

Integrating custom environment and nighttime effects into NAUTIS

CMGT Graduation Report

Abstract

VSTEP is a software development company that specializes in creating simulators for realistic virtual safety training. During my graduation period, I was assigned to work as a graphics programmer intern on environmental and nighttime effects in the company's ship simulator NAUTIS. In the current version of the simulator, the night visuals are not realistic enough which takes away a big part of the training value in dark scenarios. Furthermore, a client of VSTEP requested the implementation of convincing warfare and nighttime visual effects for their training sessions. To propose a solution to these requests, ways to integrate and alter custom effects in the underlying engine, UNIGINE, were researched. During the assignment, the relevant effects with the most visual impact on dark scenes were chosen to work on.

These include the adjustment of the moon shading and water reflections on the ocean surface and the implementation of tidal-lock and celestial body obstructions for the moon sphere. Aside from that, the environment fog was improved by addressing visual artifacts and introducing a different environment preset blending system that is based on the visibility and cloudiness condition parameters. Besides that, light source warfare particle effects, that have the ability to illuminate surrounding geometry, were implemented into the application. Next to that, a performance bottleneck related to using a large number of dynamic lights on transparent surfaces, such as the ocean, was examined. In regards to that, the clustered forward shading optimization was researched.

As a result, it was found out that lighting and environmental shading is the most important for nighttime scenarios as these light sources draw the most attention during the absence of sunlight. Moreover, NAUTIS still has room for dynamic lighting optimizations that would increase the visual quality of a scene without reducing the performance. To achieve that, UNIGINE offers to integrate custom effects by using the engine's visual effect objects, such as particle systems, or to create new effects by introducing entirely new shader passes into its rendering pipeline.

Table of contents

List of figures	4
1. Introduction	5
1.1 Company outline	5
2. Project outline	7
2.1 Assignment description	7
2.2 Client demands	7
2.3 Problem analysis	8
2.4 Problem definition	9
2.5 Assignment scope	9
3. Research	11
3.1 Research questions	11
3.2 Methodology	11
3.3 General theory & Findings	11
4. Professional product	15
4.1 Moon visuals	15
4.1.1 Theory	15
4.1.2 Implementation	17
4.1.3 Evaluation	18
4.2 Environment fog	20
4.2.1 Theory	21
4.2.2 Implementation	22
4.2.3 Evaluation	25
4.3 Warfare particle effects	27
4.3.1 Theory	27
4.3.2 Implementation	30
4.3.3 Evaluation	31
4.4 Nix light system	34
4.4.1 Theory	34
5. Conclusion	36
6. Bibliography	38
7. Appendices	42
Appendix I. Basic engine principles in UNIGINE	42
Appendix II. Professional product demo database	43
Appendix III. Visual effects company survey	44

List of figures

Figure 1. Breaching sunlight in constrained visibility and lack of particle illumination.	8
Figure 2. UNIGINE rendering pipeline.	12
Figure 3. Light performance profiling results of a test island scene with transparent water.	14
Figure 4. Previous moon specular reflections and invisible moon issue.	15
Figure 5. Atmospheric light scattering based on the Mie scattering approximation.	16
Figure 6. Moon specular reflection configuration parameters.	17
Figure 7. Moon phase bias transition visualization.	17
Figure 8. Improved moon specular water reflections and correct moon shading by the sun.	18
Figure 9. Average company survey results of the moon visuals.	19
Figure 10. Bright sky and black ring issue on the ocean surface.	20
Figure 11. Visualization of the camera view frustum based on the clipping planes.	21
Figure 12. Example of a fog based volumetric object.	22
Figure 13. Fog implementation visual comparison.	24
Figure 14. Fog visual results.	25
Figure 15. Average company survey results of the environment fog.	26
Figure 16. Warfare particle that lacks illumination of surrounding geometry.	27
Figure 17. Bloom and emission effects used to fake self-glow of the plane.	28
Figure 18. Reflection capabilities of screen-space reflections and planar reflections.	29
Figure 19. Particle lighting passing geometry and illuminating unwanted areas.	31
Figure 20. Options to adjust the particle effects and light animation.	32
Figure 21. Average company survey results of the warfare particles.	33
Figure 22. Visualization of Forward+ shading approach.	35
Appendix I. Figure 1. Example of a custom base material file.	43
Appendix III. Figure 1. Moon visuals part 1 of the company internal evaluation survey.	44
Appendix III. Figure 2. Moon visuals part 2 of the company internal evaluation survey.	45
Appendix III. Figure 3. Environment fog part 1 of the company internal evaluation survey.	46
Appendix III. Figure 4. Environment fog part 2 of the company internal evaluation survey.	47
Appendix III. Figure 5. Warfare particles part 1 of the company internal evaluation survey.	48
Appendix III. Figure 6. Warfare particles part 2 of the company internal evaluation survey.	49

1. Introduction

During my graduation internship I worked at the company VSTEP Simulation, which is a software development company that specializes in creating accurate simulators for virtual safety training. Throughout my stay at the company, I was tasked to contribute to their ship simulator NAUTIS by adding new visual effects for night scenarios as well as improving existing environment effects. This included investigating and altering the currently used render systems to optimize their overall performance for various visual effects while still delivering the desirable image quality. The features of the product are mainly demand driven, which is why it is constantly under development to satisfy the customer's needs. Since the simulations are meant to partially replace the real-world training with virtual training, it is especially crucial to ensure that as many features as possible behave similar to how they would in reality. Due to that, I was required to provide visuals of my implementations for the proposed graphical effects in a realistic way. In order to achieve that, I was given full access to the source code of both NAUTIS and its underlying engine UNIGINE as well as I was given access to its SDK browser. This did not only allow me to work on custom effects in DirectX High Level Shading Language (HLSL) and UNIGINE's Unified UNIGINE Shading Language (UUSL), but it also allowed me to dive into the engine's and NAUTIS' overall architecture by utilizing C++ or UNIGINE Script. Though, it is important to note that UNIGINE no longer offers new contracts that allow for access to the full engine source code.

1.1 Company outline

VSTEP Simulation is a software development company based in Rotterdam, in the Netherlands, which consists of about 70 employees and specializes in creating realistic training simulators. It was founded in 2002 to first develop new technologies that allow virtual safety training for the incident command field and to later expand into the maritime domain. At the time of writing, the company offers two different softwares for both ship and emergency situation simulations, which are NAUTIS and Response Simulator (RS) respectively. Looking at the previous clients of the company, it can be seen that VSTEP is delivering their products to companies that are involved in ship transports for trades and collaborates with governmental facilities, such as the port of Rotterdam. With the help of these simulators, their clients are able to construct custom scenarios based on existing real-world environments, in order to use them for virtual safety training and student examinations. Furthermore, the ability to customize these scenarios allow the training instructors to prepare training sessions that are otherwise too dangerous or difficult to set up in real life. For example, training scenarios in harbour areas with ship traffic on NAUTIS; or

1. Introduction

firefighter training sessions in regions severely affected by toxic substances using RS. Besides that, since NAUTIS is usually sold together with a physical installation that is tailored to the customer's training demands, the client's students are able to control the ship's bridges with similar hardware as they would find on a real ship. In addition to that, VSTEP closely collaborates with their clients by selling them their products and offering long-time support for their simulators and installations. Furthermore, VSTEP managed to obtain the ISO 9001:2015 certification from the international classification society DNV GL which allows trainees to officially absolve a part of their training sessions and examinations in the company's simulations.

