Linköping University | Department of Computer and Information Science
Bachelor's thesis, 16 ECTS | Datateknik
2023 | LIU-IDA/LITH-EX-G--23/061--SE

Radar and sea clutter simulation with Unity 3D game engine

Simulering av radar och sjöklotter med Unity 3D-spelmotor

Mikael Johnsson
Linus Bergman

Supervisor : Birgitta Thorslund
Examiner : Aseel Berglund

External supervisor : Håkan Göransson

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Sammanfattning

Spelmotorer är välkända för sin användning inom spelindustrin men har också fått genomslag inom andra områden. Arkitektur, fordonsindustrin och försvarsindustrin använder idag dessa verktyg för att visualisera och till viss mån, även testa sina produkter. I detta examensarbete har vi undersökt hur spelmotorn Unity kan användas för att simulera en radar i syfte att detektera och mäta sjöklotter. Efter en förstudie där olika implementeringsmetoder undersöktes, beslutades det att använda strålspårning (eng. ray tracing). Själva radarn simuleras genom att använda kameraobjektet i Unity för att sända ut strålar. Bakom kameran finns ett planobjekt som fungerar som mottagare. Strålar spåras sedan individuellt för varje pixel och sprider sig genom en given scen. Samtidigt sparas information såsom träffkoordinater, den totala färdsträckan samt riktning. Genom att använda det totala färdavståndet för varje stråle som återvänt till mottagaren kan fasen för varje stråle beräknas. Detta kan sedan användas för att beräkna den totala returnerade amplituden, vilket motsvarar den returnerade signalstyrkan. Med hjälp av en "compute shader" kan databeräkningarna göras parallellt av GPU:n vilket underlättar när så många strålar ska spåras.

Eftersom syftet med uppsatsen var mätning av simulerat sjöklotter, genomfördes tester för att mäta på ett simulerat hav. Havsytorna hade två olika sjöstadier, vilka genererades med Phillips-spektrumet för att få realistiska vågor. Ett fartygsobjekt testades sedan i frirymd och sedan även i de två olika havsytorna. Amplituden och mängden strålar som returnerades användes för att bestämma den totala returnerade signalstyrkan och "Radar Cross Section" (RCS) för objektet. Syftet med detta var att kunna jämföra med andra studier gällande sjöklotter, både simulerade som verklighetsbaserade och avgöra om vårt tillvägagångssätt kunde resultera i ett användbart verktyg för branschen.

De olika amplituder och antalet strålar som vi fick tillbaka varierade beroende på vilka vinklar och havsytor som användes. Vissa resultat var inte realistiska jämfört med verkliga mätningar av sjöklotter. Det beror främst på våra nuvarande begränsningar i att inte kunna spåra en tillräckligt stor och tillräckligt detaljerad havsyta, vilket behövs för att mätningarna ska vara mer realistiska.

Däremot matchade vi några resultat med de från en liknande studie, där verktyget OKTAL, som är ett professionellt radarsimuleringsverktyg, användes. Detta i kombination med möjligheterna för en förbättrad implementation tyder på att användningen av en spelmotor som Unity är ett intressant verktyg värd att vidareutforska radarsimuleringar med.

Abstract

Game engines are well known for their use in the gaming industry but are starting to have an impact in other areas as well. Architecture, automotive, and the defence industry are today using these engines to visualise and, to some extent, test their products. In this thesis, we have examined how the game engine Unity could be used for simulating a radar with the purpose of detecting and measuring sea clutter. Following a pre-study examining different implementation approaches, it was decided to use ray tracing. The radar itself is simulated by using the camera to emit rays and having a plane object directly behind it act as a receiver. Rays are then individually traced for each pixel, propagating throughout the scene and saving information such as hit coordinates, distance travelled, and direction. By using the total travel distance of each ray that returned to the receiver, the phase of each ray is calculated. This is then used to compute the total amplitude, which represents the returned signal strength. Using a compute shader, most of the computations are done in parallel on the GPU, enabling millions of rays to be traced.

As measuring sea clutter was an objective of the study, tests measuring the ocean were carried out. These used ocean surfaces with two different sea states, using the Phillips spectrum to generate realistic waves. A ship object was then tested in free space and on two different ocean surfaces. The calculated amplitude and the number of rays returned were used to determine the signal strength returned and the RCS of the object. The purpose of this was to compare with other results of sea clutter studied, observed both in the real world and in simulated scenarios, and determine if our approach could be a valid choice for the industry.

Some results matched the findings of a similar study that used a professional radar simulation tool called OKTAL. Other results of sea clutter were found to not be realistic due to certain limitations. The current main limitation of our implementation is not being able to trace a large enough ocean surface with the finer details needed for realistic results. However, this could be solved by creating a better implementation.

These findings suggest that simulating radar and sea clutter in Unity is a feasible approach worth continuing to explore.

Acknowledgments

We want to thank Saab Dynamics for allowing us to do our bachelor's thesis at their facility in Linköping. We would like to thank Håkan and Fredrik at Saab, who helped with analysing some of our findings and allowed us to proceed with our research. We also want to thank our supervisor, Birgitta Thorslund at Linköping University, for her valuable input when writing and conducting our thesis.

Abbreviation

BVH ..... Bounding Volume Hierarchy

CPU ..... Central Processing Unit

dB ..... Decibel

EM ..... Electromagnetic

FFT ..... Fast Fourier Transform

FOI ..... Totalförsvarets forskningsinstitution

GA ..... Geometrical Acoustics

GO ..... Geometrical Optics

GPGPU ..... General-Purpose Graphics Processing Unit

GPU ..... Graphics Processing Unit

HDRP ..... High Definition Rendering Pipeline

LiDAR ..... Light Detection and Ranging

LTS ..... Long-term Support

MMW ..... Millimeter Wave

PM ..... Pierson-Moskowitz

PO ..... Physical Optics

RCS ..... Radar Cross Section

SBR ..... Shooting and Bouncing Rays

SDK ..... Software Development Kit

SS ..... Sea State

UAV ..... Unmanned Aerial Vehicle

Contents

List of Figures

List of Tables

1 Introduction

Today, as game engines provide realistic graphics and powerful tools, industries outside of gaming are beginning to use them[1]. Industries such as architecture, automotive, aeronautics, and defence industries use them for visualisation, building 3D environments, and simulating real-world products and scenarios. This study examines how radar and sea clutter can be simulated using the Unity game engine.

Radar is a well-studied field with years of research, continuously developing in the context of defence industries but also in the area of autonomous vehicles such as unmanned aerial vehicles (UAVs) and cars using a similar technique, Light detection and ranging (LiDAR). A problem that occurs with radar is clutter, which is false signals caused by the varying surface of the ground or sea [1]. Simulating radar and radar clutter is part of developing radar systems and clarifying their outputs. It is not a trivial task, as it requires both modelling a realistic radar simulation and a realistic surface. The surface of the ground is difficult to model as it can be heavily varied. Our aim is therefore to simulate radar noises caused by the sea surface when detecting objects in the ocean.

Using a game engine for this purpose is a relevant proposition, potentially increasing its usability for the defence industry. Since both light and radar are electromagnetic (EM) waves with similar propagation behaviours, it should be possible to simulate radar in the same way light is simulated. There are different methods of simulating light and its propagation. The approach usually differs depending on which method is used to generate the graphics, with rasterization or ray tracing. Both rasterization and ray tracing are usually implemented using shaders, programs that run on the graphics card. In short, light calculated using a rasterization-based method is more of a numerical approach, whereas ray tracing physically traverses the scene in order to simulate the propagation. In this thesis, when a shader-based implementation is mentioned, we refer to it as an implementation that is not using ray tracing. Another potential method that was reviewed was using the built-in audio system. It is a relevant approach, as radar works on the same principle as echoes from sound. These three approaches, a shader-based implementation, ray tracing, and audio, were studied, and we ultimately chose one of them to create a radar implementation in Unity.

[1] Unity. *Real-Time Solutions, Endless Opportunities*. Accessed: 08.03.2023. URL: `https://unity.com/solutions`.

So far, there has been some research using ray tracing in game engines to simulate radar or LiDAR by creating a plugin for a game engine [2], [3]. Other studies implement ray tracing without the use of a game engine to simulate radar [4], [5], or sonar [6]. Some studies have implemented a radar simulation with other methods than ray tracing, such as using the Phong lighting model [7], [8], [9]. The effect ocean surfaces have on radar return when measuring the radar signature of different objects has been simulated using ray tracing in a paper by Andersson [10].

1.1 Requirement specification

We were given a task by Saab Dynamics in Linköping to study if and how sea clutter could be simulated by using another tool than the existing ones being used at the company, for example, a game engine. They wanted us to examine three different approaches: a shader-based implementation, ray tracing, and audio, choosing one of them for the implementation. By measuring the results, a conclusion could be drawn about whether using a game engine for this purpose is feasible and how well it compares to real-world results. For this thesis, we focused on using Unity as the testing environment in our research.

1.1.1 Saab Dynamics AB

Saab Dynamics AB is part of the Saab corporate group and is a Swedish company with a main focus on developing high-technology creative solutions and construction for the defence industry. Saab Dynamics AB focuses on missile systems, fire support systems, and military and civilian underwater crafts, which can be both manned and unmanned.

1.2 Aim

This study aimed to examine how Unity could be used to simulate sea clutter caused by using radar when detecting objects. By examining three different approaches: ray tracing, shaders, and audio, the aim was to reach a conclusion regarding which method was most suitable. A more conclusive evaluation was then made by implementing the chosen approach and comparing it to other studies of sea clutter.

1.3 Research question

- How can Unity be used to simulate sea clutter and how accurate are the results?

1.4 Approach

After learning more about the core concepts of the paper and evaluating related works, a pre-study was conducted where each approach was evaluated. We had three methods to choose from: one was a shader-based implementation using a lighting model, another was ray tracing, and the third was using audio. Related works regarding the implementation of radar simulations using these three methods mentioned, were analysed and evaluated. A method was then chosen to be implemented in Unity. The method chosen was based on how well-suited the approach was for our purpose, it should be estimated to have a reasonable time of implementation and preferably primarily use the graphics processing unit (GPU) for making all the heavy calculations.

In our testing environment, we created a scene in Unity with a radar that emitted a number of rays (as the radar pulse waves) and measured their propagation behaviour. The scene

either contained some objects or a sea surface that reflected the rays. By summarising the returned rays, characteristics such as the signal strength and radar cross section (RCS) were calculated.

Results from our tests were then compared to both empirical studies and other papers that have similar traits to our work but whose methods and testing environments are different from ours.

1.5 Delimitations

We have chosen the Unity 3D game engine as our experimental environment to set up our scenes. There is a possibility that other game engines like Unreal Engine might have a different performance given the same scenario and parameters, but it is outside of the scope of this thesis. We also used some existing implementations that were free to import into a Unity project, for instance, the simulated ocean and the boat object. This was helpful in giving us more time to conduct tests and write the paper.

We have conducted a pre-study to determine which method is the best candidate for us to implement within the given time we had writing this thesis. We chose to implement ray tracing in Unity, and our evaluation is based on the findings from this method. In this thesis, when a shader-based implementation is mentioned, it implies an implementation that is not using ray tracing.

1.6 Outline

The thesis is structured in the following way: An introduction Chapter 1 to the subject and our aim with the study, along with our research question. In Chapter 2 we discuss relevant topics and present theories worth knowing about for the readers of this paper. In Chapter 3, related works are presented, and how our study fits among these earlier works is discussed. Then in Chapter 4, it is explained how we conducted our experiments and what we wanted to achieve with them. In the implementation Chapter 5, we present more details about the chosen method and how we implemented it in the game engine. In Chapter 6, our results are displayed, and following that chapter is the discussion 7, where the data is analysed. In the last Chapter 8, some conclusions are made based on the theory presented earlier with regards to our results, and we also answer our research question.

2 Background

This chapter explains relevant topics for our study, starting with an introduction to game engines and Unity. Technical topics related to the implementation, such as the basics of shaders and ray tracing, will be explained, followed by explanations and some definitions of how radar, sea clutter, and audio waves work.

2.1 Game engines

Freiknecht et al. [11] explain the origin of the game engine, which was created as a tool to simplify the process of making a game. Instead of having to rebuild entire games from the ground up, game engines provide the basis upon which the game can be built. They provide the game developer with a graphical interface in which all the features needed to create a game are found. Tools for animation, level building, simulating physics, graphics rendering, as well as sound and input mechanisms, along with other features.

The game engine chosen for this study is Unity[2], and they market their platform as a tool not only for game development but also for other uses. A common use case, like visualising and simulating real-life data, can be used in many industries. Some examples are the automotive, aerospace, and defence industries, where such simulations can be used to develop and test new products. By creating what Unity calls "Digital Twins"[3], a digital copy of a real-world product is made. By injecting real-world data sets, the results of the simulated model could give insightful and helpful guidance to the user for the actual product manufactured.

2.1.1 Unity

Unity is a platform used for the 2D and 3D development of games and applications. The basic concepts of Unity are a "Scene" and a "GameObject". A scene is the virtual 2D or 3D environment where content is contained. Any given object in the scene is called a "GameObject". These can consist of several components defining visual and physical characteristics and behaviours. Furthermore, custom scripts can be created to further define the behaviour

[2]Unity. *Real-Time Solutions, Endless Opportunities*. Accessed: 08.03.2023. URL: `https://unity.com/solutions`.

[3]Unity. *Digital Twins*. Accessed 08.03.2023. URL: `https://unity.com/solutions/digital-twins`.

of the scene and the content within it.

The game engine itself is written in C++, and the current version 2021.3 (LTS) natively supports scripting in C#[4]. Unity supports many different platforms and is used in a large variety of games and applications, both for desktop and mobile devices. Because of the wide availability of tutorials and free content from its asset store, it is a popular engine also for educational purposes [12].

2.1.2 Raycast

In Unity, a Raycast is a ray used to detect collisions in a scene. By specifying an origin and a direction, information such as the distance from the starting point to the point of collision can be obtained[5].

2.2 Shaders

Computer graphics are generated by the graphical processing unit (GPU). The rendering pipeline consists of all the processes the GPU does in order to produce a frame. Shaders are programs used in the rendering pipeline where mesh, i.e., vertex data, turns into shapes and images. Several shaders can be used in the rendering pipeline, however, the vertex shader and the fragment shader are mandatory. A vertex shader uses the specified mesh to place vertices in the correct place on the screen. Lines are then drawn between each vertex, and the rasterization stage calculates the data that each pixel, called a fragment, should contain. The fragment shader then has the responsibility to fill the colour of these fragments, which will determine how the surface of the shape will look [13].

