic images, train YOLOv8, and validate in the real-only (original) ShipSG validation set. The goal of this experiment is to determine whether these configurations a useful to augment real maritime datasets for ship segmentation applications.

The experiment is repeated twice, once per best candidate implementation (**all effects** and **no lens distortion**). Since the synthetic images are recreations of 52 images of the real validation set, the corresponding real-images from the validation set of the ShipSG are removed. Consequently, the training set comprises 2856 images (2804 original + 52 synthetic), while the validation set contains 649 images (instead of the original 701). Additionally, their corresponding annotations of the synthetic images have been added to the training set.

For a fair comparison with the baseline given by [45], the training of YOLOv8x using the synthetically augmented datasets use the same initial conditions as presented in [45]. This means, random weight initialization and training during 300 epochs. Therefore, the training is comparable to the training originally performed with the real images in the the work presented by [45]. Table

YOLOv8x training	**mAP (%)**
Original ShipSG [45] (baseline)	76.5
Unity Engine (manual ann.& all effects)	**76.9**
Unity Engine (manual ann. & no lens distortion)	76.7

Table 6.3: Results of the Data Augmentation Experiment after training YOLOV8x with and without synthetic images.

Following the training of YOLOv8x on the new datasets, the configuration with all effects present exhibits a mAP increase of 0.4% (Table 6.3) compared to YOLOv8x performance on ShipSG without the utilisation of synthetic data. The dataset without lens distortion also showcases an increase of 0.2% mAP. These increments, though modest due to the low number of synthetic images utilised compared with the total dataset volume, are significant and highlight the value of synthetic data in refining the model for real-world maritime computer vision tasks.

Chapter 7

Conclusion

This thesis explored the effectiveness of game engines in generating photorealistic synthetic datasets for the improvement of maritime computer vision applications, particularly focusing on the comparison between Unity Engine and Unreal Engine. Scenarios from the real-world maritime dataset ShipSG were reconstructed in the game engines. To help with the reconstruction, a true-to-scale 3D model of the Doppelschleuse was created in 3D modelling software Blender. The key structures in the Doppelschleuse were identified and manually crafted. These structures include the buildings, roads and lighthouses. The created model was set up in the respective game engines, along with the corresponding vessel model, sky and water body. A total of 52 images from the ShipSG validation set, with their corresponding scenario and vessels, were recreated and rendered in both the game engines. To analyse effects for photorealism, the synthetic images were generated 6 times per game engine. Each synthetic validation dataset, per game engine, attains different photorealistic effects. These effects are: sun light, water body, lens distortion and colours, and allowed the numerical comparison of photorealism in both engines. Ship masks present in the rendered images generated by the Unity Engine were annotated twice, once with its auto-annotation toolkit, and a second time manually. Annotations for the Unreal Engine images were done manually as it does not offer any annotation toolkit.

Each of the 18 synthetic sets of images replaced their original counterpart in the validation set of ShipSG. The validation was performed with the instance segmentation model YOLOv8. Specifically, YOLOv8x, the model with the highest accuracy on the real validation set of ShipSG, with mAP of 75.6%, was used. The Unity Engine with manual annotations demonstrated superior performance over Unreal Engine, evidenced by an improvement of mAP of 72.3% compared to Unreal's 71.7%. When comparing Unity with auto-annotations and manual-annotations, we found a decrease of mAP in the auto-annotations of 10%. The drop in mAP in auto annotations is caused by Unity's annotations tool marking the entire ship hull, unlike real images where the hull is underwater and not visible. Manual annotations result in better replication of those found in real dataset. Despite having a pixel-perfect annotations, the fact that the performance of the auto-annotated dataset with Unity is subpar, highlights the improvements that practitioners of the Unity Engine should take into account when generating synthetic maritime datasets.

Moreover, the validation with each individual photorealistic effect showcases their contribution towards making maritime synthetic images close to real. The removal of a directional light (sunlight) resulted in a decrease of the mAP for both the engines, and false positives in the segmentation of ships. The absence of a water body caused a decline in performance for all the models. Adding a flat surface for the water improved the performance compared to its absence, and Unreal engine performs as its best in this case. For Unity engine, the performance with flat surface for the water is still lower than the realistic water body. This shows that the default water simulation in Unreal underperforms in our application compared to Unity's. In maritime computer vision applications, this is a critical consideration, given that water simulation is central to the maritime domain. Image properties, like chromaticity and lens distortion, have an impact as well.