2. Project outline

2.1 Assignment description

My general graduation task was to implement and improve various environmental visual effects, such as moon visuals for water reflections at night, to make them also look convincing during nighttime scenarios. Moreover, I was also requested to improve the currently used light system in the custom Nix ocean created by VSTEP. Next to the water illumination, one of the companies' clients, requested a list of warfare related particle effects, such as muzzle flashes or explosions, which should be developed with my assistance. In regards to the light system, the aim was to allow the usage of more light sources while still retaining a decent performance and physical accuracy. Specifically, the light sources that illuminate the ocean should ideally take occlusion into account to avoid having them shine through other opaque geometry. The physical accuracy aspect should additionally be ensured for the visualization of the fog effect, which is used to constrain the visibility of the user. Due to that reason, I was also tasked to refine the currently implemented fog behavior and investigate some issues that were introduced with a previous upgrade of the core engine. Most of the mentioned effects were already present in some form in UNIGINE, although their implementation does either not fit to VSTEP's needs or are too resource demanding for the simulation visualizer. Moreover, NAUTIS was recently upgraded from UNIGINE version 2.5 to 2.9, which caused some effects, like fog, to behave differently. Due to that, they potentially have to be revisited and adjusted when the company decides to upgrade to a newer version.

2.2 Client demands

Specifically for their project, VSTEP received a list of desired visual effects of environmental nighttime visuals and combat effects the client wants to be present in NAUTIS. Because a realistic presentation of the night is also valuable for orientation and navigation purposes of trainees, VSTEP had already planned to improve on them before the project with their client was established. In terms of warfare effects, the client requested the implementation of particle effects which would additionally contribute to the overall illumination of a training dark environment. It is important to note that the alterations in the application must ensure a smooth performance in the simulation as a low frame rate can negatively influence both the simulation itself and the training experience of the trainees. These modifications then have to be tested and approved by both the quality assurance department in the company and the client to confirm that the additions meet their expectations. Furthermore, these improvements

2. Project outline

have to be reviewed by me on the physical ship bridge installation in the company to verify their looks in their final training environment. Aside from that, VSTEP expects me to document my conducted work according to the company guidelines so that further changes or extensions can be introduced at a later point.

2.3 Problem analysis

The current lighting system seems to be a bottleneck on the GPU as it fairly limits the amount of lights that can be present in the environment without affecting the frame rate too much. On top of that, the interaction between different visibility settings, which is implemented by a distance fog effect, and the cloud system is not as desired as these clouds are sometimes not affecting the environment light scattering even though that should happen at all times (see Figure 1, left image). Additionally, opposed to the day-time simulations, it is currently hard to conduct a wide range of training sessions during the night since the current effect implementations did not account for that which is why the scene mostly appears black (see Figure 1, right image). When looking at the real world, it is clear that it is not common to have nights that are this dark due to the fact that there is often indirect lighting present. Because of that, lighting from all kinds of light sources has to be taken into account. This includes, for example, reflections of celestial bodies in the water surface or the general environment illumination based on various light sources, such as the moon or the warfare particle effects.

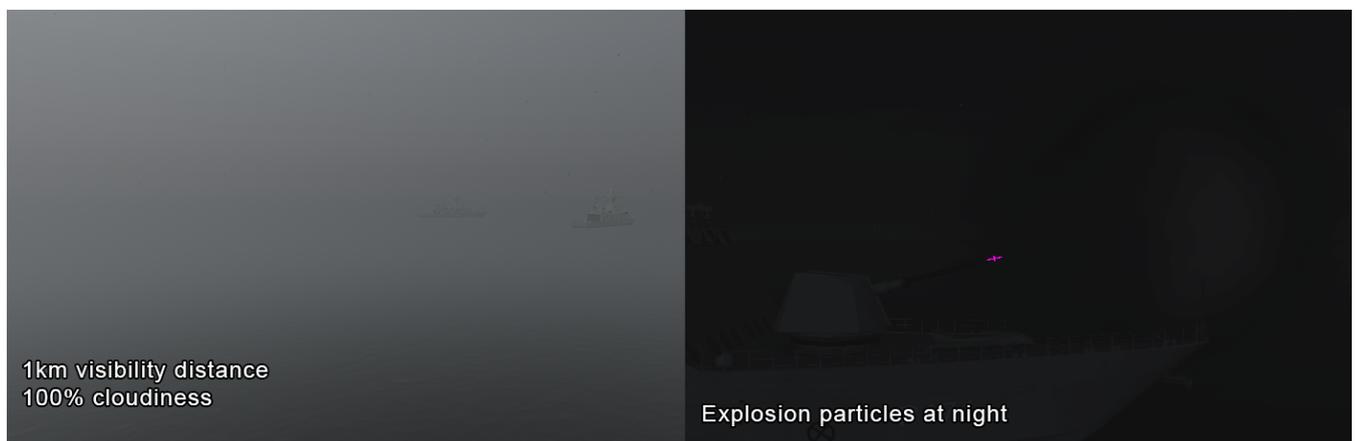


Figure 1. Breaching sunlight in constrained visibility and lack of particle illumination.

2. Project outline

2.4 Problem definition

To verify the performance and physical correctness, the effects have to be tested in a wide range of environments with various conditions. Moreover, potential interactions between certain related effects have to be tested as well in order to accommodate for unwanted behavior. Furthermore, I do not only have to analyse the visual appearance and the performance on my local development PC, but I also had to check for differences on the ship's bridge installation at VSTEP. In order to address performance issues for already implemented visual effects, it is necessary to propose certain adjustments to them or to research further options for their optimization. Moreover, since UNIGINE already has a set render pipeline, it can be **difficult to integrate custom render passes** into the engine as it has more control over the rendering sequence. Due to that reason, VSTEP decided to implement a handful of custom external shader passes which are treated like one forward pass in the rendering sequence of the engine. To come to a conclusion, the main problem is that certain **environmental effects do not behave as expected in dark scenes** while others **take away too much performance** which could result in potential limitations for the trainees.

2.5 Assignment scope

For this assignment, I was not required to design any user interface (UI) elements but I was required to extend existing UI elements with more configuration options, if applicable. Besides that, I was not allowed to change anything in the UNIGINE core source code unless it was previously discussed and approved by the project team. Therefore, I mostly needed to work with and in custom systems and shaders introduced by VSTEP. When I first received the task to work on the environment fog on Nix' surface, it was due to the massive changes that were introduced when updating UNIGINE from version 2.5 to 2.9. The main problem was that the way the environment was handled within the engine was severely changed between these two versions which is why I needed to address the migration issues regarding the fog. In regards to the particle systems, UNIGINE does not offer an option to allow the particles to illuminate surfaces around them like light sources do, which is why I was required to find and implement a solution which can be used for the warfare particles. Moreover, I had to research the possibilities of reflecting particles on the ocean surface without delivering a prototype for it. Additionally, I was unable to test the implementation of the warfare effects with the final users, because VSTEP's client will take care of that and provide feedback. Aside from that, the current light system for dynamically illuminating surfaces had to be investigated to propose improvements to shade Nix' ocean surface. Generally speaking, all

2. Project outline

assigned tasks involve a combination of using the previously developed systems and programming entirely new implementations for them. An example would be, that an underlying shader has to be extended with additional lighting calculations and the system that uses or manages that shader has to be altered to reflect the shader changes. That way the work is conducted on both, the back-end and the front-end of the application.

As it is only possible to work with a remote desktop application due to the corona crisis at this point in time, it is currently impossible to test the implementations in the simulation room at the company. Due to these constraints, the testing process for the visual effects has to be changed to taking screenshots and videos on the remote desktop in order to inspect visuals on my own PC. To verify the effect implementations itself, the screenshots have to be sent to my graduation team and the pull requests of the implementations have to be reviewed. In terms of testing and measuring the performance of effects on the remote PC, it turned out to be not as insightful as initially anticipated. This was especially the case since the installed GTX 970 graphics card has fairly limited computation power for NAUTIS and my assigned remote PC suffered from a few hardware related issues in general.

Aside from that, the corona crisis caused the communication to be shifted to Microsoft Teams, which generally made it less fluid to investigate problems and communicate when compared to the office. At last, based on the development workflow at VSTEP, I was required to manage my work with the help of work items and pull requests on the Azure DevOps cloud service.

3. Research

3.1 Research questions

Main research question:

How to integrate custom environment and warfare effects into UNIGINE in NAUTIS that achieve the required visual quality, realism and performance in order to enable VSTEP's clients to conduct virtual safety training in combat and nighttime environments?

Sub research questions:

1. How to integrate custom effects into UNIGINE?
2. What are the bottlenecks in the application regarding the assignment scope?
3. How should visual effects behave in dark scenes?
4. Which effects have to be improved or implemented for a convincing nighttime scene?