A compute shader is a type of shader used to perform calculations on the GPU, also known as "general-purpose computing on graphics processing units" (GPGPU). They are therefore not necessarily used in the render pipeline but can be used to, for example, perform more advanced light calculations.[6].

2.3 Ray tracing

Ray tracing is a technique that has been implemented in many modern applications to render realistic images and videos. Ray tracing is effective when reproducing optical effects such as light scattering, reflections, and refraction. There are different types of algorithms and implementations depending on the intended use and effect [14].

A common algorithm that is used is the shooting and bouncing rays (SBR) or “brute-force” [15]. Rays are emitted, or "shot", from the camera into the scene. When the ray interacts with another object, it bounces and gets another trajectory depending on the object’s surface area. The computational cost increases with the number of rays emitted, how many bounces a ray is allowed, and the complexity of the scene. There are techniques used to reduce the computational cost, enabling faster and more complex computations. A common method is using an acceleration structure called "Bounding Volume Hierarchy" (BVH)[7]. Using this technique,

[4]Unity. *Creating and Using Scripts*. Accessed 08.03.2023. URL: https://docs.unity3d.com/2021.3/Documentation/Manual/CreatingAndUsingScripts.html.

[5]Unity. *Physics.Raycast*. Accessed 04.04.2023. URL: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html.

[6]Unity. *Compute shaders*. Accessed 21.04.2023. URL: https://docs.unity3d.com/Manual/class-ComputeShader.html.

[7]Nvidia. *Ray Tracing*. Accessed 05.06.2023. URL: https://developer.nvidia.com/discover/ray-tracing.

the scene is divided into several boxes or volumes, ordered as a tree structure. Rays are then traced through the structure using a tree traversal technique, such as depth-first, instead of being traced against each triangle. Using the GPU for these types of calculations is optimal since ray tracing is a problem well suited for parallel computations.

The advantage of using ray tracing is that it can provide path-loss information, in other words, if anything is making the signal decline or if an object is in the way. It is also possible to provide data like the object's angular spread, time delay, and distances to another target, which are all important factors when considering the use of radar equipment [14].

2.4 Radar

Radar is the shorter term for "Radio Detection and Ranging". It works on the same principle as an echo. More specifically, it sends signals in the form of EM waves, and then it measures the returned signals. Since these waves travel at a constant speed, the distance to an object can be calculated using the time it takes for the wave to return. The direction and elevation of the object are derived from measuring the horizontal angle, i.e., azimuth, and altitude, i.e., elevation angle, of the received signal. For this purpose, a type of radar called "pulse radar" is used. It sends a signal and waits a short time before sending another signal. This type of application is usually used for military purposes [16].

Radar operates on different bands, depending on the situation. In Barton [1, p. 6], there is an extensive table listing radar band usage, but we will focus on the X-band, which is the frequencies between 8 and 12 GHz, which has been widely used for open sea, tracking ships, and such in military surveillance.

2.4.1 Radar cross section

The radar cross section (RCS), also known as the radar signature, is important for radar detection since a high RCS means that the object is easier to detect. RCS is affected by the size, material, radar-absorbing paint, and smoothness of the object. A large container ship is therefore easier to detect due to its metal material and rectangular shape than a stealth ship, which has pointy surfaces and non-reflective material. This is because the stealth ship design aims at scattering or absorbing the radar signals, making it have a lower RCS and thus be harder to detect.

2.4.2 Radar signal strength formula

The classic radar equation in 2.1 returns the power, P_e, measured in Watts (W), of a signal from a radar pulse. P_s is the transmitted power, G is the antenna gain, i.e., a factor in which signal strength is increased, lambda λ is the radar wavelength in metres (m), and sigma σ is the RCS. These are mostly constant values, whereas the range, R, is for the object range.

$$P_e = \frac{P_s G^2 \sigma \lambda^2}{(4\pi)^3 R^4} \tag{2.1}$$

The final implementation of our radar simulation did not use the formula 2.1. Some variables, such as the distance, did not need to be calculated since they were given by the nature of our approach. After discussing with our supervisor at Saab, the amplitude, meaning the signal strength of a wave, was calculated. A high amplitude equals a high signal strength. By first calculating the phase for each ray returned, the following equation was used: 2.2.

$$\textit{Total received signal } s_r = A * \sin(\omega_0 t + \phi_r) \Rightarrow A = \sqrt{I_{tot}^2 + O_{tot}^2} \tag{2.2}$$

The sine and cosine waves could cancel each other out if the phase of one wave is shifted 180 degrees from another wave. This is called interference[8], and is why the phase is needed to calculate the total amplitude of multiple waves. What this means for the radar return is that some objects can be hard to track if the rays' total in- and out-phases interfere with each other. We will go further in explaining the equation 2.2 in Section 5.3.

2.5 Sea clutter

When target detection with radar occurs, there are a lot of things other than the intended objects that will reflect some type of noise that will be picked up by the radar for each pulse emitted. This noise can be from vegetation, buildings, structures, or even the sea. This type of noise, or backscatter, is known as clutter, and our main focus will be on sea clutter in this thesis. Barton describes in his book that depending on certain sea state (SS) numbers, the sea surface conditions can be determined, which include the height of the waves and the wind speed, among others [1, pp. 109–110]. Barton presents in Table [1, Tab. 3.2], where we can read the sea surface parameters for SS levels between 0 and 8, which have been converted to metrics that he has derived from the work of Nathanson [17], [1, pp. 109–110]. Furthermore, the sea surface is composed of water with an average salinity of 35 ppt.[9] and this will have an effect on the reflectivity of a water surface and therefore have an impact on the amount of noise that the radar is detecting. In a report by Andersson [10], it was noted that capillary waves also have a large impact with regard to sea clutter. Capillary waves are centimetre-large waves that are on top of other waves. It is further mentioned that these are highly complex to model and are an area in need of further research.

Some known empirical sea clutter modelling for computing reflectiveness are the GIT model (Horst 1978, as cited in Greger-Hansen) [18] [19], the Hybrid model [20], the TSC model developed by Technology Service Corporation [21], and the NRL model [18].

Yang et al. [22] discuss these models and how they are able to measure high sea states. They are pointing out that different empirical models seem to be good at matching empirical data depending on different types of radar incidence angles. But for sea states of levels 5-7, TSC, NRL, and Morchin could be extended in their use of predicting sea clutter reflectivity.

The sea clutter, like all clutter, can be a problem when target detecting because there can be circumstances that make the target almost impossible to distinguish from ambient noises. Barton [1] describes ways of adjusting for the problems that might impact the radar signal, but this is not an issue that we will address in this paper.

2.6 Ocean wave simulation

To simulate waves and wave formations, different techniques exist, and to mention a few of these models, Phillips Pierson-Moskowitz (PM) and the JONSWAP spectrum. It should be noted that from the PM equation have been derived two other spectra, the Bretschneider spectrum and the Ochi spectrum, which have replaced PM since these formulations consider additional parameters and therefore allow a more precise calculation of how the ocean waves behave. But in the interest of time, we have used in our simulations an implementation of the Phillips spectrum, which is based on the paper by Tessendorf [23]. In short, the surface is modelled by multiple sine waves with different speeds and amplitudes, using a spectrum

[8] George N. Gibson, Ph.D. *5.2 Constructive and Destructive Interference*. Accessed 05.06.2023. URL: https://www.phys.uconn.edu/~gibson/Notes/Section5_2/Sec5_2.htm.

[9] MatLab. *Maritime Radar Sea Clutter Modeling*. Accessed 12.04.2023. URL: https://jp.mathworks.com/help/radar/ug/sea-clutter-simulation-for-maritime-radar-system.html.

based on statistical data measured from real-world oceans. These are then summarised using the Fast Fourier Transform (FFT), resulting in a surface that closely approximates empirical studies of a real-world ocean.

With the help of empirical data on sea clutter done by Nathanson in 1991 [17], we might be able to compare our simulations with the ones registered from this work. The data might be from a while ago, but the implications remain, and it is used and referenced in many other studies, books, and papers, some of which we have mentioned in chapters 1 and 2. If we can get a similar result, then there exists a potential use for Unity as an application for modelling radar signals and the types of clutter that might occur when hitting different types of objects, like ocean waves when trying to detect a ship.

2.7 Audio

Audio moves through the air with a wave movement, just like an EM wave does, and can bounce on surfaces as well, which makes it an interesting candidate to include in our thesis. While the propagation of sound is similar to that of EM waves, the frequency of a sound wave is lower with a wavelength of about 17 mm to 17 m compared to the frequency and wavelength of an EM wave, commonly used by radar, of 8–12 GHz and 3 cm. EM waves, depending on the kind of source material, will be able to move through certain materials, whereas audio will always be absorbed by the material it hits when moving through space. Audio also needs molecules to travel through space. That is why audio waves cannot move, and therefore cannot be heard, in outer space [24]. Even if they are on a different spectrum from each other, the focus will be on how audio is implemented in Unity and if it is working differently from shaders and ray tracing.

3 Related Work

In order to start working on our contributions, we will discuss previous works that have been done in the fields of radar simulations and sea modelling. Firstly, reviewing studies focused on sea clutter and sea modelling, both real-world observations and simulated, to get an understanding of how measuring sea clutter works. Then, works using the different methods of simulating radar are reviewed in order to get a grasp on what differentiates the approaches from each other. It was found that ray tracing was more commonly used as an approach to simulate radar waves compared to a shader-based or audio approach. Therefore, the related works on ray tracing became more relevant later in the study, while the related works from the shader and audio sections were only used for the pre-study.

3.1 Sea modelling

Andersson [10] did a study for The Swedish defence research agency (swe. "Totalförsvarets forskningsinstitut") FOI regarding the modelling and calculations of radar signatures emitted from a ship in a sea setting. Andersson simulated, with the help of OKTAL [25], radar waves that hit two different targets in two separate tests. One test was emitting signals against a 16x16-metre cube, and another was against a 16x4-metre object that was modelled to look similar to an actual ship's figure. Each test was made up of smaller tests where the objects were surrounded by water in different sea states and also a test with the object floating in free space. Different grazing angles (the angle from the radar pulse relative to the surface) were tested, i.e., the radar wave's trajectory to the object. The results were then observed by looking at the polarisation of each angle and measuring the signal strength in dBm^2/m^2. This report by Andersson [10] is close to what we want to try to achieve as well, but the difference will be in its implementation.

Greger-Hansen [18] and Watts [26] made models of the sea clutter effect on radar with different angles and matched their results with Nathanson [17]. Watts used a K-distributed Gaussian model to simulate coherent sea clutter. Greger-Hansen looked at the NRL model and proposed a new model with a modest number of free parameters that would still match the empirical data set. However, this article mentions that any high-fidelity performance evaluations of target detection, like detecting a moving ship, are not taken into consideration, and Greger-Hansen mentions that more statistical data on sea clutter's temporal and spatial

characteristics will be needed [18] in order to achieve this.

Our aim is to figure out if a Unity 3D setting could measure and generate sea clutter using a radar wave simulation. Greger-Hansen [18] and Watts [26] could help verify our test results in order to determine if simulating sea clutter in Unity could be a viable option for the defence industry to use.

3.2 Ray tracing

Zhengqing et al. [27] did a review on modelling radio propagation using ray tracing. They presented and reviewed previous works in the area, such as important aspects regarding rays and propagation mechanics, different ray tracing algorithms, and methods of accelerating the computations. The conclusion of the review is that ray tracing is an ideal method of simulating radio propagation. Reasons include its similar characteristics to the actual physics of radio propagation and the continuous development and support of hardware acceleration [27].

An important algorithm that is mentioned in the review of Zhengqing et al. [27] is the SBR method of ray tracing. It was invented by Ling et al. [15], who calculated the RCS of a cavity. This method involves three parts: determining ray paths and the rays returning from the cavity using geometrical optics (GO), and then calculating the backscatter and RCS using physical optics (PO).

Egea-Lopez et al. [2] created an EM wave propagation simulator called Opal. They used the ray tracing method SBR to simulate the trajectory of the EM waves. Opal is an open-source project that can act as a standalone program or as a plugin for Unity. Opal is written in C++ using the ray tracing API Nvidia OptiX.

Yilmaz [3] created a radar sensor plugin for game engines capable of imaging radar and RCS simulation. They used a Matlab-based tool called POFACETS to calculate the RCS and model the propagation using the SBR approach to ray tracing with Nvidia's OptiX ray tracing engine.

OKTAL-SE [25] provides a tool called SE-Workbench-RF for simulating radar using physical models. They list several publications on implementing a radar simulation using the SBR approach to ray tracing with GO and PO interactions [4] [5]. Douchin et al. [5] also mention the ray tracing characteristics in combination with their simulated ocean using either the JONSWAP sea spectrum or the SWAN model.

Ulmstedt and Stålberg [6] used Nvidia's Optix ray-tracing engine in their thesis. This work was based on, at the time, a new Nvidia card that had been released that they wanted to use for simulating ray tracing in an underwater setting. In their work, they had been working directly with the graphic software known as CUDA[10] to make an application that they would then test the performance of. Their experiments were done in a 2D setting, and the result that they found was that to get a good simulation of a real-world result, they needed about 500–1000 rays emitted at a target. Their conclusion was that the work could be expanded to a 3D setting, but it would require more rays and therefore take a lot more performance from the GPU, which could be done with modern graphics cards and optimisation.

The work by Ulmstedt and Stålberg [6] shows the potential of using the GPU for radar wave simulation in a 3D setting, which we will explore in our thesis.

[10] Nvidia. *CUDA Toolkit*. Accessed 12.04.2023. URL: `https://developer.nvidia.com/cuda-toolkit`.

3.3 Shaders

There have been several implementations of Radar and LiDAR using a shader-based implementation, [7], [8], [9]. Since shaders and the GPU are already used for rendering images and lighting calculations, a reasonable approach is to mimic existing lighting models but apply them to radar simulations. Peinecke et al. [7] used the Phong lighting model to simulate a millimetre-wave (MMW) radar to produce radar images. They used the graphics library OpenGL, in which they modified the vertex and fragment shaders of the render pipeline.