The data augmentation experiment further demonstrated the practical utility of synthetic data when combined with real images for training. By integrating synthetic images into the training set of ShipSG, and after retraining YOLOv8x, there was a noticeable improvement in model perfor-

mance, evidenced by an increase in mAP from 76.5% to 76.9% when the augmented data was used. This increment, though modest due to the low number of synthetic images utilised compared with the total dataset volume, underscores the value of synthetic datasets in enhancing the robustness and accuracy of maritime computer vision models.

Overall, this thesis provides a systematic comparison of the efficacy of game engines in synthetic image generation for maritime computer vision applications. While studying individually each photorealism effect, their role and combination has be shown crucial in generating synthetic images that closely mimic real images. The increment in performance when used for further training signifies the value of synthetic data in refining the model for real-world maritime computer vision tasks. This outcome emphasizes the critical role of carefully selecting the correct setup for crafting synthetic datasets.

Chapter 8

Future Work

A significant insight from this thesis is the critical role of annotations in synthetic dataset quality. While the auto-annotation toolkit (Perception package) offered by the Unity Engine substantially reduces time and effort in annotation processes, it does not achieve the accuracy of manual annotations. Future work should focus on improving the outcome of this toolkit to ensure that auto-generated annotations can match the precision of manual methods. Enhancements to the auto-annotation processes are essential for enabling wider adoption and reliability in practical applications, especially when scaling up the volume of data that requires annotation.

Another pivotal area for future research is the exploration of water physics in synthetic maritime environments. The study has highlighted the importance of further developing photorealistic water simulations but there is still scope for improving the water simulation such as the water ripples the vessels cast in their motion. There is a need to extend this to more complex scenarios, such as the depiction of stormy weather conditions, which could significantly influence the realism and applicative value of synthetic datasets in maritime computer vision. Investigating these aspects would help in creating more accurate and diverse training datasets that are crucial for improving model robustness under various environmental conditions.

Lastly, this thesis has demonstrated positive outcomes from replacing real images with synthetic ones, though in limited number. To further validate and expand upon these findings, future studies should experiment with integrating a larger set of synthetic images into training datasets. This expansion would provide more comprehensive insights into the scalability and effectiveness of synthetic data in enhancing the training and performance of computer vision models. The impact of a broader application of synthetic datasets across different training scenarios and model types would be invaluable in understanding the full potential of synthetic data in maritime and other computer vision domains.

Appendix A

Appendix

Lotsenstation

(a) Width proportions

(b) Height proportions

Figure A.1: Proportions of the Lotsenstation

Panoramic Skyboxes

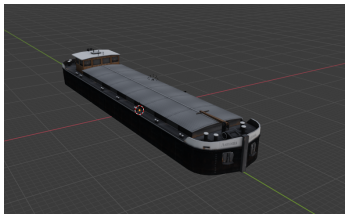
Figure A.2: Panoramic Skybox-3
Source [100]

Figure A.3: Panoramic Skybox-4
Source [101]

Figure A.4: Panoramic Skybox-2
Source [102]

All Vessel Models

(a) Vessel 1



(b) Vessel 2

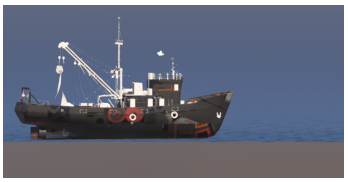
(c) Vessel 3

(d) Vessel 4

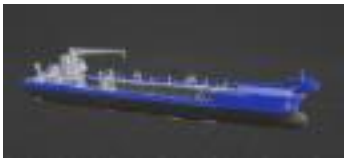
(e) Vessel 5

(f) Vessel 6

(g) Vessel 7

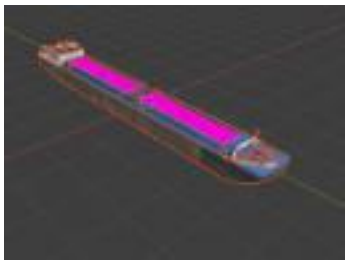


(h) Vessel 8



(i) Vessel 9

(j) Vessel 10



(k) Vessel 11

(l) Vessel 12

Figure A.5: Vessels (12 of 21)