3.2 Methodology

It was often required to consult team colleagues and company experts for their expertise on simulator and naval topics. To research the exact behaviour in NAUTIS, however, it was often necessary to conduct self-research by analysing the program's code base. Moreover, the application offers to profile the performance with the use of the microprofiler of the engine which is used to assess the implementations. Aside from that, VSTEP's project management platform Azure DevOps sometimes provided important information regarding specific work items and could be used to easily search for specific code snippets on entire repository branches. If none of the previously described methods were applicable to find a solution for a problem in the assignment, online desk-research and team discussions were conducted to come up with possible approaches. However, sometimes UNIGINE's engine architectures restricted us to find suitable processes to address the issues which is why it was sometimes required to contact the UNIGINE developers in their forum.

3.3 General theory & Findings

When it is required to integrate custom effects into UNIGINE, the engine offers quite a few options to do so. Before going into detail, though, it is first necessary to understand how the engine itself (see Appendix I) and the rendering in the engine generally functions. The rendering sequence consists of several shader passes that are executed in a set order to compose the final scene image that is seen on the screen (see Figure 2).

3. Research

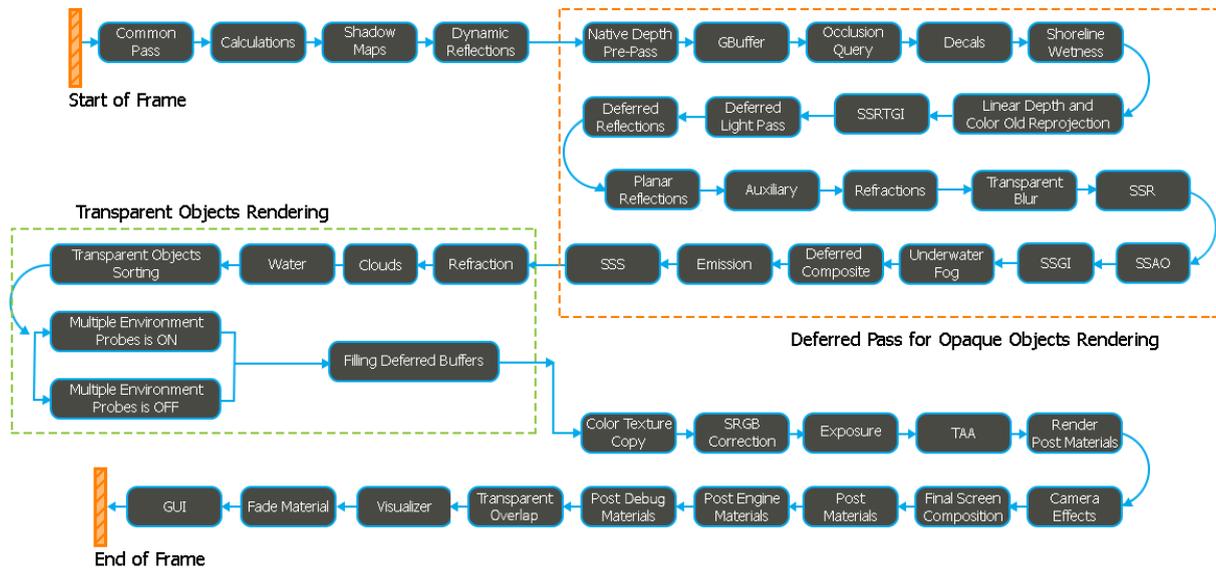


Figure 2. UNIGINE rendering pipeline (UNIGINE Corp., 2019).

To do so, the engine first calculates all relevant data for the frame, such as shadow maps and reflection cubemaps, and then proceeds to execute the deferred passes which are followed by the forward rendering passes (UNIGINE Corp., 2019). Afterwards, several post-processing effects are applied to the composited image of the previous passes and the graphical user interface is rendered. During the deferred passes, all opaque geometry is rendered into a set of screen-space textures which is commonly referred to as G-Buffer (De Vries, 2014). These texture buffers are then used to compute various image effects, such as screen space reflections (UNIGINE Corp., 2019). After all required textures have been rendered, the scene image containing only solid objects will be composited and lit by the relevant light sources in a single shading pass. In succession to that, the forward passes are executed to render transparent objects. To avoid visual artifacts, the render order is specified by sorting the objects based on their depth in the scene. When each transparent object is rendered, it is lit individually by the relevant light sources and added to the scene texture buffers of the deferred passes (UNIGINE Corp., 2019). Once all geometry has been rendered, gamma correction and exposure are applied to the scene image before the specified screen-space post-processing effects will be applied. For instance, the Bloom effect, which is responsible for visualizing a blurred glow around bright objects, is rendered during this stage in the sequence.

When talking about a shader pass in UNIGINE, it traditionally consists of at least a vertex shader and a fragment shader (or pixel shader) which are used to process the mesh vertices and the mesh pixels respectively (De Vries, 2014). On top of that, a pass can be extended by

3. Research

optional geometry and tessellation shader stages to alter the mesh geometry while rendering. The biggest difference between these shader types is that the geometry shader only allows to use already present primitives while tessellation shaders can be used to subdivide vertices to create a denser mesh (OpenGL Wiki, 2019). Besides that, the engine also allows the usage of individual compute shaders to execute calculations on the GPU instead of the CPU and to read and write buffer data (UNIGINE Corp., 2019). Moreover, all shaders can either be written in either shader language: UUSL, GLSL and HLSL.

If the engine's pipeline requires developers to introduce entirely new passes, as it was the case with the Nix rendering pipeline, it is possible to use a certain external shader which is responsible for fetching the externally rendered pixel data and forwarding it to UNIGINE to have it contribute to the final composited image. For instance, the ocean surface is rendered in DirectX with the help of several compute shader passes, a tessellation pass and the default vertex and fragment passes. The resulting texture of the final ocean is then bound to the external object shader and integrated into the specified render pass in the render sequence, which, in this case, was the ambient (or forward) pass.

Utilizing this technique can help when attempting to optimize render systems that otherwise result in bottlenecks in the application. In the case of NAUTIS, the most apparent bottleneck regarding the assignment scope is the currently used light system for illuminating a scene where many transparent areas and dynamic lights are present. This can be seen in Figure 3, where the engine requires about 70% more time on average to render 100 additional optimized omni lights on transparent surfaces when compared to a scene that only contains one world light. Furthermore, the rendering times even spike to a maximum of roughly 4.7ms which results in an increase of about 135% when compared to the maximum of the first profile with the single light. Based on that, when a huge amount of lights are paired with more complex scenes and other demanding effects, the performance will significantly suffer.

In regards to Nix shader implementation, the used lights are collected and passed to the ocean fragment shader to calculate their contribution to the currently processed pixel. When light sources are rendered in UNIGINE's own rendering pipeline, an optimization approach is utilized to batch lights together which decreases the total amount of draw calls (UNIGINE Corp., 2019). In order to optimize the lighting for Nix as well, the same or a similar optimization has to be implemented in its external pipeline.

When working on the implementations for the discussed effects, it can be especially difficult to debug visuals by looking at the final scene image. Because of that, the usage of programs like NVIDIA Nsight are greatly desired, as they allow developers to sample one or more frames of their application in order to analyse the used textures, shaders, draw calls and more (NVIDIA Corp., 2020). Even though it can only be used together with an NVIDIA

3. Research

graphics card, it offers a wider spectrum of options to inspect and modify GPU data at runtime when compared to alternatives.