Wang et al. [8] implemented a shader-based radar and LiDAR for autonomous car testing. They mention that ray tracing for this purpose is infeasible, however, this study was from 2012, and today, in 2023, ray tracing is more commonly available via modern hardware. Their implementation is based on the works of Peinecke et al. [7].

Ciarambino et al. [9] uses Microsoft AirSim for Unreal Engine and implement an imaging radar simulation using custom vertex and fragment shaders. They simulate MMW radar behaviour, calculating the radar return and the RCS and modelling the reflection with a Phong lighting model. Their implementation is also based on the work of Peinecke et al. [7]

3.4 Audio

Beig et al. [28] introduce the different methods of implementing spatial sound in virtual environments. The authors mention several techniques to achieve spatial sound using sound propagation and the basic necessities of localisation, i.e., determining where a sound is coming from in a 3D space. A commonly used method for simulating sound propagation in virtual environments is geometrical acoustics (GA), which uses the same concept of simulating the propagation of light as GO does. This is implemented in tools such as Steam Audio and FMOD by different ways of shooting rays, such as ray tracing or with multiple/singular raycasts.

Savioja et al. [29] review different methods of modelling room acoustics. The authors name two main methods: a numerical wave-based and a GA wave-based. GA is similar to GO found in computer graphics, and it is implemented using ray tracing. Since sound waves are an order of magnitude longer (lower frequency) than EM waves, this method is only accurate for sound waves with high frequencies. The reason is that some objects in a scene might be smaller than the length of a sound wave and therefore go unnoticed. If an implementation of audio were to be used, it is likely that the implementation would involve ray tracing.

Wolf et al. [30] used the Native Audio SDK from Unity to create a plugin with the aim of improving localisation accuracy. The plugin was created using C/C++ and showed better results than the default audio implementation from the horizontal plane but not from the elevation angle. Their method of implementation could be relevant if the audio approach was used for the implementation in our study.

4 Method

This chapter will present the different methods used in the study. Starting with how the pre-study was conducted in order to examine the different approaches to simulating radar. The specific details regarding the setup of the different tests and the scene are explained, following a brief section about the equipment used for the study.

4.1 Pre-study

To evaluate what approach should be used, a pre-study was conducted. By studying related works, documentation and details regarding the implementation, along with the expected results of each approach, are presented. Following this, a discussion provides context behind each method and the reason why one method was chosen. The parameters that were considered are the following:

- How well-suited the method is for the purpose of simulating radar wave propagation.
- How accurate the results are expected to be based on previous studies.
- Expected time of the implementation, taking both the author's experience and the complexity of the implementation into account.
- The method should use the GPU for its calculations.

How well a method is suited for the purpose of this study is based on the similarities of this work to related works and their methods of implementation. There may be some features of a certain approach that are not relevant to our purpose at all, and this should be taken into account.

It would be preferred that the chosen approach has been used in previous studies with good results, meaning results that correlate to real-world measurements. The outcomes of different approaches may differ in realism or in other ways that should be considered.

Estimating the time for the implementation took into account the limited experience the authors had in graphics programming. A preferable method was considered one where most of the code was already in place and minor adjustments had to be made.

Performing calculations on the GPU can be very beneficial in terms of performance compared to the central processing unit (CPU). While no requirements on performance were specified, it is good for practical reasons to have a reasonable load and test time for testing an implementation.

4.2 Testing equipment

The equipment used in the study was a laptop using Windows 10 with an i9 processor, 64GB of RAM, and a Quadro RTX A5500 graphics card with 16GB of VRAM.

4.3 Scene

Depending on the test, the scenes contained a reflective object, e.g., a ship or an ocean surface. It was needed to measure the effect water can have on a radar signal at a certain point in time, and thus we decided to use static meshes. We did not have time to build an ocean surface generator, so an existing implementation on Github[11] was used. To get a realistic representation of the sea surface, the author of this implementation took inspiration from the paper by Tessendorf [23] using the Phillips spectrum. By using this implementation, we could pause the scene, save a mesh of the ocean, and copy it to our project. This allowed us to create ten meshes, which we used in our tests described in Section 4.4. We used two different sea states; see Figure 5.3. Matching the table by Barton [1], the sea states are roughly 5 and 6, with waves of about 2.5 and 6 metres, respectively.

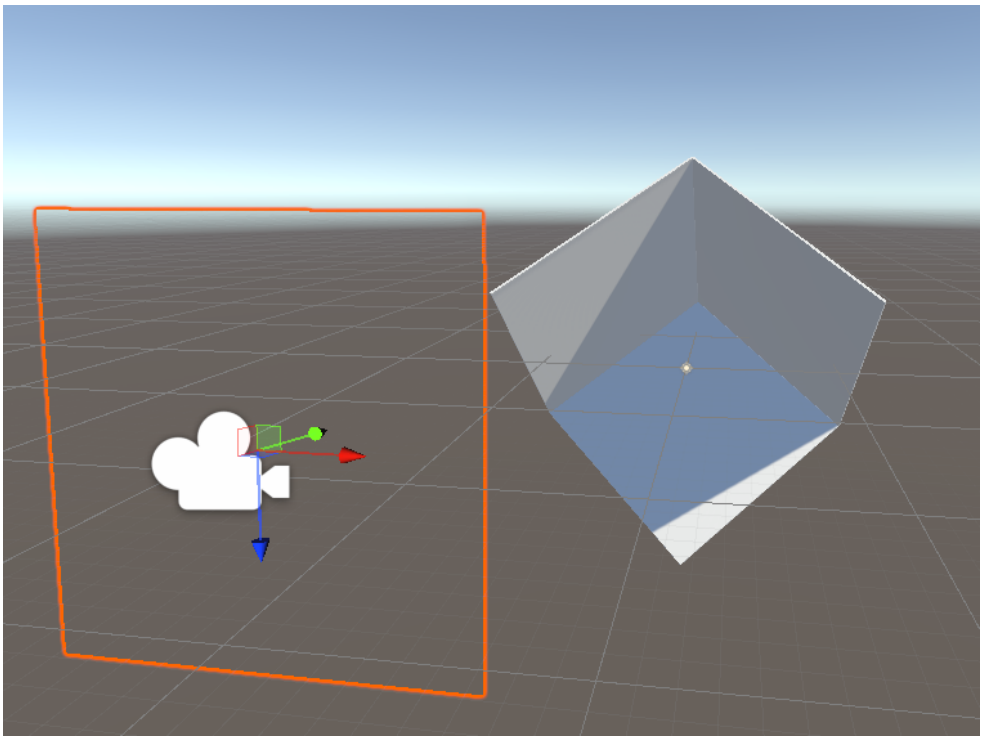


Figure 4.1: Scene with the corner reflector, the camera, and the receiver plane with an orange outline.

[11] Scrawk. *Phillips-Ocean*. Accessed 23.05.2023. URL: https://github.com/Scrawk/Phillips-Ocean.

4.4 Tests

Like the FOI rapport by Andersson [10], the implementation created was tested with different objects in different environments. It was decided to use a similar approach to Andersson so that the results of a similar simulation could be compared. It should be noted that the study of Andersson was more focused on the effect the ocean had on an object in the sea and not only on sea clutter itself. Therefore, there are some differences. We have, instead of using a cube object, chosen a corner reflector and a more detailed ship than Andersson used.

Tests were made using the same distance of 50 metres from the camera to the object and with a grazing angle of 10 degrees. As a result of the pre-study, it was decided to use ray tracing. Having the resolution set to 1920x1080 meant that 2.073.600 rays were traced, with a maximum of 4 reflections. The field of view, determining how wide the camera angle is, was set to 60 degrees.

4.4.1 Test 1 - Corner reflector

The first test was to verify the functionality of the implementation. After discussing this with our supervisor at Saab, a corner reflector object was deemed an appropriate first test. We therefore, created a corner reflector that will be the reflecting object. A corner reflector will create a strong signal with a large RCS, which makes it a good first test for our radar implementation. This object was then rotated 1 degree at a time, up to 45 degrees. Figure 4.1 shows how our object looks in the game engine.

4.4.2 Test 2 - Two different sea states

With test 1 being successful in the sense of receiving reflected rays with different amplitudes measured, we moved on to testing emitting rays onto the simulated ocean surfaces. In test 2, which was divided into two parts, rays were emitted against the sea surface with SS 5 and SS 6. 10 different meshes for each ocean were used. The purpose of this test was to measure sea clutter.

4.4.3 Test 3 - Ship in free space

In this test, we had a ship model, which can be seen in Figure 5.4. We used Blender[12] to modulate and reduce its total triangles for the ship's mesh. This needed to be done to be able to smoothly run our implementation in Unity. By measuring the ship in free space at every 15 degrees for a full 360-degree measurement, it was a similar method as Andersson [10] used. The reason was to compare the results to those of Andersson to further verify our implementation and also to later use the measured data to compare with the ship on the sea surface. This would then show the effect of sea clutter.

4.4.4 Test 4 - Ship in two different sea states

We created a scene in Unity consisting of both the ocean surfaces with the two different sea states and the ship. Like test 2, this test was divided into two parts since we have two different sea states. The difference was that one of the time stamps for each ocean was used and not all ten. For the surface with the larger waves, i.e., SS 6, it was decided to use the first time stamp since it returned some sea clutter. The ship was rotated every 15 degrees as in free space. The aim of this test was to show the effect of sea clutter and also to verify that our implementation is capable of drawing similar conclusions and results as those concluded by Andersson.

[12] Blender Foundation. *Blender 3.5.* Accessed 02.06.2023. URL: `https://www.blender.org/`.

5 Implementation

This chapter will first present the results of the pre-study, which will serve as the basis for our implementation. Our implementation using ray tracing is then presented, along with implementation details for the scene. This aims to answer the first part of our research question, which is about how a radar implementation simulating sea clutter could be designed in Unity. Lastly, the method used for evaluating each ray's signal is described.

5.1 Pre-study

The following pre-study investigates each proposed method to further understand details regarding its implementation and expected results. This is based on earlier studies and official Unity documentation. A method is then chosen to be implemented after a discussion explaining the reasoning behind the choice. Because of limited time, using as many built-in features of the game engine as possible has been preferred.

5.1.1 Ray tracing

Based on previous related works [2], [3], [4], [5], [10], [27], ray tracing is deemed to be a well-suited method for simulating radar waves. Each ray is traced through the geometry of a scene, resembling the propagation of actual radar waves. Several works use ray tracing for this purpose, as previously mentioned in Section 3.2. [2], [3], [10]. The results of using this method are considered to be relatively good since a company such as OKTAL, whose tools are used by many defence companies around the world, also uses ray tracing for their radar simulations [4], [5]. Furthermore, the review of modelling radio propagation by Zhengqing et al. [27] concluded that ray tracing is an ideal method for this purpose, not only because of the similar properties of rays to radar waves but also because of its continuous development and hardware support.

Ray tracing can be implemented in several ways. A potential implementation is to use the Unity "High Definition Rendering Pipeline" (HDRP) in some way. HDRP differs from the normal rendering pipeline in that it contains more advanced, physics-based lighting

techniques[13]. After further research, it was deemed too difficult and time-consuming to customise the built-in features to our needs. Another way of utilising the HDRP is to use the ray tracing API provided by Unity. Using this API, it is possible to create a custom ray tracer using a ray tracing shader. However, since this API was "out of experimental" in 2023[14], there is not a lot of documentation and examples available. Other approaches investigated include creating your own implementation of ray tracing using a compute shader or using the built-in raycasts available in Unity. Using raycasts for this purpose is not ideal, however, as this will run on the CPU and not take advantage of the much better parallel computational power of the GPU. Creating a ray tracing implementation in a compute shader was a well-documented approach, as there were several examples of this online.

There are several works where the authors have created a plugin for Unity, implementing ray tracing using the ray tracing engine Nvidia Optix [2], [3], [6]. Because of limited time, this option is too time-consuming since it not only involves creating a ray tracer but also building a plugin that has to work within Unity.

5.1.2 Shaders

Simulating light with other methods than ray tracing involves implementing different lighting models in the rendering pipeline, such as in the vertex and fragment shader, like Peinecke et al. [7], Wang et al. [8], and Ciarambino et al. [9] did. The work by Peinecke et al. [7] contains a detailed description of the implementation, which would provide a good base for us to create an implementation in a timely manner. These methods use a numerical approach to simulate the propagation of rays in a scene, such as using the Phong lighting model. Similarly to ray tracing, the implementation will utilise the GPU for better parallelization of the calculations. Since ray tracing provides better realistic lighting and is also more computationally intensive, not using ray tracing might provide less accurate results but better performance in terms of computation time.

5.1.3 Audio

The Unity audio system is built around an audio source and a listener. The built-in Unity audio system is not capable of calculating echoes based on the geometry of a scene or an object, which would be beneficial for radar simulations. Instead, different filters, such as an echo filter or a reverb filter, are used to simulate different audio behaviours. The distance between the audio source and the listener is still used, however, meaning that if the objects are moving, the Doppler effect, which is used in some radar implementations, can be simulated[15].

The existing sound system in Unity is not very advanced, but they provide developers with a software development kit (SDK) to create their own audio implementations and extend its functionality. There are also other plugins that can be used inside Unity, such as Steam Audio and FMOD, as mentioned by Beig et al. [28]. However, these use ray tracing and sometimes simple raycasts to determine the wave propagation, as mentioned in the review by Savioja et al. [29].

[13]Unity. *Render pipelines*. Accessed 21.04.2023. URL: https://docs.unity3d.com/Manual/render-pipelines.html.

[14]Unity. *Raytracing API Out of Experimental in 2023.1*. Accessed 21.04.2023. URL: https://forum.unity.com/threads/raytracing-api-out-of-experimental-in-2023-1.1350566/.

[15]Unity. *Audio overview*. Accessed 17.04.2023. URL: https://docs.unity3d.com/Manual/AudioOverview.html.

5.1.4 Discussion

After investigating each option, the first decision was to not use the audio approach. The existing audio system in Unity was not very advanced in terms of simulating wave propagation with objects in a scene, and existing implementations of such a simulation used either simple raycasts or ray tracing. Since using raycasts will use the CPU instead of the GPU, only the ray tracing option is left, and if so, we might as well use another approach that does not involve the Audio SDK. A final reason was that audio waves operate on a different spectrum than EM waves, which further motivated the conclusion that the other options would be better.