Link to the Synthetic Dataset Images

You can find all the synthetic images here: Link

(a) Vessel 13 (b) Vessel 14 (c) Vessel 15

(d) Vessel 16 (e) Vessel 17 (f) Vessel 18

(g) Vessel 19 (h) Vessel 20 (i) Vessel 21

Figure A.6: Vessels (9 of 21)

(a) Vessel 22 (b) Vessel 23

Figure A.7: Vessel 10 and 15 with new materials

Bibliography

[1] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[2] Dilip K. Prasad et al. "Video Processing From Electro-Optical Sensors for Object Detection and Tracking in a Maritime Environment: A Survey". In: *IEEE Transactions on Intelligent Transportation Systems* 18.8 (2017), pp. 1993–2016. DOI: 10.1109/TITS.2016.2634580.

[3] Jonathan Becktor et al. "Bolstering Maritime Object Detection with Synthetic Data". In: *IFAC-PapersOnLine* 55.31 (2022). 14th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2022, pp. 64–69. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2022.10.410. URL: https://www.sciencedirect.com/science/article/pii/S2405896322024569.

[4] Benjamin Kiefer, David Ott, and Andreas Zell. *Leveraging Synthetic Data in Object Detection on Unmanned Aerial Vehicles*. 2021. arXiv: 2112.12252 [cs.CV].

[5] Angela Dai et al. "ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes". In: *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*. 2017.

[6] Xiaomin Lin et al. *SeaDroneSim: Simulation of Aerial Images for Detection of Objects Above Water*. 2022. arXiv: 2210.16107 [cs.CV].

[7] *Lens Distortion Types*. URL: https://lenspire.zeiss.com/photo/app/uploads/2022/02/technical-article-distortion.pdf.

[8] *Lens Distortion*. URL: https://reolink.com/blog/lens-distortion/#:~:text=Wide%2Dangle%20lens%20distortion%2C%20also,compared%20with%20standard%20lens%20distortion..

[9] *Focal Length and Field of View Explained in 4 Steps*. URL: https://www.photographytalk.com/beginner-photography-tips/7204-focal-length-and-field-of-view-explained-in-4-steps.

[10] Luis Ramírez-Hernández et al. "Improve three-dimensional point localization accuracy in stereo vision systems using a novel camera calibration method". In: *International Journal of Advanced Robotic Systems* 17 (Jan. 2020), p. 172988141989671. DOI: 10.1177/1729881419896717.

[11] *Blender*. URL: https://www.blender.org.

[12] *Blender Measure Tool*. URL: https://docs.blender.org/manual/en/latest/editors/3dview/toolbar/measure.html.

[13] *Blender Extrude Tool*. URL: https://docs.blender.org/manual/en/2.80/modeling/meshes/editing/duplicating/extrude.html.

[14] *Blender Knife Tool*. URL: https://docs.blender.org/manual/en/2.80/modeling/meshes/editing/subdividing/knife.html.

[15] *Blender Camera*. URL: https://docs.blender.org/manual/en/latest/render/cameras.html.

[16] *Blender Light*. URL: https://docs.blender.org/manual/en/latest/render/lights/light_object.html.

[17] *Blender Bake Material*. URL: https://docs.blender.org/manual/en/latest/render/cycles/baking.html.

[18] *Blender Shader Editor*. URL: https://docs.blender.org/manual/en/latest/editors/shader_editor.html.

[19] *Blender Image Texture Node*. URL: https://docs.blender.org/manual/en/latest/render/shader_nodes/textures/image.html.

[20] *Blender Principaled BSDF*. URL: https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/principled.html.

[21] *Blender Procedural Material*. URL: https://docs.blender.org/manual/en/2.79/render/blender_render/textures/types/procedural/introduction.html.

[22] *Blueprints Unreal Engine*. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/introduction-to-blueprints-visual-scripting-in-unreal-engine.

[23] *Visual Scripting Unity Engine*. URL: https://unity.com/features/unity-visual-scripting.