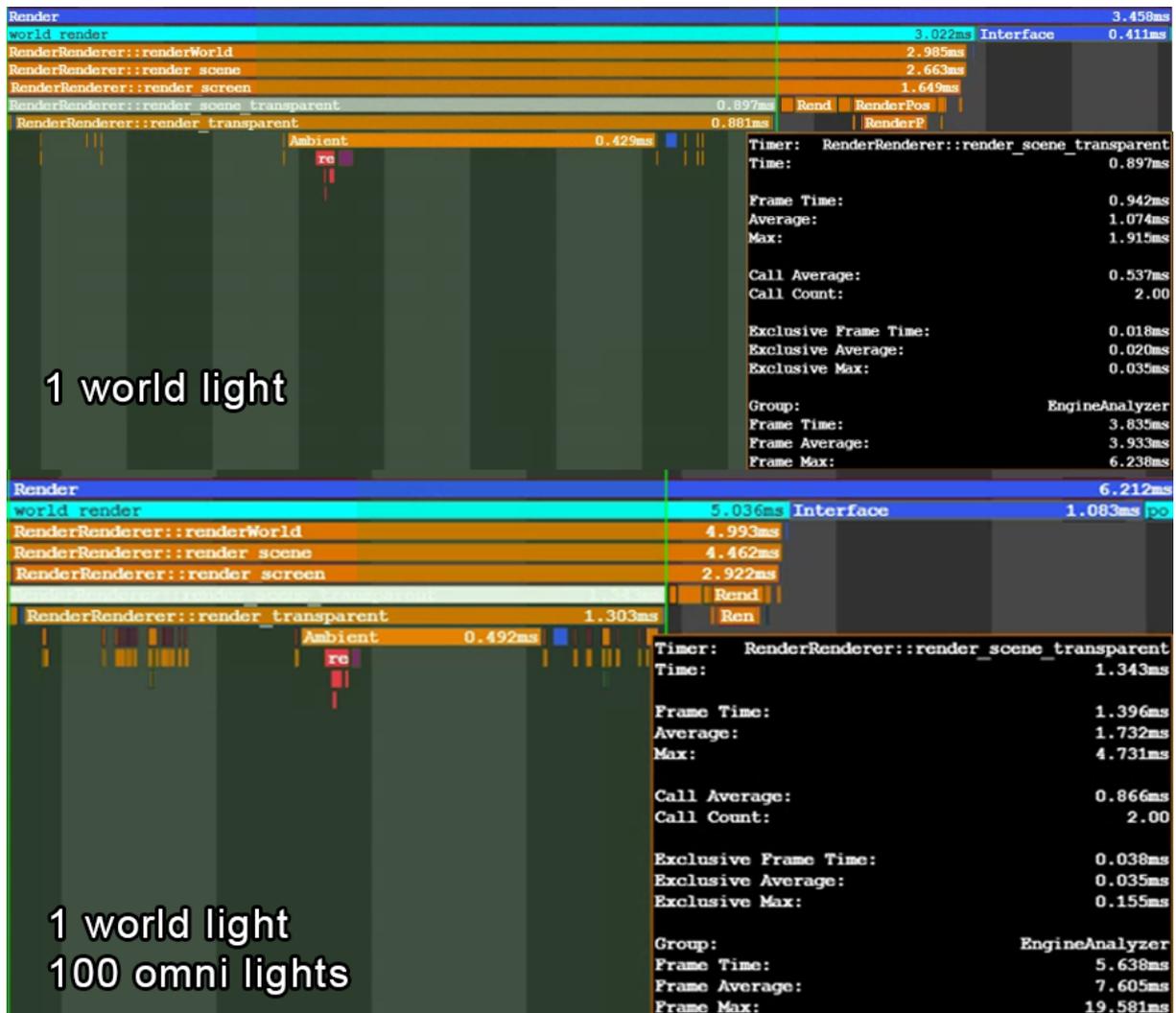


Figure 3. Light performance profiling results of a test island scene with transparent water.

The testing process is the most important, when checking visual effects that are intended to be used during the nighttime. The reason for this is that due to the absence of the sunlight at night, other light emitting effects are much more noticeable. Furthermore, it is even possible that some effects appear differently during the night than during the day. An example for this, are the warfare effects, which should barely contribute to the illumination of the water surface during the day as the sunlight is so strong that most other light sources can be neglected. During the night, the water surface illumination should only come from omnidirectional and spot lights as no directional light from the sun has to be taken into account.

4. Professional product

4.1 Moon visuals

In order to get used to working with NAUTIS and its underlying engine UNIGINE, the introductory task was to improve the moon visuals. Prior to working on that task, the moon was rendered as a simple alpha blended sphere that only received diffuse lighting from the sun and was transparent on its unlit side. However, this implementation suffered from a few issues that made the moon visualization not realistic as seen in Figure 4.

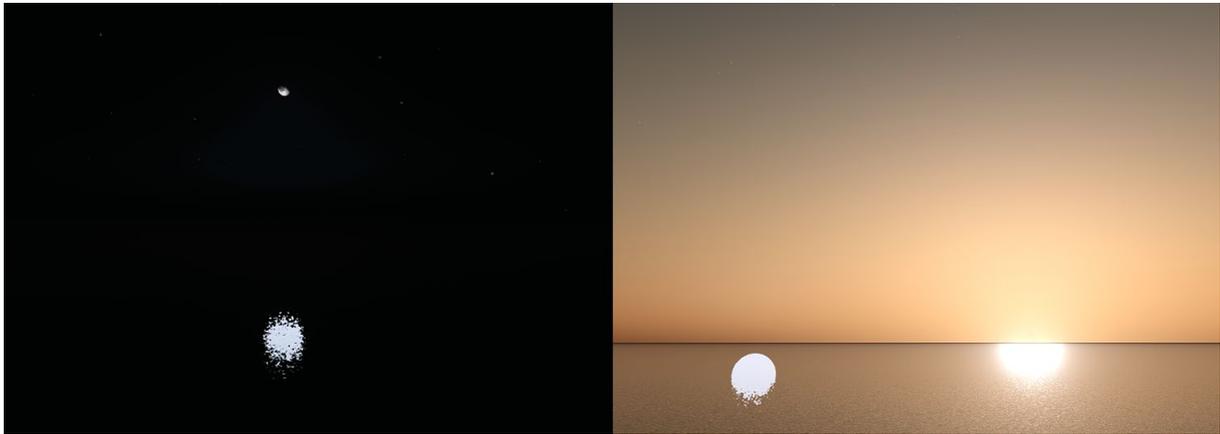


Figure 4. Previous moon specular reflections and invisible moon issue.

This was mainly the case due to the lack of a tidal-lock implementation and the usage of alpha blending since the moon was not always facing the player with the same side and the celestial bodies behind it were not obstructed. Besides that, the sun light direction for moon sphere shading was incorrect and did not fit to the sun position which additionally resulted in incorrect moon phase representations. Furthermore, the specular reflections of the moon light were visible on the water surface of the Nix ocean, however, the moon phase was not taken into account which resulted in an unrealistic representation of the specular reflection on the water when a new moon or half moon was present.

4.1.1 Theory

Similar to other objects in the application, the moon is a node that has been specified with its mesh and other properties in an XML file, which is generated with the help of the UNIGINE editor (see Appendix I). This node file contains a reference to a material XML file which dictates the uniform parameters, rendering passes with the respective shader file paths and other properties the material uses while rendering. Regarding the tidal-lock implementation, it was assumed that a simple look-at function call to orient the moon sphere towards the player or the world's origin should suffice to see the moon from the same side at all times. The issue

4. Professional product

with the incorrect moon phases and shading appeared to be fixable by uploading a sunlight direction to the shader that is represented by a vector that points from the sun to the moon instead of from the sun to earth as it was previously implemented. Since the moon phase also has an impact on the reflections of the externally rendered ocean, a constant buffer (cbuffer) to sync the parameters is required. The cbuffer is used to upload the relevant shading parameters from the CPU to the GPU once per frame to author the shading of the ocean mesh. However, due to the fact that the buffer is frequently updated and provides a low latency access, several limitations, such as a restricted memory layout, have to be accounted for (Microsoft Corp., 2020).

To tackle the last issue with the missing sky obstruction due to the specified alpha blending, it was necessary to color the unlit side of the moon in the same colors as the environment itself. The main difficulty was, however, that NAUTIS is not using a skybox cubemap to render the sky but instead uses a technique called Mie scattering, where several look-up textures (LUT) are used to approximate how the light scatters across the atmosphere in real-time, as seen in Figure 5 (NVIDIA Corp., 2005). These LUTs are set in the respective environment presets with which UNIGINE determines the color of the sky.