The non-ray tracing methods of light simulation are estimated to be less accurate. While this method can be sufficient, it is not as good as emitting rays and tracing their actual path in a scene. However, an implementation could be estimated to be completed in a reasonable time considering the detailed description available in papers by Peinecke et al. [7], for example. Furthermore, this method is not as computationally expensive and may provide better performance in terms of computation time, etc., compared to ray tracing. However, as there was no requirement for our simulation to run in either real-time or, if it did, at a high frame rate, there is more reason to use ray tracing than not.

The chosen approach was ultimately to use ray tracing. After considering the alternative ways of implementation, it was decided to do this using a compute shader. Data calculated in the compute shader can be transferred to the CPU side using existing Unity methods such as "ComputeBuffer", and interactions with objects in the scene should be more straightforward as the compute shader will run within Unity. This will save time compared to using Nvidia Optix, which would need a plugin to be created along with the ray tracer, complicating the communication between tracing rays and objects defined in the Unity scene. The other implementation alternative, which was using the recently "out of experimental" ray tracing API, could be considered the more optimal approach. However, as the documentation for this was somewhat lacking, combined with the fact that the authors had limited experience in programming shaders, it was decided to use the more documented alternative of building a ray tracer from scratch using a compute shader.

5.2 Implementation

The following section will first present how the chosen approach of using ray tracing was implemented using a computer shader in Unity. Following this is how the different scenes were implemented, such as how the ocean surfaces with different sea states were generated and scene-specific details.

5.2.1 Ray tracing

The resulting implementation is based upon an article by David Kuri[16], providing a guide to how a simple ray tracer can be created from scratch in Unity. It provided a base to build upon and modify to meet our needs. It works by sending out rays from the scene's camera, tracing each ray through the scene, and recording any hits. For simplicity, rays that hit objects are specularly reflected, i.e., the rays will be perfectly reflected. Rays that do not hit anything for a certain distance [17] are ignored. Rays are also only allowed to reflect a finite amount before being ignored. We have set a limit of four reflections in our testing environment. Information about each ray is stored in a struct called "RayData". This struct was modified to contain

[16] D. Kuri. *GPU Ray Tracing in Unity – Part 1*. Accessed 21.04.2023. URL: http://three-eyed-games.com/2018/05/03/gpu-ray-tracing-in-unity-part-1/.

[17] The distance is defined as #INF in the shader.

more variables needed to calculate the signal strength, such as distance and phase, as further explained in Chapter 6 and shown in Figure 5.1. By adding some functionality, such as being able to calculate the total distance travelled and the signal strength, our implementation resulted in a ray tracer that is able to calculate and send information about each ray. This is then used to calculate the total signal strength, the number of rays returned, and visualise the path of each ray.

```
struct RayData
{
    float3 origin;
    float3 direction;
    float energy;
    float distance;
    float inPhase;
    float outPhase;
    float phi;
};

RWStructuredBuffer<RayData> _Rays;
```

Figure 5.1: How the struct containing the data collected for each ray is declared in the HLSL file, i.e., the compute shader. The "RWStructuredBuffer" that will transfer this data to the CPU is also shown here.

The ray tracing implementation involves a C# script representing the CPU side and an HLSL file, i.e., the compute shader file, representing the GPU side. The C# script will create and initialise "ComputeBuffers" and call upon the compute shader. In Unity, "ComputeBuffers" are memory buffers, i.e., arrays used to transfer data between the CPU and the GPU. Structs that will contain information about each ray and object in the scene are defined on both sides. When running the program, the CPU first gathers information about objects that the rays will intersect with. Information that only needs to be sent to the GPU uses a "StructuredBuffer", in our case, this includes all objects from the scene that contain some mesh that the rays will interact with. After the scene data has been sent to the GPU, rays will be dispatched and calculated against the scene data. When this is done, the results are sent back to the CPU using an "RWStructuredBuffer", i.e., a "ComputeBuffer" that can also send data back to the CPU. This is needed in order to be able to analyse and visualise the data. The information we gathered is then summarised. Each ray's coordinates at different points of reflection are also collected and used to draw lines, visualising each individual ray's path. Figure 5.2 illustrates how the implementation works after the C# script has gathered information about the scene, initialised the "ComputeBuffers", and called the compute shader.

5.2.2 Scene

Our scene contains a camera from which all rays will be dispatched, a plane object placed directly behind the camera that collects returning rays, and lastly, some objects that the rays will intersect with. The plane object is a "Plane", 3 by 3 units[18], and scaled to the local world matrix in our scene.

The water mesh was generated using the previously mentioned implementation of the Phillips spectrum. Since our ray tracer only allows static meshes, the mesh generated was saved at 10 different times. A single tile of these meshes is 64 by 64 metres. Our ray tracer is limited in the amount of triangles, or polygons, it is able to trace. Above a certain threshold

[18] 1 unity units $\approx$ 1 metre

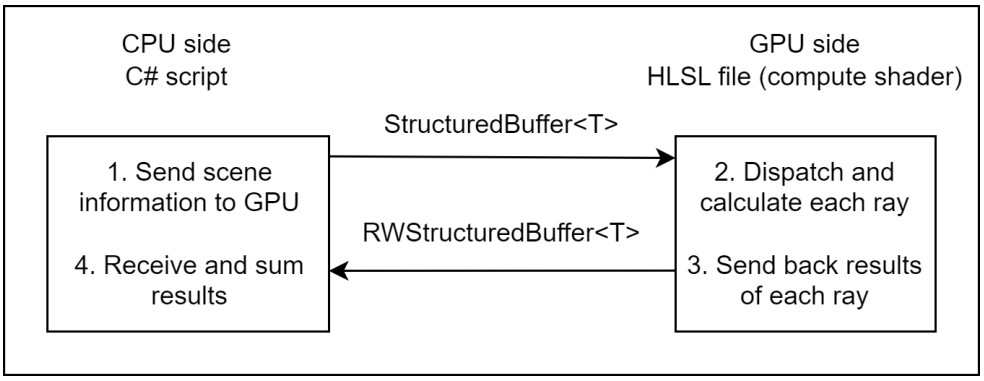


Figure 5.2: A diagram illustrating how the implementation works.

of about 100k triangles, a simulation will either take up to 30 seconds to finish, freeze, or crash. Therefore, only two meshes are used for each water scene, creating a surface that is 64m wide and 128m long. The scene with the two water meshes and the ship totals 79k triangles, where the two water meshes combine for a total of 35k triangles. It was found that any ocean with a sea state lower than 5 did not return any sea clutter, but as sea state 6 did, these two were thus chosen for the tests that we conducted.

The corner reflector was made by scaling 3 cube objects in the shape of a corner reflector, and the ship is a free model found on cgtrader.com[19].

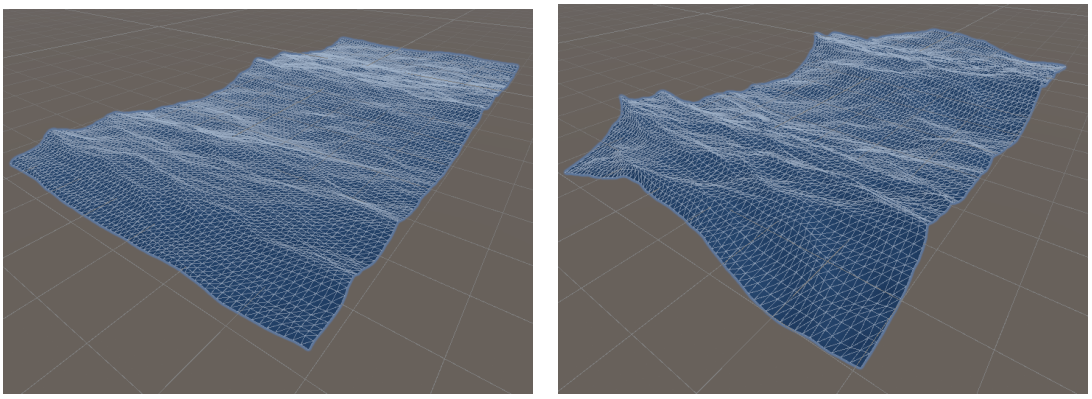


Figure 5.3: Ocean surfaces with SS 5 (left) and SS 6 (right).

5.3 Evaluation method

In a "ComputeBuffer", we collect the data structures we created on both the C# script side and on the compute shader side. On the GPU side, we use the structure to calculate distance, direction, and the coordinates of the last hit, in- and out-phase, for each ray that returns and hits our receiver (the plane object). With the data collected from the "ComputeBuffer", we then summarised the total signal received in the C# script. This signal received is measured by its amplitude on a logarithmic unit scale, i.e., decibels (dB). The formulas 2.2 and 5.1 to 5.5 show the evaluation of the total signal received. The RCS is represented by the number of returned rays.

[19] cgtrader. *Destroyer ship Low-poly 3D*. Accessed 30.05.2023. URL: `https://www.cgtrader.com/3d-models/aircraft/military-aircraft/saab-jas-39-gripen-4db25d41-50c0-451d-9a66-7773b123e002`.

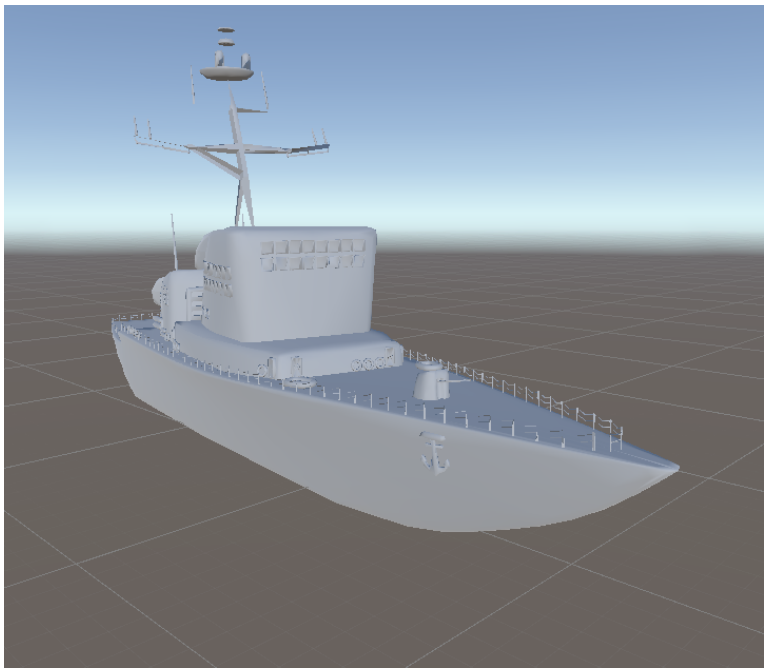


Figure 5.4: The ship model used.

In our implementation, we calculate the amplitude (A) as the total received signal. The amplitude is derived from the following formula, which is also mentioned in section 2.4.2:

$$\textit{Total received signal } s_r = A * \sin(\omega_0 t + \phi_r) \Rightarrow A = \sqrt{I_{tot}^2 + O_{tot}^2} \qquad \text{(2.2 revisited)}$$

Firstly, we calculate the phase (ϕ_i) of each ray, indicating the ray's position on its sine curve (from 0 to 2π). This will be used to determine the in- and out-phases needed to calculate the amplitude. We have to remove all of the full-length wave lengths since we are only interested in the phase of the wave hitting our radar receiver. R_i is the total distance the ray has travelled, and λ is the radar wavelength obtained in 5.5.

$$\phi_i = \frac{2\pi \bmod (R_i, \lambda)}{\lambda} \tag{5.1}$$

The in-phase and out-phase are calculated as follows:

$$I_{in} = \cos(\phi_i) \tag{5.2}$$

$$O_{in} = \sin(\phi_i) \tag{5.3}$$

The amplitude A is then finally calculated using the following formula, where I_{tot} is the sum of all the rays in phase and O_{tot} is the sum of all the rays out phase:

$$A = \sqrt{I_{tot}^2 + O_{tot}^2} \tag{5.4}$$

The radar wavelength is calculated by taking the speed of light constant and dividing it by the frequency of the radar antenna used. In our tests, we have decided to use a frequency of 10 GHz, which belongs in the commonly referred-to X-band frequency and is often used in military radar equipment when detecting radar signatures of objects in the ocean.

$$\lambda = \frac{c}{f} = \frac{299792458\ m/s}{10^9\ Hz} \approx 3\ cm \tag{5.5}$$

The lobe width, or the width of the radar emitting a pulse, is normally a few degrees horizontally, and the height is another few degrees more. In our implementation, it was decided to use the resolution of our game engine and a camera field of view of 60 degrees to act as the radar lobe. A resolution of 1920x1080 will produce 2.073.600 rays emitted from the camera's perspective.

6 Results

This chapter will present the measured results from the tests described in our method. Firstly, showing the results from the corner reflector, then the ocean surfaces. Lastly, the results of the ship in free space and the ship together with the two chosen ocean surfaces are presented.

6.1 Test 1 - Corner reflector

In Figure 4.1, we have our scene in Unity consisting of a camera object for emitting the rays, a corner reflector object which we want to detect or bounce our rays against, and a plane object for collecting the returning rays. The returning rays are the ones that we want to calculate the amplitude of with the help of equation 5.4. The results show the highest amplitude at degree 0 when the object is facing directly towards the camera, decreasing heavily until degree 29, and then decreasing again until degree 42, which returns nothing. Because the corner reflector is symmetrical and the side and back of it do not return any rays, no further measurements were needed.

A first intuition would be that many rays returned would equal a high amplitude. Looking closely at the specific data, this is shown to be false. For example, at degree 0, there is an amplitude of about 46.6 dB with 676 rays returned, compared to degree 23, where the amplitude is only 1.4 dB but has 1464 rays returned. To explain this, we examined the distribution of each ray's phase in two histograms, shown in Figures 6.1 and 6.2. What can be read from the histograms is that degree 23 has an equal distribution of the phases, causing the waves to cancel out each other and the resulting amplitude to be lowered. Meanwhile, degree 0 has a more uneven distribution, resulting in a higher amplitude.