[24] Antonín Šmíd. "Comparison of Unity and Unreal Engine". 2017.

[25] *Occulsion Culling*. URL: [cite:%20https://developer.nvidia.com/gpugems/gpugems/part-v-performance-and-practicalities/chapter-29-efficient-occlusion-culling#:~:text=Occlusion%20culling%20increases%20rendering%20performance,query%20and%20early-z%20rejection].

[26] *Ubisoft*. URL: https://www.ubisoft.com/en-us/.

[27] *Rockstar Games*. URL: https://www.rockstargames.com.

[28] *Unity Engine*. URL: https://unity.com.

[29] *What is the best game engine: is Unity right for you?* URL: https://www.gamesindustry.biz/what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you.

[30] *Unity Engine URP*. URL: https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@17.0/manual/index.html.

[31] *Unity Engine HDRP*. URL: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@17.0/manual/index.html.

[32] *Unity Pricing and License*. URL: https://unity.com/pricing#plans-individualsandteams.

[33] *Most successful videogame engine*. URL: https://www.guinnessworldrecords.com/world-records/most-successful-game-engine.

[34] *Unreal Engine*. URL: https://www.unrealengine.com/en-US.

[35] *Unreal Game 1998*. URL: https://en.wikipedia.org/wiki/Unreal_(1998_video_game).

[36] *Unreal Pricing*. URL: https://www.unrealengine.com/en-US/blog/we-are-updating-unreal-engine-twinmotion-and-realitycapture-pricing-in-late-april.

[37] *Procedural Skybox Unity Engine*. URL: https://docs.unity3d.com/Manual/shader-skybox-procedural.html.

[38] *Sky Atmosphere Unreal Engine*. URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/sky-atmosphere-component-in-unreal-engine.

[39] *Kloofendal 48d Partly Cloudy (Pure Sky)*. URL: https://polyhaven.com/a/kloofendal_48d_partly_cloudy_puresky.

[40] *Polyhaven*. URL: https://polyhaven.com.

[41] Microsoft. *COCO Dataset Dataset*. https://universe.roboflow.com/microsoft/coco. Open Source Dataset. visited on 2024-03-11. 2024. URL: https://universe.roboflow.com/microsoft/coco.

[42] Zhenfeng Shao et al. "SeaShips: A Large-Scale Precisely Annotated Dataset for Ship Detection". In: *IEEE Transactions on Multimedia* 20.10 (2018), pp. 2593–2604. DOI: 10.1109/TMM.2018.2865686.

[43] Xinqiang Chen et al. "Video-Based Detection Infrastructure Enhancement for Automated Ship Recognition and Behavior Analysis". In: *Journal of Advanced Transportation* 2020 (Jan. 2020), pp. 1–12. DOI: 10.1155/2020/7194342.

[44] Borja Carrillo-Perez, Sarah Barnes, and Maurice Stephan. "Ship Segmentation and Georeferencing from Static Oblique View Images". In: *Sensors* 22.7 (2022). ISSN: 1424-8220. DOI: 10.3390/s22072713. URL: https://www.mdpi.com/1424-8220/22/7/2713.

[45] Borja Carrillo-Perez et al. "Improving YOLOv8 with Scattering Transform and Attention for Maritime Awareness". In: Sept. 2023, pp. 1–6. DOI: 10.1109/ISPA58351.2023.10279352.

[46] *What is YOLOv8? The Ultimate Guide. [2024].* URL: https://blog.roboflow.com/whats-new-in-yolov8/.

[47] Simon Baker and Rick Szeliski. "A Database and Evaluation Methodology for Optical Flow". In: *Proceedings of the IEEE International Conference on Computer Vision*. IEEE Computer Society, 2007. URL: https://www.microsoft.com/en-us/research/publication/a-database-and-evaluation-methodology-for-optical-flow/.

[48] Jonathan Stets et al. "Comparing Spectral Bands for Object Detection at Sea using Convolutional Neural Networks". In: *Journal of Physics: Conference Series* 1357 (Oct. 2019), p. 012036. DOI: 10.1088/1742-6596/1357/1/012036.