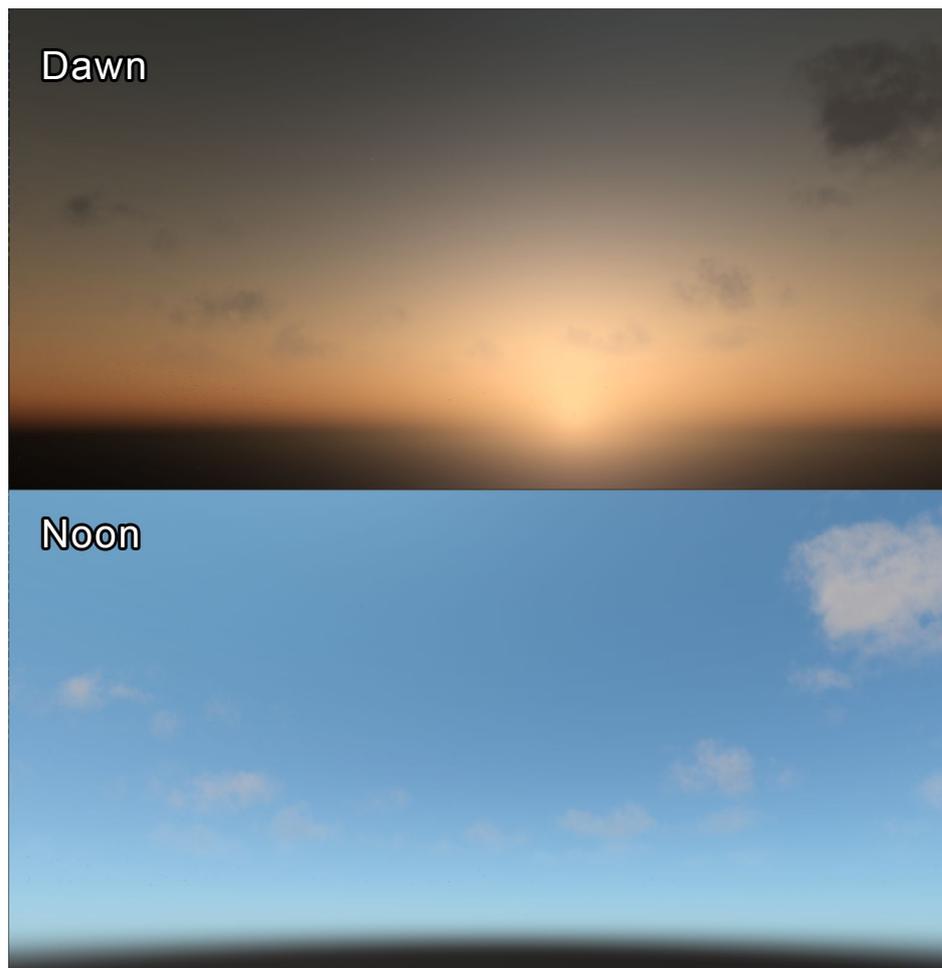


Figure 5. Atmospheric light scattering based on the Mie scattering approximation.

4. Professional product

4.1.2 Implementation

To fix the moon shading, the sunlight direction in the moon shader had to be corrected. Defining the moon phase intensity parameter that is responsible for controlling the moon's specular reflections needed to be done on the CPU to be able to send the parameter to both UNIGINE's and Nix' external shaders. The moon phase intensity parameter itself was defined as a float value which is based on the dot product between the sun and moon node positions and is mapped to the range 0 and 1. To make this value available in the ocean shader, it needs to be uploaded to a cbuffer that already contains various shading parameters. With the help of the parameter, it is possible to control the size, the strength and the scattering of the moons' specular reflections on the water. Additionally, it was necessary to add tweaking parameters for the specular reflection by introducing maximum and minimum values for full moon and new moon phases respectively to the configuration UI (see Figure 6). To allow for slightly smoother transitions and better tweaking of intermediate values for phases like the half moon, they are interpolated using a bias function as seen in Figure 7 (Wolfe, 2012).

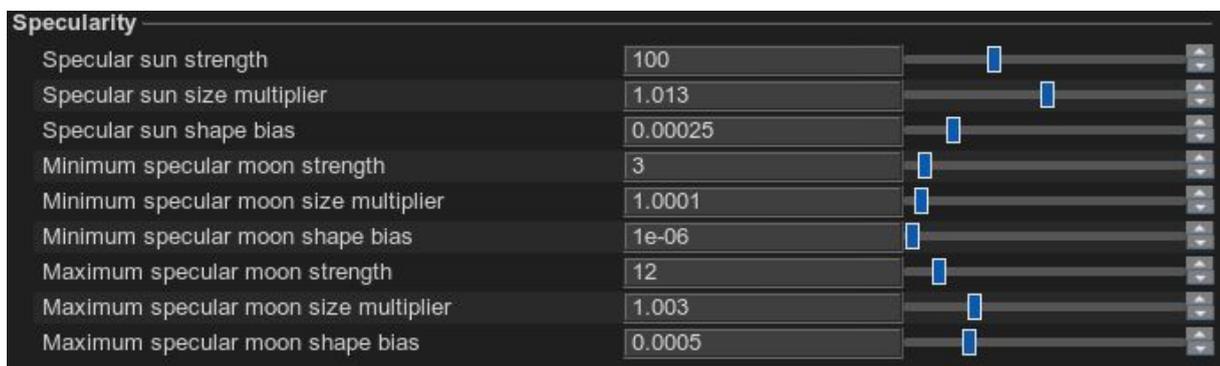


Figure 6. Moon specular reflection configuration parameters.

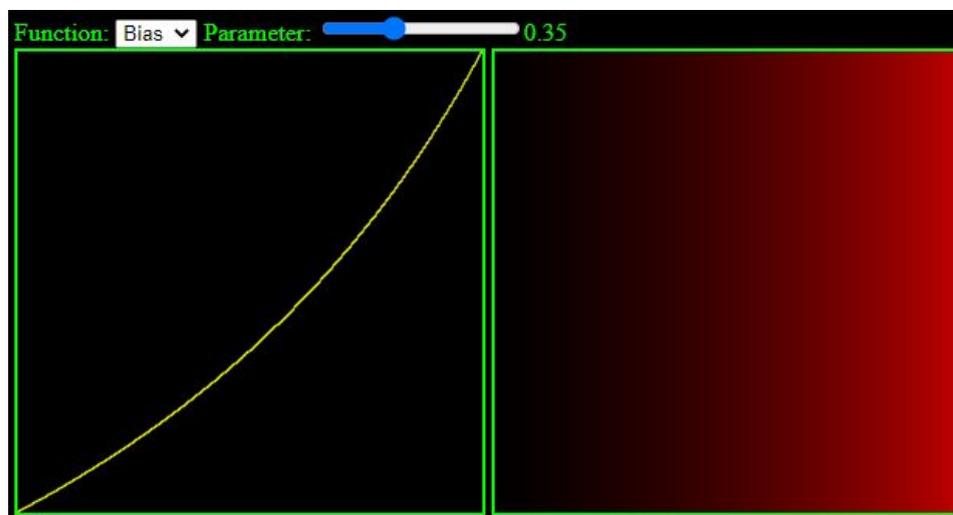


Figure 7. Moon phase bias transition visualization (Wolfe, 2012).

4. Professional product

The tidal-lock feature was first implemented by calling the look-at function on the transform of the moon sphere to make it look at the player at all times. However, this approach suffered from severe visual artifacts which looked like the shading of the moon was changing based on the time of day. The solution was to multiply the normals with the moon rotation matrix in the vertex shader as they had not been updated automatically when setting the moon's transformation matrix.

When addressing the lack of celestial body obstruction due to the used transparency blending on the moon, disabling alpha blending resulted in the unlit side ending up completely black while the lit side remained the same as before. While this was not an issue for nighttime scenes due to the dark sky, it was an apparent issue during the day. This problem was solved by applying the Mie scattering approximation on the unlit side, as discussed in the moon visuals theory.