While these results show that the phases can cancel out each other, they do not accurately portray radar waves reflecting off a corner reflector. In reality, the returned waves should have the same phase, i.e., the histograms should only consist of a single bin. By increasing the distance to a more realistic distance of 10km and lowering the field of view to 0.3 degrees, it was observed that all of the returned rays had the same distance travelled. This implies that all the returned rays also had the same phase, which is a realistic behaviour.

Object angle (deg)	Amplitude (dB)	Rays received
0	46.66582	676
23	1.418771	1464

Table 6.1: Measured data of the corner reflector at the angle degrees of 0 and 23, in relation to the camera. The complete data is in Table A.2 of Appendix A.

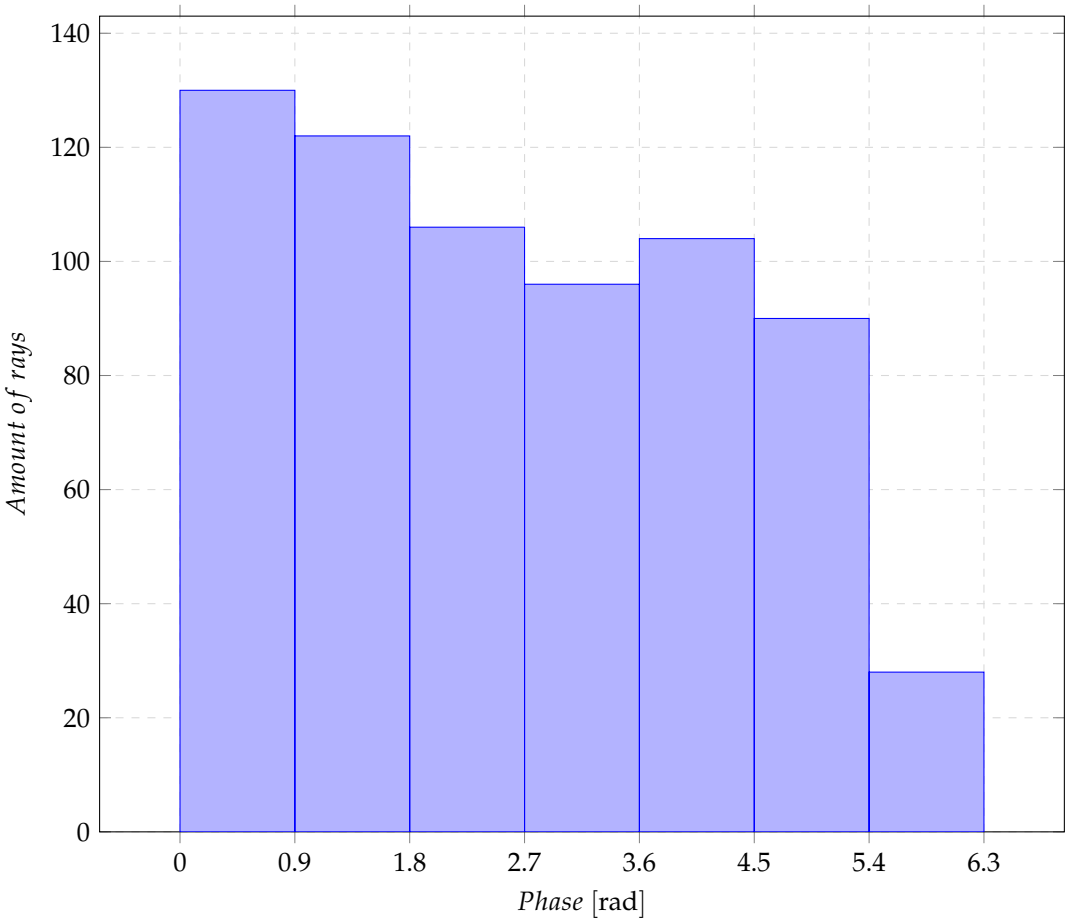


Figure 6.1: The number of rays and their phase hitting our receiver when the angle of the object is turned 0 degrees in relation to the camera.

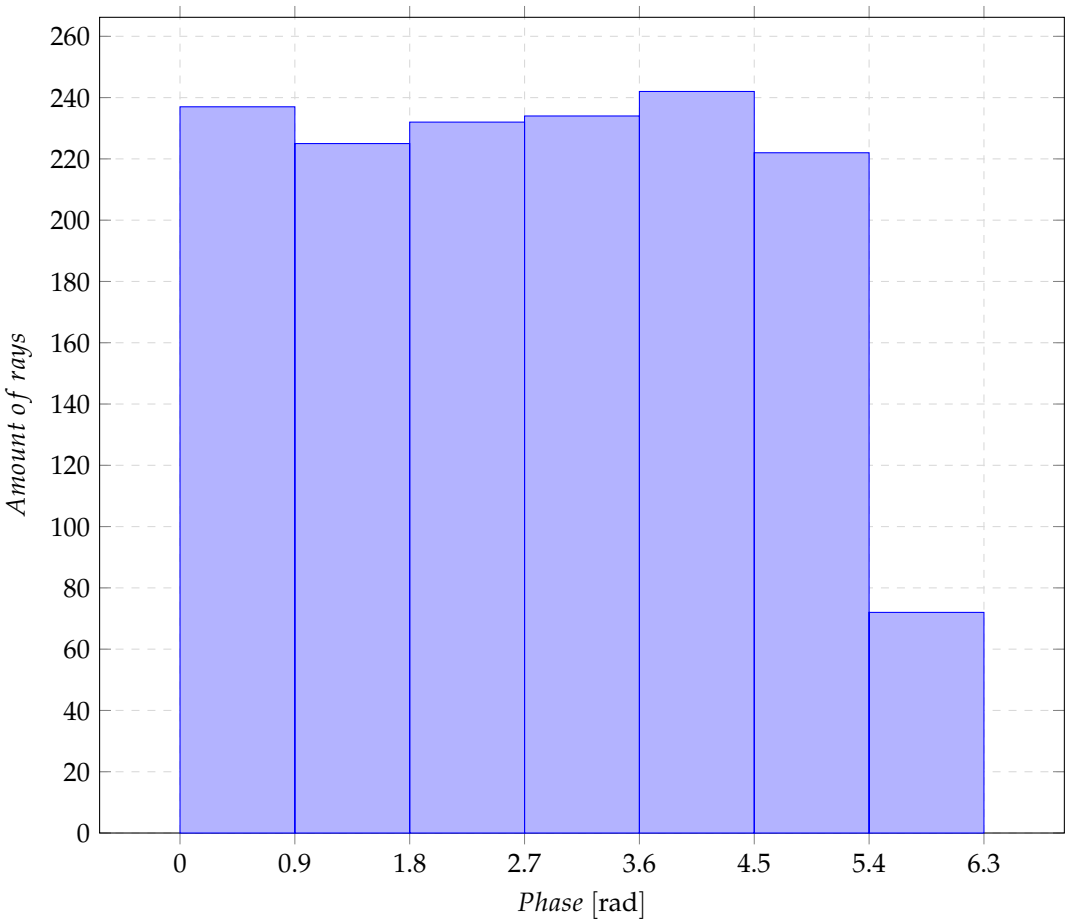


Figure 6.2: The number of rays and their phase hitting our receiver when the angle of the object is turned 23 degrees in relation to the camera.

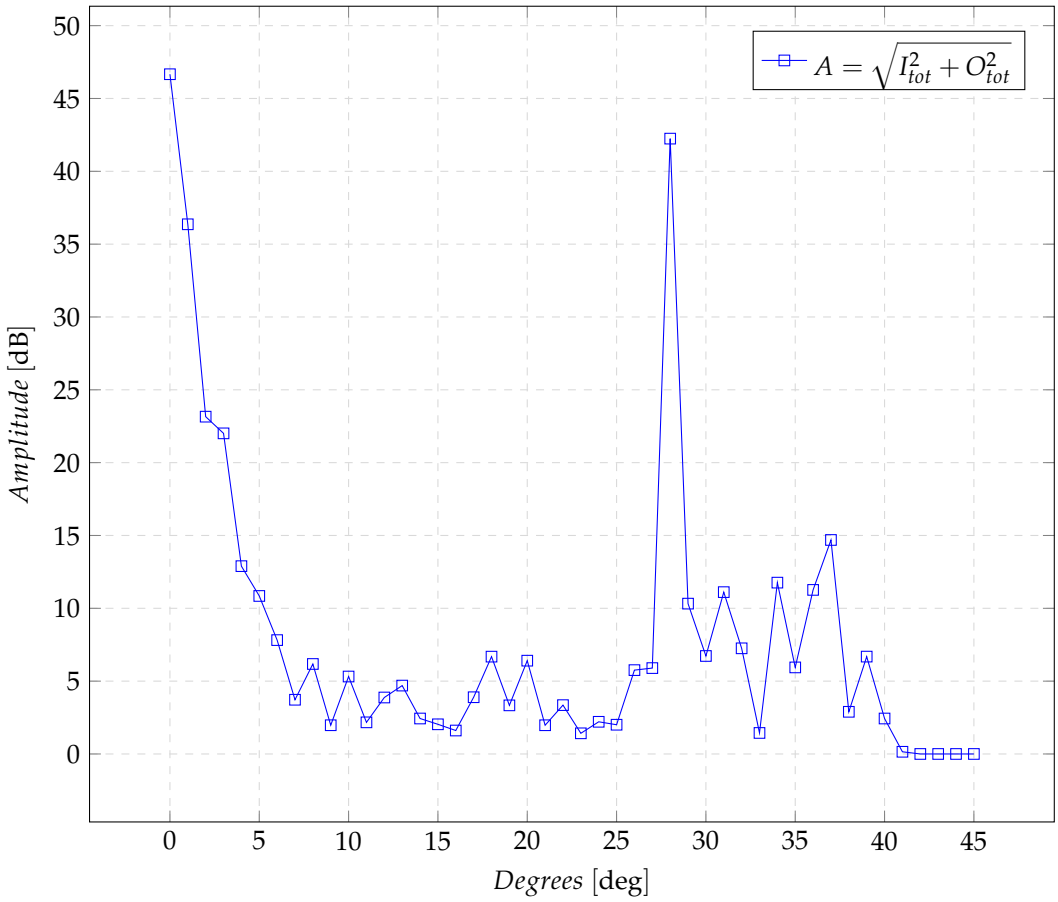


Figure 6.3: The resulting amplitude when rotating the corner reflector, 1 degree at a time, and emitting 2.073.600 rays each time from the camera.

6.2 Test 2 - Two different sea states

The second test conducted used water meshes from the implementation mentioned earlier in 4.3. The test with SS 5 did not produce any results, meaning that no rays were bouncing back to our receiver in the scene. So no matter the different time stamps we tried, there were no readings that were made with SS 5.

In Figure 6.4, we have an ocean with SS 6, and it shows that we received rays back to our receiver at time stamps 1, 4, 6, and 9. For time stamps 1 and 6, the rays were returned from a close range on the closest wave to the camera, thus having relatively large amplitudes. The time stamps 4 and 9 returned a small number of rays and had a lower amplitude after reflecting on the backmost wave. The distance travelled is greater for each ray, which in this case explains the lower amplitude gained in these time stamps.

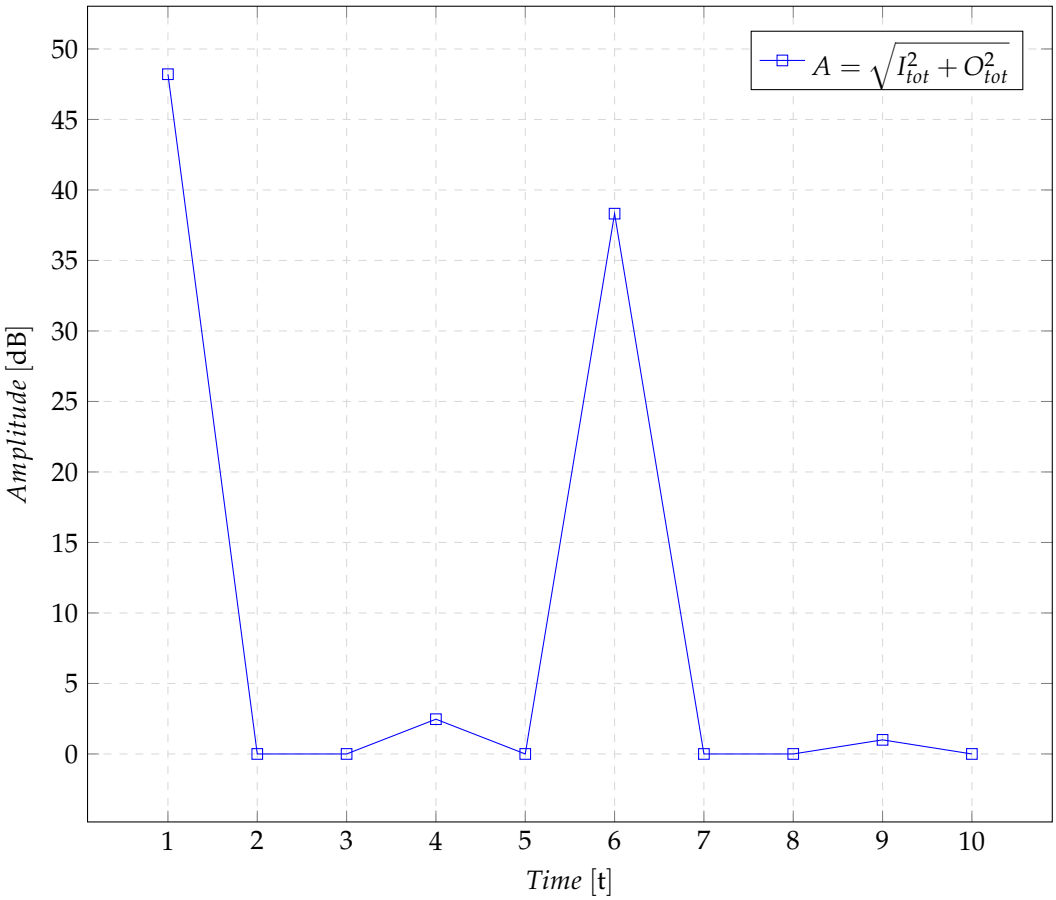


Figure 6.4: The resulting amplitude of rays hitting the ocean with SS 6 in 10 different time stamps. The wave formations are thus in different positions for each time stamp.