[49] D. J. Butler et al. "A naturalistic open source movie for optical flow evaluation". In: *European Conf. on Computer Vision (ECCV)*. Ed. by A. Fitzgibbon et al. (Eds.) Part IV, LNCS 7577. Springer-Verlag, Oct. 2012, pp. 611–625.

[50] Alberto Garcia-Garcia et al. "The RobotriX: An eXtremely Photorealistic and Very-Large-Scale Indoor Dataset of Sequences with Robot Trajectories and Interactions". In: *CoRR* abs/1901.06514 (2019). arXiv: 1901.06514. URL: http://arxiv.org/abs/1901.06514.

[51] Pablo Martinez-Gonzalez et al. "UnrealROX+: An Improved Tool for Acquiring Synthetic Data from Virtual 3D Environments". In: *CoRR* abs/2104.11776 (2021). arXiv: 2104.11776. URL: https://arxiv.org/abs/2104.11776.

[52] Mike Roberts and Nathan Paczan. "Hypersim: A Photorealistic Synthetic Dataset for Holistic Indoor Scene Understanding". In: *CoRR* abs/2011.02523 (2020). arXiv: 2011.02523. URL: https://arxiv.org/abs/2011.02523.

[53] *Sintel Film (2010).* URL: https://en.wikipedia.org/wiki/Sintel.

[54] Stephan R. Richter et al. "Playing for Data: Ground Truth from Computer Games". In: *European Conference on Computer Vision (ECCV)*. Ed. by Bastian Leibe et al. Vol. 9906. LNCS. Springer International Publishing, 2016, pp. 102–118.

[55] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[56] *Virtual Traffic Simulator.* URL: https://www.informatik.uni-bremen.de/projekttag/2022/en/projekte/virtual-traffic-simulator/.

[57] Shital Shah et al. "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles". In: *Field and Service Robotics*. 2017. eprint: arXiv:1705.05065. URL: https://arxiv.org/abs/1705.05065.

[58] Fan Jiang and Qi Hao. "Pavilion: Bridging Photo-Realism and Robotics". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 8285–8290. DOI: 10.1109/ICRA.2019.8794235.

[59] *Robotic Operating System.* URL: https://www.ros.org.

[60] Ramiz Salama and Mohamed Elsayed. "A live comparison between Unity and Unreal game engines". In: *Global Journal of Information Technology: Emerging Technologies* 11 (Apr. 2021), pp. 01–07. DOI: 10.18844/gjit.v11i1.5288.

[61] Hussain Ali Juma Al Lawati. "The Path of UNITY or the Path of UNREAL? A Comparative Study on Suitability for Game Development". In: (). DOI: 10.47611/jsr.vi.976. URL: https://www.jsr.org/index.php/path/article/view/976.

[62] Paul E. Dickson et al. “An Experience-based Comparison of Unity and Unreal for a Standalone 3D Game Development Course”. In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE ’17. Bologna, Italy: Association for Computing Machinery, 2017, pp. 70–75. ISBN: 9781450347044. DOI: 10.1145/3059009.3059013. URL: https://doi.org/10.1145/3059009.3059013.

[63] *Google Maps*. URL: https://www.google.com/maps.

[64] *Material Sand*. URL: https://polyhaven.com/a/asphalt_floor.

[65] *Material Grass*. URL: https://ambientcg.com/view?id=Grass001.

[66] *Material Gravel*. URL: https://ambientcg.com/view?id=Gravel038.

[67] *Material Asphalt*. URL: https://polyhaven.com/a/aerial_asphalt_01.

[68] *Ambient CG*. URL: https://ambientcg.com/.

[69] *Molenfeuer Bremerhaven-Geestemündung*. URL: https://www.deutsche-leuchtfeuer.de/binnen/weser/bremerhaven-geeste_molenfeuer.html.

[70] *Blender Glass BSDF*. URL: https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/glass.html.

[71] *Prodcdural Brick Blender*. URL: https://docs.blender.org/manual/en/latest/render/shader_nodes/textures/brick.html.

[72] *FBX Format*. URL: cite:%20https://en.wikipedia.org/wiki/FBX.

[73] *Hand Coordinte System*. URL: https://www.evl.uic.edu/ralph/508S98/coordinates.html.