4.1.3 Evaluation

Due to some restrictions in the engine, the company implemented various fixes and workarounds into UNIGINE's source code which made it unclear whether or not certain behaviours were still part of the intended functionalities. An example for this is how the tidal-lock implementation had to be handled since no explanation that justifies the behaviour of the moon sphere's normals was encountered. Instead, the moon transform should have been uploaded to the shader with the correct orientation without any extra work.

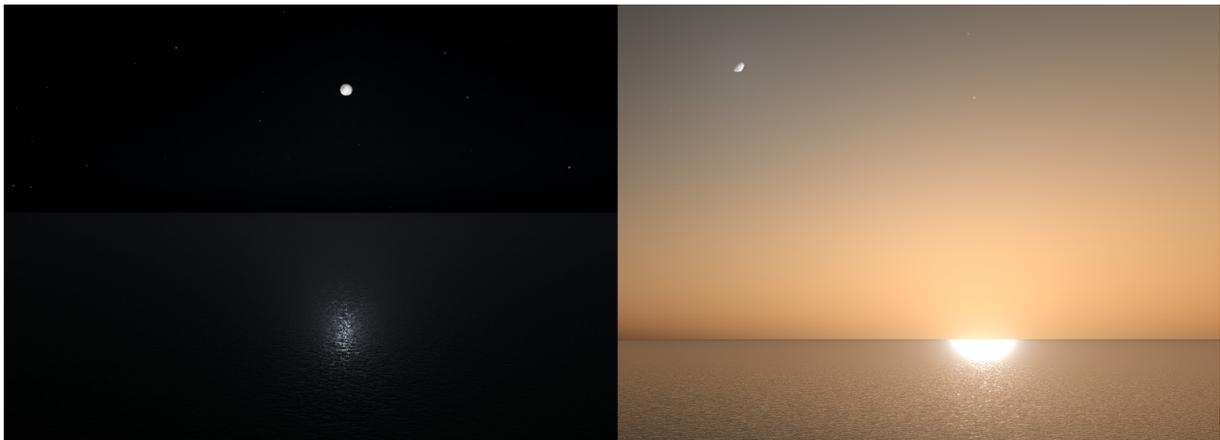


Figure 8. Improved moon specular water reflections and correct moon shading by the sun.

In order to test the viability of the provided moon visuals, the results (see Figure 8 and Appendix II) were first compared to real world reference pictures and then shown to various colleagues which generally agreed on the proposed implementation. Furthermore, a company internal survey (see Appendix III Figure 1 and Appendix III Figure 2) was conducted to have the participants rate the moon visuals before and after my work with a

4. Professional product

grade from one to nine. In this survey, the overall impression, the noticeability, the realism and the impact on the scene mood of the visuals were asked to be assessed. The average results seen in Figure 9 show that the overall rating, the realism and the impact in the scene were evaluated better for the new moon visuals when compared to the old ones. Even though the noticeability was rated slightly worse for the new visuals, the great increase in realism and the overall rating show that this is not necessarily a bad aspect as the old moon visuals seemed to be more noticeable due to their worse quality. This was additionally verified by the remarks submitted for the old visuals. In the comments of the new moon visuals, it was suggested to enhance the visuals by having the overall moon brightness change based on the moon phase as a full moon usually appears brighter than a half moon. Besides that, other potential visual improvements regarding the moon visuals are the addition of lunar eclipses and the implementation of the moon corona for night time scenes.

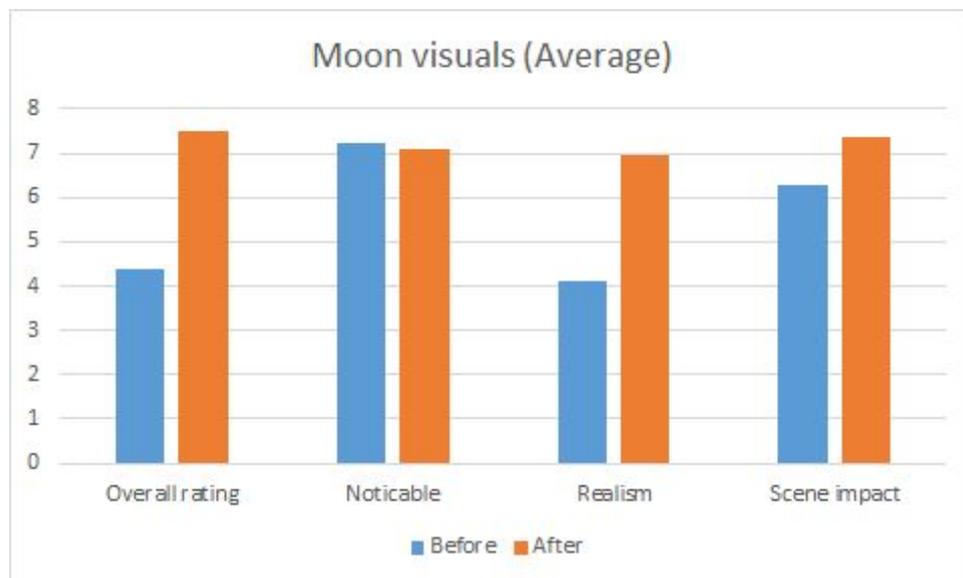


Figure 9. Company survey results of the moon visuals.

4. Professional product

4.2 Environment fog

With the update to UNIGINE 2.9, the fog scattering calculations were adjusted for all objects that natively participate in its rendering pipeline. One of the resulting issues was that the fog on the ocean surface consisted of a black ring when the overall visibility in the scene was really low (see Figure 10, bottom image).