6.3 Test 3 - Ship in free space

The ship model was measured every 15 degrees from 0 to 360 degrees. The amplitude peaks at 0 and 180 degrees. This is when the ship is facing directly towards the camera, creating two large flat areas for the rays to reflect. The more sideways the ship is rotated, the more rays are hit. At angle 0 there are about 123k rays hit compared to at angle 90 which has about 330k rays hit. As the ship rotates to the side, the area hit by the rays increases, but it is mostly the hull, which is not a flat surface or at an angle that would return many rays by itself.

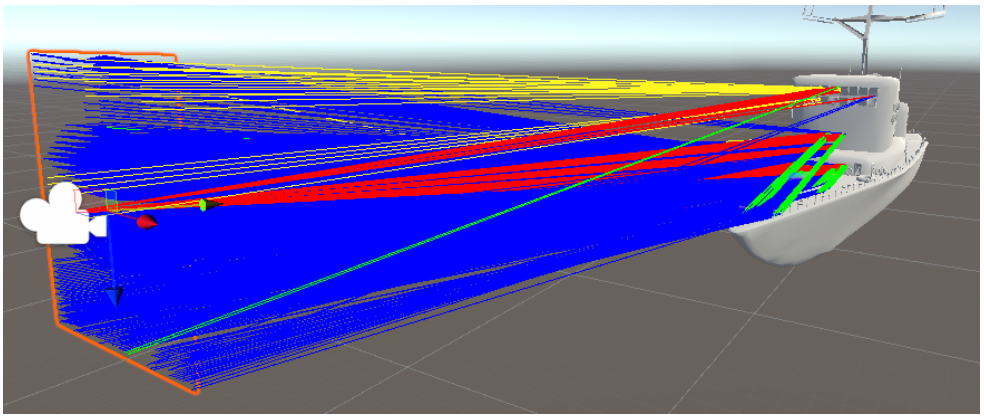


Figure 6.5: The resulting propagation of rays that returned to our receiver. The order of the rays propagation is first red, then green, blue, and yellow.

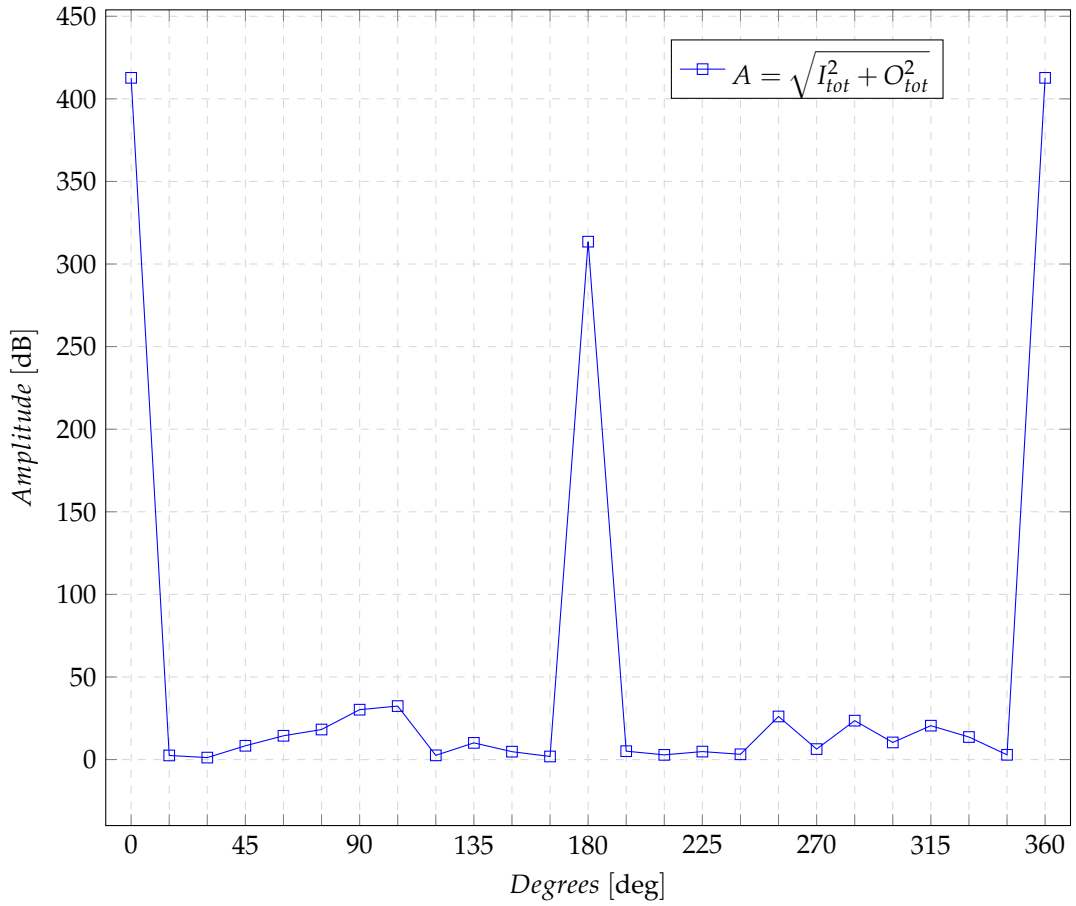


Figure 6.6: The resulting amplitude of the rays hitting a ship rotating 15 degrees at a time, in free space.

6.4 Test 4 - Ship in two different sea states

When measuring the ship in the ocean, an increase in amplitude and rays returned was found compared to the previous test. The surface of the ocean enables more rays to be returned. For example, some rays that previously would not return after interacting with the hull may now return thanks to the ocean surface, see Figure 6.7.

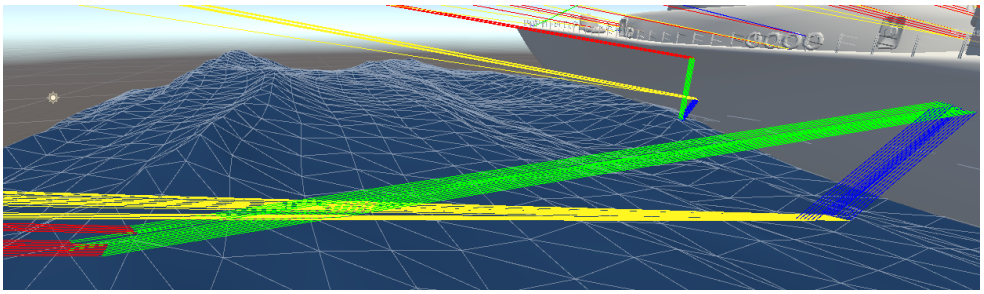


Figure 6.7: Rays reflecting off the hull back to the receiver. Rays are originally red, then after the first reflection they turn green, then blue, and lastly yellow.

For both sea states, the angle that returned the most rays was still degree 0, with almost the same amount of rays received compared to without the ocean surface. SS 5 showed a relatively large increase in rays received in certain rotations, such as at 300 degrees, but no increase at some rotations, such as at 0 degrees. The total amplitude increased somewhat for all degrees. For SS 6, using time stamp 1, there was a substantial increase in the amount of rays returned. As previously measured 6.4, this surface already returns some rays, so an increase of the previously measured rays returned was expected.

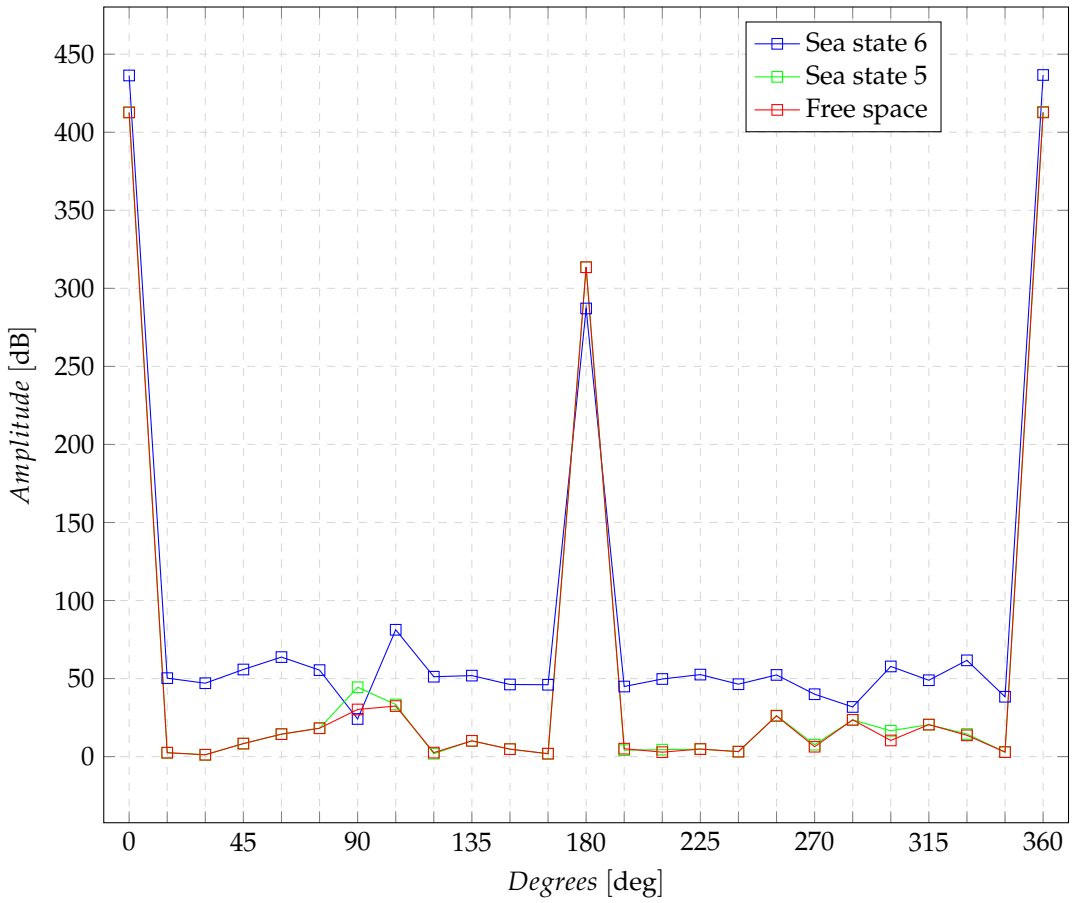


Figure 6.8: The resulting amplitude of a ship in free space (red), in SS 5 (green), and in SS 6 (blue).

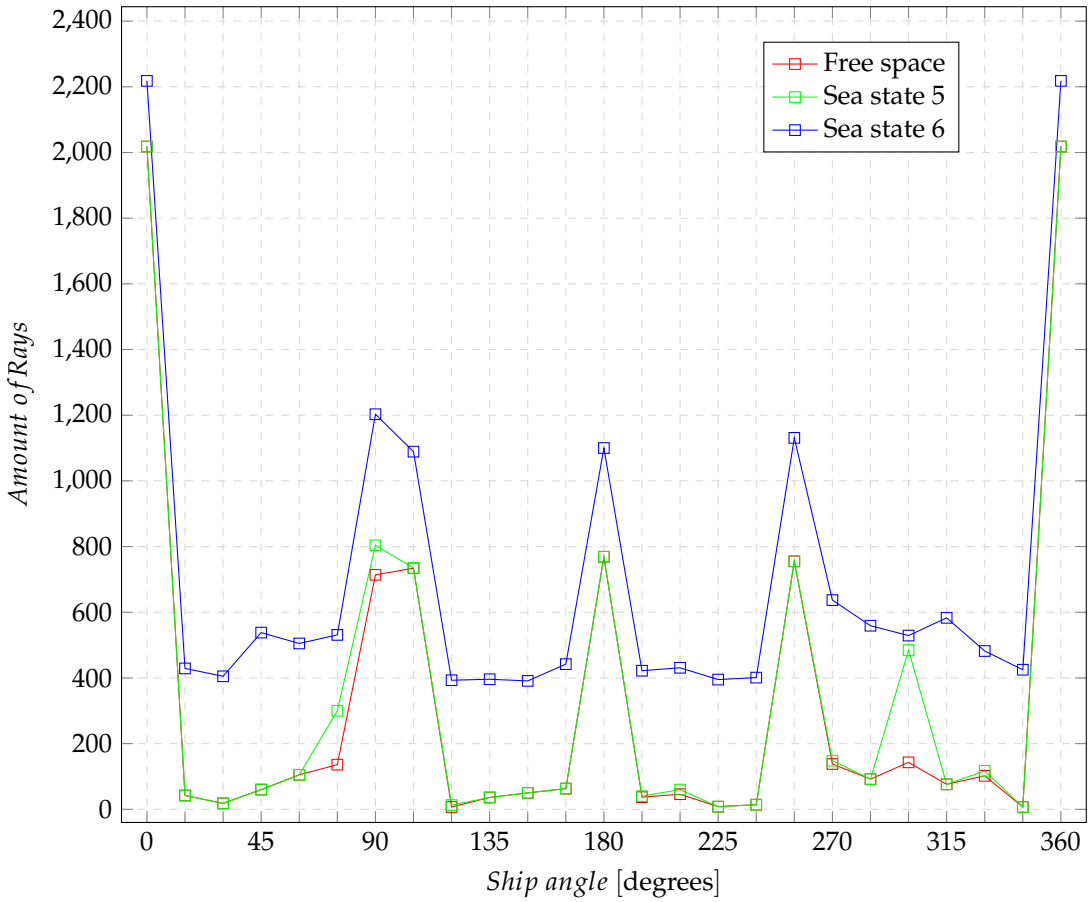


Figure 6.9: Amount of rays returning to our receiver from the ship in free space (red), the ship in ocean with SS 5 (green), and in SS 6 (blue).

7 Discussion

This chapter will discuss the results based on our research question, divided into two sections. Firstly, how accurate the measured results are, and secondly, a discussion of the limitations of our implementation.

7.1 Results

This section will discuss each of the test results measured from the implementation and compare the results from other studies to estimate how accurate the results are.

7.1.1 Test 1 - Corner reflector

The test of the corner reflector was the first test for validation of our ray tracing implementation. By both looking at the propagation of the rays and the resulting data, it showed the expected results. An early assumption was made that more rays returned would equate to a higher amplitude, but this was proven to be false. An example of this was at degree 0, which returned a higher amplitude but a lower number of rays compared to degree 23. To explain this, a histogram showing the distribution of all the ray's different phases was created. An even distribution would mean that there are many rays that will cancel each other out, causing the total amplitude to be lowered. An uneven distribution instead means that there are several rays that do not cancel out each other, causing the total amplitude to be higher. This behaviour is important to highlight since it is also applicable to the other tests.