[74] *Turbosquid*. URL: https://www.turbosquid.com.

[75] *Raspberry Pi Camera*. URL: https://www.reichelt.de/de/en/raspberry-pi-camera-12mp-75--rasp-cam-hq-p276919.html?PROVID=2788&gclid=CjwKCAjw44mlBhAQEiwAqP3eVpApaUJedxAhaBwE&&r=1.

[76] *Raspberry Pi Camera Lens*. URL: https://www.reichelt.de/de/en/raspberry-pi-6mm-camera-lens-wide-angle-rpiz-cam-6mm-ww-p276922.html?GROUPID=9006&START=0&OFFSET=16&SID=960055722f562d65442c5e23d44cf1a58dca9f3fded4bd6b219b0&LANGUAGE=EN&&r=1.

[77] *Unity Engine Terminologies*. URL: https://docs.unity3d.com/Manual/Glossary.html#SectionGeneral.

[78] *Exposure Unity Engine*. URL: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@14.0/manual/Override-Exposure.html.

[79] *Light Unity Engine*. URL: https://unity.com/how-to/lights-shadows-with-hdrp#:~:text=and%20exposure%20levels-,Physical%20Light%20Units%20and%20intensity,equal%20one%20meter%20for%20accuracy.

[80] *Rendering and Post Processing Unity Engine*. URL: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@17.0/manual/rendering-and-post-processing.html.

[81] *Water System Unity Engine*. URL: https://blog.unity.com/engine-platform/new-hdrp-water-system-in-2022-lts-and-2023-1.

[82] *Water System Unity Engine Documetation*. URL: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@14.0/manual/WaterSystem-use.html.

[83] *Script Interactions Water System Unity Engine*. URL: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@14.0/manual/WaterSystem-scripting.html.

[84] Unity Technologies. *Unity Perception Package*. https://github.com/Unity-Technologies/com.unity.perception. 2020.

[85] *Perception Package - Synthetic Optimized Labeled Objects (SOLO) Dataset Schema.* URL: https://docs.unity3d.com/Packages/com.unity.perception@1.0/manual/Schema/SoloSchema.html.

[86] *YOLO Format.* URL: https://docs.ultralytics.com/datasets/segment/.

[87] *Creating Glass Material in UNity Engine.* URL: https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@15.0/manual/refraction-use.html.

[88] *Unreal Engine Terminologies.* URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-terminology?application_version=5.1.

[89] *Unreal Engine HLOD.* URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/hierarchical-level-of-detail-in-unreal-engine.

[90] *HDRI Backdrop Unreal Engine.* URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/hdri-backdrop-visualization-tool-in-unreal-engine?application_version=5.3.

[91] *Light Unit Unreal Engine.* URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/using-physical-lighting-units-in-unreal-engine?application_version=5.3.

[92] *Gerstner Waves Unreal Engine.* URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/simulating-waves-using-the-water-waves-asset-in-unreal-engine?application_version=5.3.

[93] *Recorder Unity Engine.* URL: https://unitytech.github.io/unity-recorder/manual/index.html.

[94] *Radial Falloff Unreal Enigne.* URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/BlueprintAPI/Field/SetRadialFalloff?application_version=5.3.

[95] *Frenel Node UE5.* URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/using-fresnel-in-your-unreal-engine-materials.

[96] *Lerp Node UE5.* URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/BlueprintAPI/Math/Float/Lerp?application_version=5.3.

[97] *Index of Refraction Unreal Engine.* URL: https://dev.epicgames.com/documentation/en-us/unreal-engine/using-refraction-in-unreal-engine.

[98] *CVAT (Computer Vision Annotation Tool).* URL: https://www.cvat.ai.

[99] *NVIDIA Deep learning Dataset Synthesizer (NDDS).* URL: https://github.com/NVIDIA/Dataset_Synthesizer.

[100] *Kloofendal Misty Morning (Pure Sky).* URL: https://polyhaven.com/a/kloofendal_misty_morning_puresky.

[101] *Industrial Sunset (Pure Sky).* URL: https://polyhaven.com/a/industrial_sunset_puresky.

[102] *Mud Road (Pure Sky).* URL: https://polyhaven.com/a/mud_road_puresky.