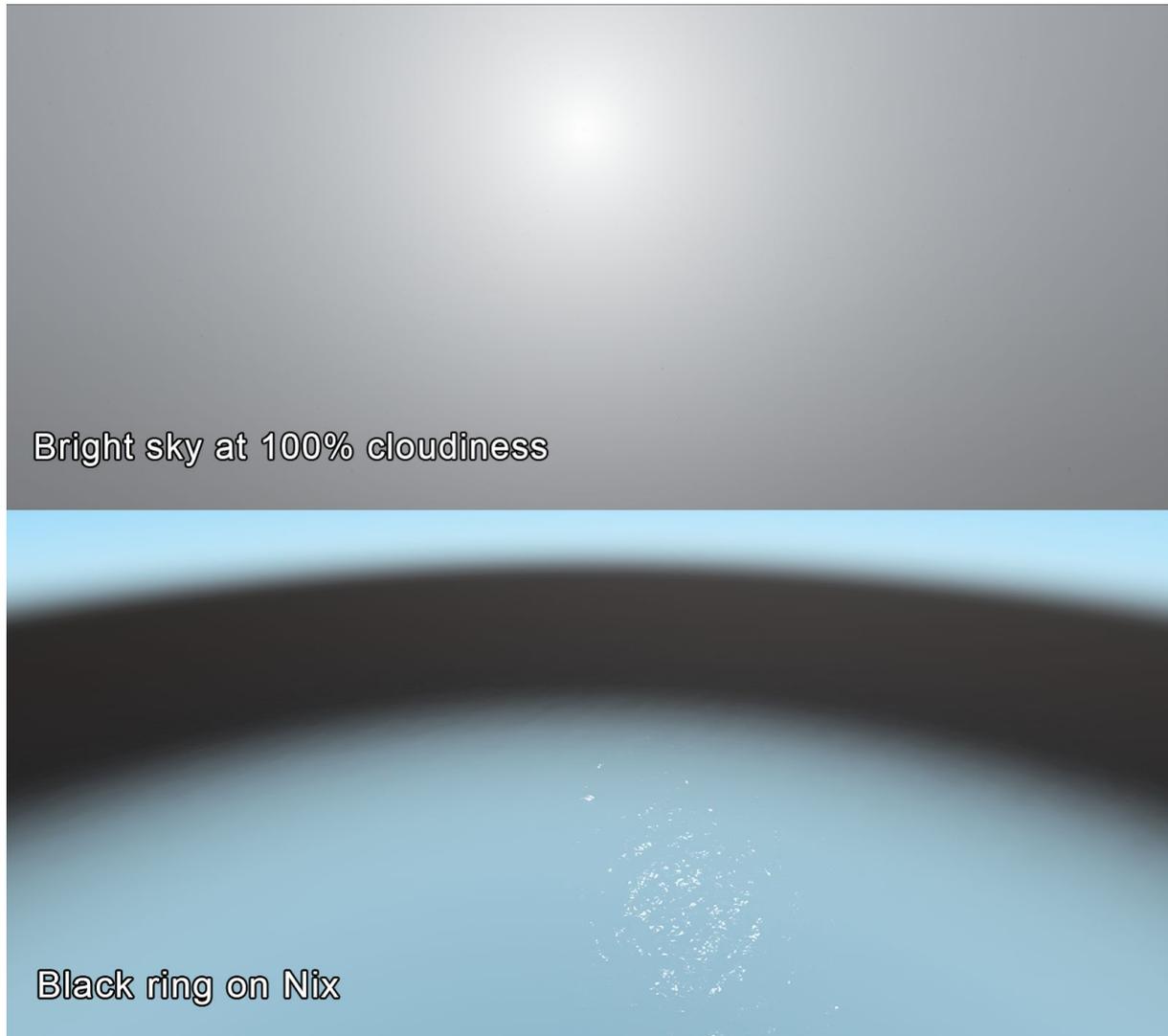


Figure 10. Bright sky and black ring issue on the ocean surface.

Since the Nix ocean is rendered separately, manual adjustments to its shader calculations were necessary to match its surface fog representation to the other objects that are rendered. Furthermore, while the ocean shaders are implemented in HLSL directly, the majority of the UNIGINE core shaders are written in the OpenGL and DirectX wrapper shader language UUSL. Because of that, Nix required certain important effect calculation functions to be copied from the engine's core shaders and translated into HLSL to get rid of some issues. Besides that, in the UNIGINE 2.5 implementation of NAUTIS, the environment color

4. Professional product

was determined by blending a sunny and a cloudy environment preset together based on the visibility in the scene. However, since the cloud system has changed to have more impact in the environment in UNIGINE 2.9, both the visibility and cloudiness had to be respected. The last fog related issue that had to be worked on, was to investigate an issue where the sky would get brighter instead of darker when a scene was rendered with a high cloud density and a low visibility.

4.2.1 Theory

In UNIGINE, the environmental fog is approximated by using Mie scattering and environment preset LUTs similar to how it was discussed during the moon visuals. Due to that, the objects affected by heavy fog blend perfectly into the sky color. The most important parameters to control this scattering effect are the fog density and the maximum fog (or visibility) distance (UNIGINE Corp., 2019). In this implementation, the maximum fog distance authors how far objects have to be away from the camera to have them fully occupied by fog. The fog density, on the other hand, is used to specify how heavy objects should be affected by the fog if they did not exceed the maximum fog distance. However, since the fog scattering is not constant, the fog naturally gets denser the closer objects are to the maximum visibility distance.

Since the camera's far clipping plane and the maximum fog distance are usually kept at the same distance, all objects that are further away are also automatically not rendered due to the camera's frustum culling. A camera's frustum is defined by a near and a far clipping plane where the resulting frustum cone represents the camera's view as seen in Figure 11 (Johnson, 2014). Frustum culling is an additional optimization that prevents sending vertices to the GPU if they do not intersect with the camera's view frustum (Epic Games Inc., 2020).

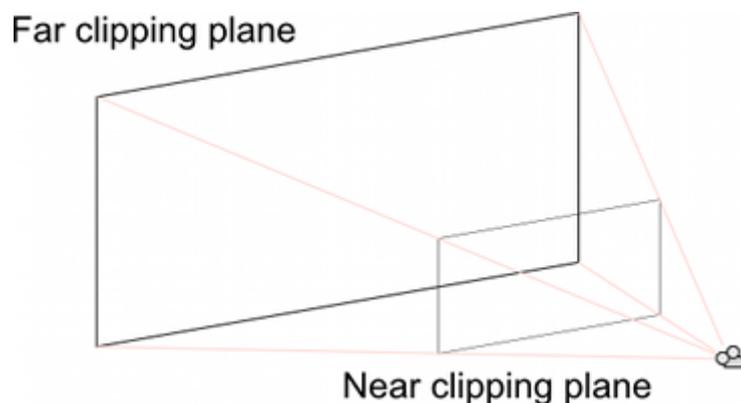


Figure 11. Visualization of the camera view frustum based on the clipping planes (Unity Technologies, 2020).

Another concept that is related to the environment fog, are the volumetric clouds as they were introduced as part of the environment system with the upgrade to UNIGINE 2.9. For

4. Professional product

most objects, only their surface is visualized by a vertex mesh with a solid or transparent material. If, however, the entire volume of an object should be visible or the object is too hard to model with geometric primitives, it is applicable to use volumetric objects (NVIDIA Corp., 2007). Furthermore, volumetric objects work well with translucent materials as they allow light to pass through and scatter within it (NVIDIA Corp., 2007). Approximating the light scattering within a translucent object is also referred to as subsurface scattering (Pixar, 2013). Examples for typical volumetric objects are smoke, clouds, fog or light shafts (see Figure 12). In the case of NAUTIS, volumetric clouds contribute a lot to the overall environment illumination as the light is scattered when it passes through the clouds. Since volumetric clouds are treated as normal 3D objects, they are additionally affected by the camera's frustum culling and other visual effects.

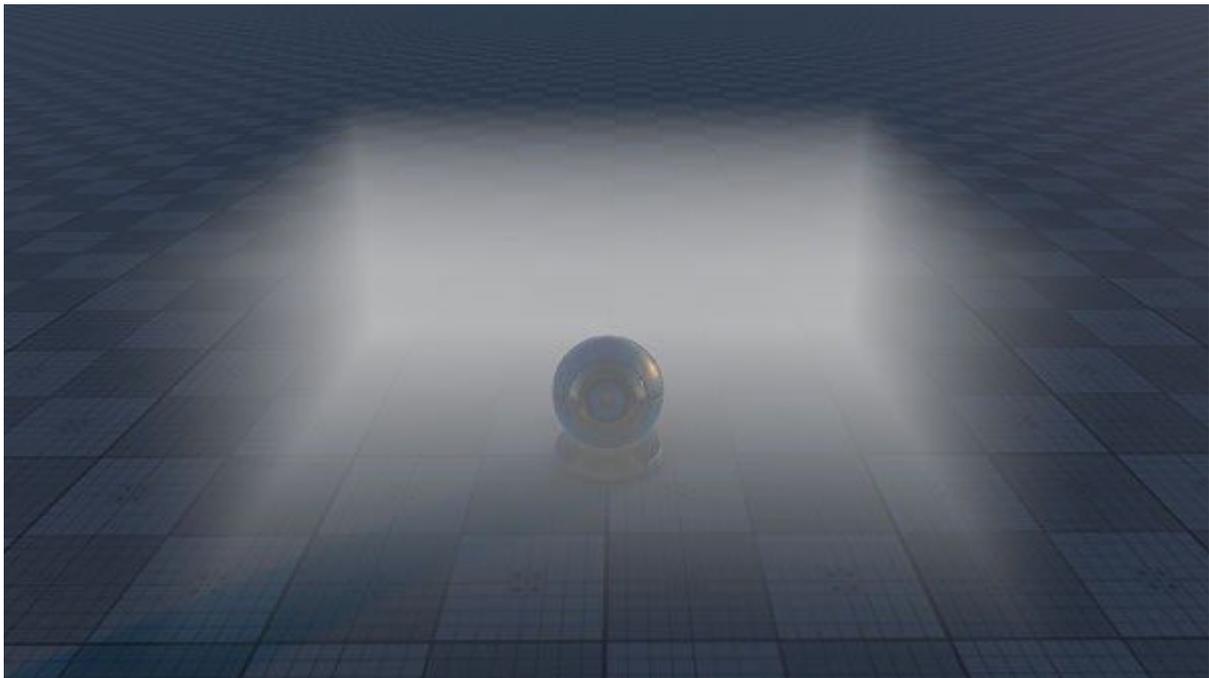


Figure 12. Example of a fog based volumetric object (UNIGINE Corp., 2019).