After the tests were analysed further, the phase measured in the histograms did not correlate to a realistic scenario of a radar reflection from a corner reflector. The geometrical properties of a corner reflector will cause all the incoming waves to travel the same distance and thus return with the same phase. However, as noted, this was not the case in our implementation. The reason is that radar waves are, in reality, parallel when entering the corner reflector. Because of the short distance and the large area of the reflector, the rays that entered were not parallel, causing some rays to travel farther than others. To verify this, a minor test was done, setting a realistic distance to the corner reflector of 10km and a field of view of 0.3 degrees instead of 60. Results showed that all rays returned had the same distance and, thus, as seen in equation 5.1, the same phase. Figure 7.1 shows the difference in how rays enter the

corner reflector depending on distance and field of view. The reason we did not include this in all of our tests was because it was a late discovery and we did not have time to remake all the tests.

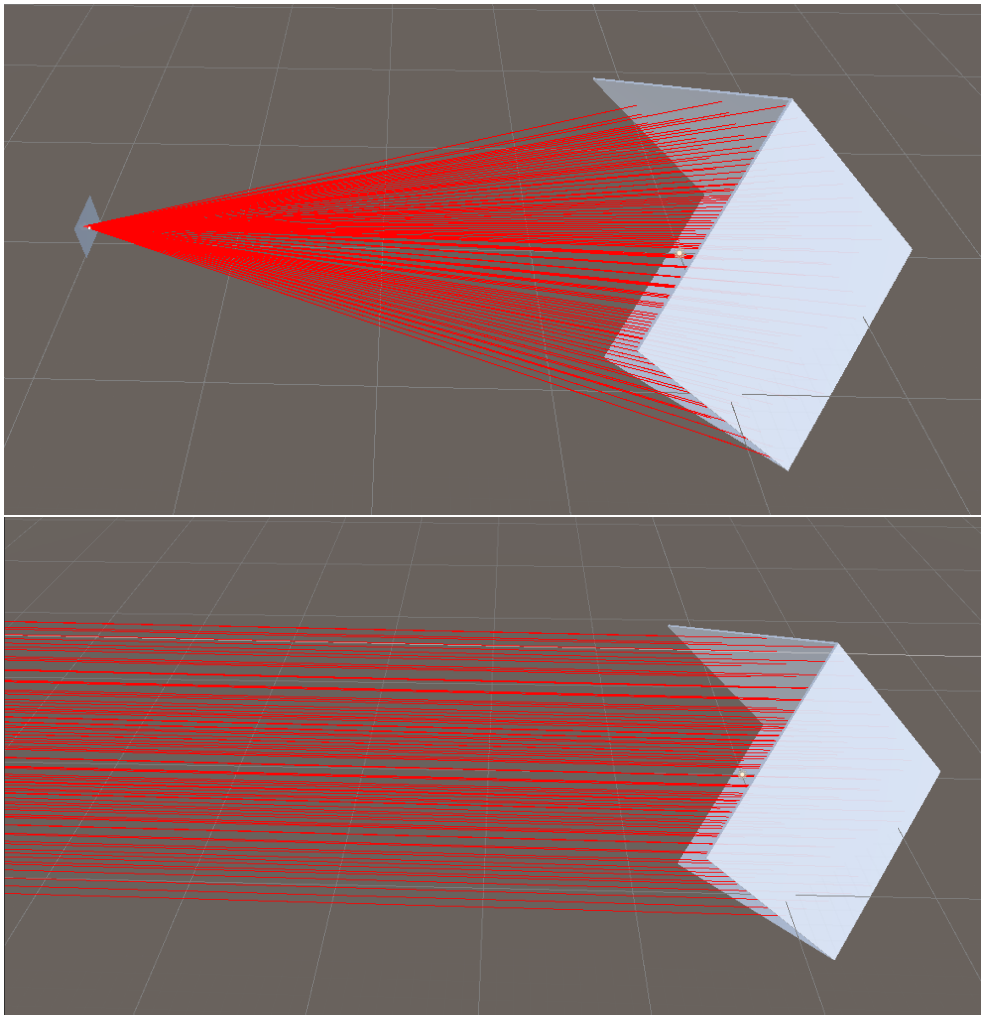


Figure 7.1: The top image shows the rays entering the corner reflector from a short distance of 50m with a field of view of 60. The bottom image shows the rays from a distance of 10km with a field of view of 0.3 degrees. The longer distance and smaller field of view causes the rays to be parallel.

7.1.2 Test 2 - Ocean with sea states 5 and 6

Using the ocean mesh for SS 5, we did not get any returned signals with our implementation. Because the measured mesh did not have very large waves, it did not return any rays for any of the 10 time stamps tried. However, this is not realistic, as there would always be some amount of signal returned in the real world. As seen, for example, in the work by Greger-Hansen [18, Fig. 18, 19], where for the same frequency of 10 GHz as used by our implementation, there are some returns measured for every sea state. Because the implementation assumes all materials to be perfect reflectors, only specular reflections occur. In the real world, a sea state that is not completely wind-still will produce diffuse and random reflections, usually causing some signal to be returned. Furthermore, as mentioned by Andersson [10], the small capillary waves play a large role in the radar returns, and our surface does not include these. These are also mentioned as one of the parameters for the ocean surface

in the work by Nathanson [17]. The last factor is the limited size of the ocean's surface. In reality, the ocean surface measured is a lot larger, up to 1000 square metres [18], increasing the chance of some returned signals. Such a large surface, as mentioned by Andersson, who only used a 100x100m surface for the simulation, is not feasible to use for reasons including the time it will take to simulate.

The surface of SS 6 returned signals from several time stamps. The time stamps with a low amplitude showed a low amount of rays received after returning from a wave far in the back of the ocean surface. The longer travel distance, along with the low number of rays returned, resulted in a lower amplitude. The other time stamps that returned a high amplitude along with a high number of rays were time stamps 1 and 6. In both of these cases, rays were returned from a wave close to the camera. As previously mentioned, our limitations meant that only some time stamps returned rays, while in a realistic setting, some amount of rays should be returned for each time stamp.

Andersson [10] mentions an empirical study that uses the NRL model from the work of Greger-Hansen [18] to calculate the reflectivity of different sea states. The NRL model states that higher waves will have a higher reflectivity and thus produce more sea clutter compared to smaller waves. This relationship corresponds to our findings, where the higher waves produced a higher amplitude and the number of rays returned.

7.1.3 Test 3 - Ship in free space

The results from the ship in free space showed that the ship from the side did not return many rays or have a high amplitude since its hull is at an angle, as seen in Figure 6.6. However, when the ship faced the camera directly, or at an angle of 180 degrees, a high amplitude along with a high number of rays returned were measured. The front of the ship was especially good at returning rays because of its two relatively large and flat areas. As seen in Figure 6.5, some rays are reflected multiple times in the corners of some of the windows, creating a shape and propagation effect similar to a corner reflector.

In a similar test by Andersson [10, Fig. 11], a strong reflectivity was seen from the side of the ship in free space. This can be explained by their ship model having large flat areas in its hull compared to our ship's hull, which has a large number of angled areas, causing fewer rays to return. Otherwise, our test results are similar in that angles where the ship is directly facing the camera at degree 0 or facing away from the camera at degree 180 cause spikes of increased amplitude and rays to return.

7.1.4 Test 4 - Ship in ocean with sea states 5 and 6

The results of the ship in SS 5 were similar to those of the ship in free space, both in terms of amplitude and rays returned. Since the surface of SS 5 did not return any signals by itself, it is logical that the surface will not contribute as much when the ship is facing 0 or 180 degrees from the camera. However, a minor increase both in amplitude, as seen in Figure 6.8, and more visibly in rays returned can be seen at 90 degrees in Figure 6.9. At 300 degrees, a large increase in the number of rays returned is observed, with 485 rays returned compared to the ship in free space, which returned 143 rays. These results show that the RCS of the ship increases with a surface, something Andersson [10] also found.

The same test in SS 6 showed an increase in both amplitude and rays returned. Since this surface did return rays by itself, 387 of them, it was expected that as long as the ship did not block the surface from which these rays were reflected, an additional signal would be added on top of the returns measured from the ship. By looking at the simulation, this is not

the case for most of the angles, especially the ones where the ship is facing sideways to the camera. This test shows the effect of sea clutter, increasing the signal strength and RCS even though the object is the same as before and, thanks to the large waves, is somewhat hidden behind the surface. This means that a large part of the returned rays from certain angles, such as 90 degrees, are not from the ship but actually from the sea surface. Comparing the results from Andersson [10, Fig. 11], their results also show an increased return of signals, whereas, in free space, not a lot of returned signals were measured.

7.2 Method discussion

We have worked with perfect reflecting material in our tests, meaning that no energy is lost for any of the rays hitting an object. Thus, the implementation will only simulate perfect specular reflections. In reality, there are a lot of other physical behaviours that apply to each signal that will affect a radar's ability to detect targets. Each simulated ray would therefore need to apply additional effects for each bounce. To name some parameters, they would be:

Refraction - When a ray crosses between two materials with different indexes of refraction

Diffraction - How light bends around an object

Absorbing material - Different materials can absorb a portion of a signal and therefore reduce the receiving signal

Diffusion - Light and signals do not always reflect off an object in a perfect way, and there is some randomization to the way they can scatter off the object.

Atmosphere - If the atmospheric pressure changes, it can also bend the signal and mask certain areas, making it look like no signal is being picked up by the radar.

Our implementation has the limitation that only a small ocean surface can be simulated. Andersson [10] mentions that a limitation in her work is the size of the ocean surface measured as well. A large ocean surface with detailed waves is computationally very heavy, but it is needed in order to get a realistic representation of the radar wave propagation on a sea surface. Andersson further mentions the importance of including the centimetre-large waves known as capillary waves, which play a large role in the radar returns of a sea surface. Andersson does not, however, mention if the sea surface modelled in their report uses these but acknowledges that it is a highly complex area. Our ocean surface does not include capillary waves for these reasons.

It is possible to improve the performance of our ray tracer using different methods of accelerating the computations. A common method is called "Bounding Volume Hierarchy" (BVH)[20], mentioned in background 2.2. Our current implementation uses none of these techniques. Furthermore, something that was found out during the pre-study was that Unity recently released its ray tracing support out of its "experimental" phase. Because of limited time and documentation, this approach was not chosen. However, this may be the optimal way of implementing a ray tracer in Unity. The reason is that this API provides the user with a ray tracing pipeline built for implementing ray tracing. This includes acceleration structures that would increase the performance and thus the capability of tracing more detailed scenes. This is currently available in a beta version of Unity.

[20]Nvidia. *Ray Tracing*. Accessed 05.06.2023. URL: `https://developer.nvidia.com/discover/ray-tracing`.

7.3 The work in a wider context

Radar is a widely used technology and is implemented in many systems, not only for civilian applications but also in the military. The ethical aspects have to be taken into account for this reason. Is our implementation ethical is something we have reflected on. We do not think our contributions are unethical since they do not directly impact the way radar is used in the real world and we are not trying to improve existing radar-detecting technology. Our goal has been to study the Unity game engine's capabilities for simulating radar. The simulations might as well be those of a civilian radar used to detect objects in an ocean.

8 Conclusion

Our aim with this thesis has been to evaluate the possibility of simulating a radar in Unity and, furthermore, to simulate sea clutter that might occur when scanning for objects at sea. After the pre-study, it was decided to implement ray tracing with a compute shader. Because of the similar properties of ray tracing and radar wave propagation, along with the many related studies using ray tracing for radar wave propagation, it is considered the optimal approach. By implementing it using a compute shader, the GPU is used, enabling a vast amount of calculations to be done in a timely manner.

Some of the results were found to match empirical data and the results presented in the report by Andersson [10]. Findings such as the effect the different sea states had on the RCS, higher waves resulting in more sea clutter returned, and the ocean surface may increase the RCS of an object.

The main limitation of our implementation is that only a certain number of triangles are able to be traced. Measuring sea clutter is usually done over very large surfaces. Only using specular reflections and not including other wave propagation factors is also a limitation that decreases the realism of the implementation. Furthermore, for a realistic radar, the distance to the target should be further away, and the rays emitted should be more concentrated in a specific area and not evenly spread.

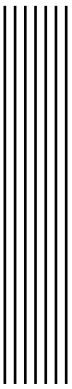
Even though there are several limitations in the current implementation, many of them can be solved by either improving the existing solution or using a different implementation approach. The limitation of how many triangles are able to be traced could be fixed by using some type of acceleration structure or implementing a ray tracer using the ray tracing API with Unity. The different ways rays are propagated on different materials have already been solved for light simulations and should thus be able to be implemented in a similar way in a radar implementation. The one limitation of our implementation that needs to be further researched is generating a detailed and realistic enough ocean surface.

Combining the results with the current limitations and their proposed solutions, it can be concluded that using ray tracing with a game engine such as Unity is a feasible approach for simulating sea clutter.

8.1 Future works

This thesis has focused on simulating radar waves in a 3D environment using the Unity game engine. There is a lot more that can be done with this type of approach and also with other game engines, for example, Unreal Engine.

We have seen that ray tracing, using compute shaders, is a quick and efficient way of computing a substantial amount of information since it uses the graphics card for its calculations. By improving the existing implementation in the ways mentioned, there is the potential to include additional variables and trace more complex scenes. Something that could be worth exploring is having the range of an object be further away from the camera and having each ray include additional or different parameters depending on the type of mesh of the object it hits. For example, adding a diffused parameter for randomising the ray's trajectory after it bounces off an object, and having the objects in the scene composed of different absorbing materials, thus lowering the energy of the signal.



Bibliography

[1] D. K. Barton. *Radar system analysis and modeling*. Norwood, MA: Artech House, 2004.

[2] E. Egea-Lopez, J. M. Molina-Garcia-Pardo, M. Lienard, and P. Degauque. "Opal: An open source ray-tracing propagation simulator for electromagnetic characterization". In: *PLOS ONE* 16.11 (2021), pp. 1–19. DOI: 10.1371/journal.pone.0260060.

[3] E. Yilmaz. "Radar Sensor Plugin for Game Engine Based Autonomous Vehicle Simulators". MA thesis. Harvard Extension School, 2020. URL: https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37365609.

[4] J. Latger and T. Cathala. "Millimeter waves sensor modeling and simulation". In: *Millimetre Wave and Terahertz Sensors and Technology VIII*. Ed. by N. A. Salmon and E. L. Jacobs. Vol. 9651. International Society for Optics and Photonics. SPIE, 2015, 96510B. DOI: 10.1117/12.2195038.

[5] N. Douchin, C. Ruiz, J. Israel, and H.-J. Mametsa. "SE-Workbench-RF: Performant and High-Fidelity Raw Data Generation for Various Radar Applications". In: *2019 20th International Radar Symposium (IRS)*. 2019, pp. 1–10. DOI: 10.23919/IRS.2019.8768180.

[6] M. Ulmstedt and J. Stålberg. "GPU accelerated ray-tracing for simulating sound propagation in water". MA thesis. Linköpings Universitet, 2019. URL: https://www.diva-portal.org/smash/get/diva2:1352170/FULLTEXT01.pdf.

[7] N. Peinecke, H.-U. Döhler, and B. R. Korn. "Simulation of imaging radar using graphics hardware acceleration". In: *Enhanced and Synthetic Vision 2008*. Ed. by J. J. Güell and M. U. de Haag. Vol. 6957. International Society for Optics and Photonics. SPIE, 2008, p. 69570L. DOI: 10.1117/12.782622.

[8] S. Wang, S. Heinrich, M. Wang, and R. Rojas. "Shader-based sensor simulation for autonomous car testing". In: *2012 15th International IEEE Conference on Intelligent Transportation Systems*. 2012, pp. 224–229. DOI: 10.1109/ITSC.2012.6338904.

[9] M. Ciarambino, Y.-Y. Chen, and N. Peinecke. "A game engine-based millimeter wave radar simulation". In: *Virtual, Augmented, and Mixed Reality (XR) Technology for Multi-Domain Operations II*. Vol. 11759. SPIE. 2021, pp. 21–29. DOI: 10.1117/12.2587595.

[10] Å. Andersson. *Modellering av sjöytans inverkan på radarsignaturen för fartyg*. Report FOI-R–4151–SE. Sensorer och signaturannpassning. FOI, Dec. 2015. URL: https://www.foi.se/rest-api/report/FOI-R--4151--SE.

[11] J. Freiknecht, C. Geiger, D. Drochtert, W. Effelsberg, and R. Dörner. "Game Engines". In: *Serious Games: Foundations, Concepts and Practice* (2016), pp. 127–159. DOI: 10.1007/978-3-319-40612-1_6.

[12] E. F. Anderson, L. McLoughlin, J. Watson, S. Holmes, P. Jones, H. Pallett, and B. Smith. "Choosing the infrastructure for entertainment and serious computer games-a white-room benchmark for game engine selection". In: *2013 5th international conference on games and virtual worlds for serious applications (VS-GAMES)*. IEEE. 2013, pp. 1–8. DOI: 10.1109/VS-GAMES.2013.6624223.

[13] K. Halladay. *Practical Shader Development : Vertex and Fragment Shaders for Game Developers*. Apress, 2019. DOI: 10.1007/978-1-4842-4457-9.

[14] A. Navarro and D. Guevara. "Applicability of game engine for ray Tracing Techniques in a Complex Urban Environment". In: *2010 IEEE 72nd Vehicular Technology Conference-Fall*. IEEE. 2010, pp. 1–5. DOI: 10.1109/VETECF.2010.5594343.

[15] H. Ling, R.-C. Chou, and S.-W. Lee. "Shooting and bouncing rays: calculating the RCS of an arbitrarily shaped cavity". In: *IEEE Transactions on Antennas and Propagation* 37.2 (1989), pp. 194–205. DOI: 10.1109/8.18706.

[16] P. G. Alf Sandqvist. *Lärobok i telekrigföring för luftvärnet – Radar och radartaktik*. Försvarsmakten, 2004.

[17] F. Nathanson, J. Reilly, and M. Cohen. *Radar Design Principles: Signal Processing and the Environment*. McGraw-Hill, 1991. ISBN: 9780071127264.

[18] V. Gregers-Hansen and R. Mital. "An Improved Empirical Model for Radar Sea Clutter Reflectivity." In: *IEEE Transactions on Aerospace and Electronic Systems, Aerospace and Electronic Systems, IEEE Transactions on, IEEE Trans. Aerosp. Electron. Syst* 48.4 (2012), pp. 3512–3524. ISSN: 0018-9251. DOI: 10.1109/TAES.2012.6324732.

[19] M. Horst, F. Dyer, and M. Tuley. "Radar sea clutter model". In: *Antennas and Propagation* (1978), pp. 6–10.

[20] J. Reilly and G. Dockery. "Influence of evaporation ducts on radar sea return". In: *IEE Proceedings F (Radar and Signal Processing)*. Vol. 137. 2. IET. 1990, pp. 80–88. DOI: 10.1049/ip-f-2.1990.0012.

[21] I. Antipov. *Simulation of sea clutter returns*. Tech. rep. DEFENCE SCIENCE and TECHNOLOGY ORGANISATION CANBERRA (AUSTRALIA), 1998.

[22] B. Yang, M. Jiang, and J. Wang. "Analysis of Extendibility of Sea Clutter Model in High Sea States Based on Measured Data." In: *2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA)* (2022), pp. 140–143. ISSN: 978-1-6654-5911-2. DOI: 10.1109/CVIDLICCEA56201.2022.9825361.

[23] J. Tessendorf. "Simulating Ocean Water". In: *SIG-GRAPH'99 Course Note* (2001). URL: https://people.computing.clemson.edu/~jtessen/reports/papers_files/coursenotes2004.pdf.

[24] D. Begault. "3-D Sound For Virtual Reality and Multimedia". In: (Sept. 2001). URL: https://ntrs.nasa.gov/citations/20010044352.

[25] OKTAL-SE. *OKTAL-SE Publications: Radio-Frequency (radar)*. Ed. by OKTAL-SE. Accessed 12.04.2023. URL: https://www.oktal-se.fr/publications/#37-15-wpfd-radio-frequency-radar.

[26] S. Watts. "Modeling and Simulation of Coherent Sea Clutter." In: *IEEE Transactions on Aerospace and Electronic Systems, Aerospace and Electronic Systems* 48.4 (2012), pp. 3303–3317. ISSN: 0018-9251. DOI: 10.1109/TAES.2012.6324707.

[27] Z. Yun and M. F. Iskander. "Ray Tracing for Radio Propagation Modeling: Principles and Applications". In: *IEEE Access* 3 (2015), pp. 1089–1100. DOI: 10.1109/ACCESS.2015.2453991.

[28] M. Beig, B. Kapralos, K. Collins, and P. Mirza-Babaei. "An Introduction to Spatial Sound Rendering in Virtual Environments and Games". In: *The Computer Games Journal* 8.3 (2019), pp. 199–214. ISSN: 2052-773X. DOI: 10.1007/s40869-019-00086-0.

[29] L. Savioja and U. P. Svensson. "Overview of geometrical room acoustic modeling techniques". In: *The Journal of the Acoustical Society of America* 138.2 (2015), pp. 708–730. DOI: 10.1121/1.4926438.

[30] M. Wolf, P. Trentsios, N. Kubatzki, C. Urbanietz, and G. Enzner. "Implementing Continuous-Azimuth Binaural Sound in Unity 3D". In: *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*. 2020, pp. 384–389. DOI: 10.1109/VRW50115.2020.00083.

A Appendix

This chapter contains the data collected from our tests.

I Tables

Time t	Amplitude (dB)	Rays in	Object Ray hits	Ratio
1	48.20876	387	1318956	0.000293414
2	0	0	1379363	0.000000000
3	0	0	1510754	0.000000000
4	2.466245	7	1364198	0.000005131
5	0	0	1384639	0.000000000
6	38.31813	1240	1399344	0.000886130
7	0	0	1493893	0.000000000
8	0	0	1377939	0.000000000
9	0.9999999	1	1257377	0.000000795
10	0	0	1302686	0.000000000

Table A.1: Rays hitting an ocean with SS 6 in 10 different time stamps.

Object angle (deg)	Amplitude (dB)	Rays in	Object Ray hits	Ratio
0	46.66582	676	217470	0,003108475
1	36.36328	676	217425	0,003109118
2	23.15611	676	217332	0,003110449
3	22.0108	676	217159	0,003112926
4	12.89441	676	216982	0,003115466
5	10.84723	702	216702	0,003239472
6	7.817428	676	216366	0,003124336
7	3.721986	676	215972	0,003130035
8	6.177623	676	215522	0,003136571
9	1.972834	676	215001	0,003144171
10	5.316562	702	214421	0,003273933
11	2.171506	676	213787	0,003162026
12	3.877004	730	213082	0,003425911
13	4.695029	1011	212316	0,00476177
14	2.434667	1338	211515	0,006325792
15	2.036887	1452	210663	0,006892525
16	1.610066	1445	209738	0,006889548
17	3.895624	1489	208761	0,007132558
18	6.681597	1480	207719	0,00712501
19	3.337313	1450	206604	0,007018257
20	6.404984	1445	205444	0,007033547
21	1.972385	1442	204214	0,00706122
22	3.353456	1463	202917	0,007209844
23	1.418771	1464	201553	0,007263598
24	2.208403	1431	200136	0,007150138
25	2.007055	1428	198621	0,007189572
26	5.756506	1452	197097	0,007366931
27	5.896248	1452	195497	0,007427224
28	42.24391	1424	193819	0,007347061
29	10.32077	1450	192081	0,007548899
30	6.726961	1450	190278	0,007620429
31	11.11066	1453	188418	0,007711577
32	7.253452	1454	186482	0,007796999
33	1.442497	1456	184493	0,007891898
34	11.76207	1486	182428	0,008145679
35	5.942334	1489	180317	0,00825768
36	11.26303	1440	178131	0,008083938
37	14.69102	1425	175883	0,008101977
38	2.894636	1327	173573	0,007645198
39	6.684335	1041	171213	0,006080146
40	2.434766	455	168771	0,002695961
41	0.1501886	2	167752	0,000011922
42	0	0	167587	0
43	0	0	167344	0
44	0	0	167103	0
45	0	0	166834	0

Table A.2: The resulting amplitude when rotating the corner reflector, 1 degree at a time, and emitting 2.073.600 rays each time.

Ship angle (deg)	Amplitude (dB)	Rays in	Object Rays hit	Ratio
0	412.7191	2018	125508	0.016078656
15	2.492766	42	191450	0.000219378
30	1.192167	18	272511	0.000066052
45	8.336901	60	324771	0.000184746
60	14.39047	105	346423	0.000303098
75	18.18481	136	346168	0.000392873
90	30.20041	714	331864	0.002151484
105	32.38122	734	324897	0.002259178
120	2.547861	7	310341	0.000022556
135	10.10438	36	285294	0.000126186
150	4.777356	50	245273	0.000203854
165	1.869673	63	190870	0.000330068
180	313.5358	769	161522	0.004760961
195	5.076405	37	191333	0.000193380
210	2.846501	46	246187	0.000186850
225	4.829083	8	286512	0.000027922
240	3.183795	14	311137	0.000044996
255	26.09214	755	324442	0.002327072
270	6.346202	138	329567	0.000418731
285	23.55026	92	343454	0.000267867
300	10.36733	143	343436	0.000416380
315	20.5091	76	322372	0.000235752
330	13.64809	102	272182	0.000374749
345	2.88715	7	192509	0.000036362
360	412.7191	2018	125508	0.016078656

Table A.3: Rays hitting a rotating ship in free space.

Ship angle (deg)	Amplitude (dB)	Rays in	Object Rays hit	Ratio
0	412.7191	2018	1371026	0.001471890
15	2.492766	42	1376931	0.000030503
30	1.192167	18	1385497	0.000012992
45	8.336901	60	1393467	0.000043058
60	14.39047	105	1400260	0.000074986
75	18.17033	300	1396103	0.000214884
90	44.43472	804	1383894	0.000580969
105	33.37251	735	1382635	0.000531594
120	1.8572	13	1383396	0.000009397
135	10.10399	36	1384111	0.000026009
150	4.777356	50	1382106	0.000036177
165	1.869673	63	1374517	0.000045834
180	313.5358	769	1368847	0.000561787
195	4.115617	40	1375398	0.000029082
210	4.451248	60	1383902	0.000043356
225	4.829083	8	1387602	0.000005765
240	3.183795	14	1387483	0.000010090
255	26.31306	756	1384184	0.000546170
270	7.663702	148	1383579	0.000106969
285	23.55232	92	1393921	0.000066001
300	16.54374	485	1397726	0.000346992
315	20.50153	76	1391797	0.000054606
330	14.49085	117	1384688	0.000084496
345	2.888661	7	1377017	0.000005083
360	413.0969	2018	1371026	0.001471890

Table A.4: Rays hitting a rotating ship in an ocean with SS 5.

Ship angle (deg)	Amplitude (dB)	Rays in	Object Rays hit	Ratio
0	436.3851	2218	1350656	0.001642165
15	50.26257	429	1353080	0.000317054
30	47.03388	405	1359469	0.00029791
45	55.81226	538	1362204	0.000394948
60	63.7632	505	1365858	0.000369731
75	55.41702	531	1368670	0.000387968
90	24.11356	1203	1364147	0.00088187
105	81.26122	1089	1360186	0.000800626
120	51.18072	393	1355810	0.000289864
135	51.89151	396	1357600	0.000291691
150	46.2159	391	1357121	0.00028811
165	46.03915	442	1351537	0.000327035
180	287.1039	1100	1348121	0.00081595
195	44.90582	422	1351897	0.000312154
210	49.72807	431	1358218	0.000317328
225	52.5461	395	1360172	0.000290404
240	46.39145	401	1358851	0.000295102
255	52.32863	1131	1358563	0.000832497
270	39.96687	637	1355773	0.000469843
285	31.80833	559	1355346	0.000412441
300	57.76644	529	1355853	0.00039016
315	48.92788	583	1357712	0.000429399
330	61.68186	482	1357629	0.000355031
345	38.34174	425	1352473	0.000314239
360	436.6895	2218	1350656	0.001642165

Table A.5: Rays hitting a rotating ship in an ocean with SS 6.