4.2.2 Implementation

The first goal was to update the UNIGINE function database that Nix shaders are using to compute the surface fog. This was achieved by comparing the old database to the new core shader implementation and translating it to HLSL. However, on some occasions the engine had introduced several other shading parameters, such as the sky intensity, that were unknown to the Nix shaders at that point in time. Because of that, the cbuffer containing the shading parameters for the ocean shaders had to be updated to include these values. Once Nix got access to both the missing parameters and the updated fog scattering functions, the fog still did not blend well into the horizon as it did in UNIGINE 2.5. Additionally, the black

4. Professional product

ring was still present on the ocean surface. Since the fog scattering functions were implemented correctly, the problems had to emerge from one of the parameters passed to the functions. In fact, some of them were scaled or otherwise modified with hardcoded values while having no explanations in the related comments. The main issue was that a scaled camera height parameter was used to sample from the environment LUTs which resulted in an incorrect fog color. Passing the native values without any modification to the fog scattering function, though, resulted in a proper transition between the ocean and the horizon and also addressed the black ring issue.

Afterwards, the blending between the sunny and cloudy environment presets based on the visibility distance and the cloudiness were taken care of. There, the initial problem was that neither of the environment controller classes on the C++ side knew both of the required parameters. Because of that, a general preset blending function had to be implemented into a suitable controller class which was exposed to UNIGINE Script with the help of the API interface, as all environmental parameters are known within Script. This function was then used to determine the blend factor, the blend threshold and the blend weight based on the current scene conditions, so that an appropriate environment color could be applied. Moreover, Nix' fragment shader was adjusted to make certain effects, such as sunlight or moonlight specular reflections, less intense at a high cloudiness or a low visibility setting.

When working on the last issue where the sky would get unusually bright at a low visibility distance and a high cloudiness, various unsuccessful implementation attempts were necessary to investigate the cause for the issue. The problem was caused by UNIGINE's core fog implementation, which culled the clouds away when a low visibility range was used. Because of that, the clouds were no longer able to contribute to the light scattering in the scene. After discussing these findings during a team meeting, a post in the UNIGINE forum was opened to have the engine developers help out. There, they suggested implementing the fog by keeping the visibility distance to the camera's maximum far plane at all times while only modifying the fog density. In the previously used implementation, both parameters were adjusted constantly. However, since the main intention of the fog in NAUTIS is to hide objects that exceed a certain distance threshold, it was necessary to come up with a prototype based on the forum suggestions that results in extremely dense fog after the distance threshold. To do so, the calculations used for the fog approximation had to be reverse engineered while assuming a constant fog distance. While this prototype implementation allowed the clouds to contribute to the overall light scattering at all times on one hand, it resulted in very heavy fog contribution on ships at low visibility settings on the other hand (see Figure 13, top image).

4. Professional product

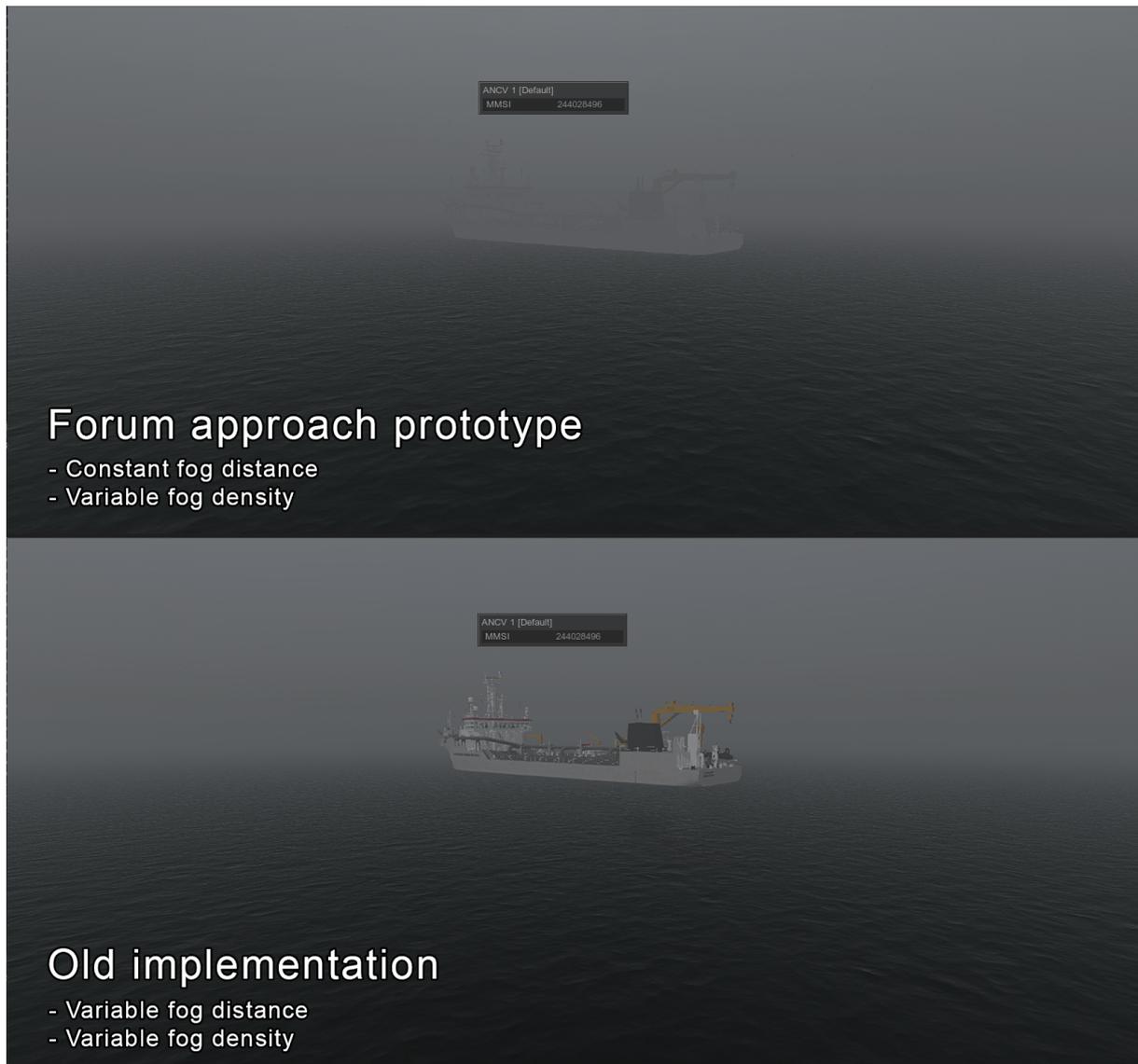


Figure 13. Fog implementation visual comparison.

This also meant that the trainee simulation view of the ship would purely be occupied by the fog color which was not desired. Since both fog implementations had their advantages, a comparison of the new fog prototype and the old fog implementation were presented to the project team and VSTEP's naval expert. As a result, we decided to keep the old version (see Figure 13, bottom image), which only contained the new environment present blending and the fixes for the black ring issue, as its advantages outweigh the advantages of the newly proposed forum approach prototype. After informing the UNIGINE developers about our findings and requesting an adjustment of their environmental system, they confirmed that changes to the cloud and fog system are planned for a future update. On top of that, all findings were documented in the respective DevOps work item entry so that it can be picked up again in the future.

4. Professional product

4.2.3 Evaluation

Since most of the fog issues that were worked on were not present when using UNIGINE 2.5, the new UNIGINE 2.9 implementations were mostly compared to the old ones as they have been used in several client submissions already. Aside from that, the fog also had to fulfil the condition to hide objects that are further away than the maximum visibility distance. This was verified with the help of a built-in tool in NAUTIS that allows the user to see a map overview of the currently loaded scenario and measure distances between two points. Moreover, similar to the moon visual task, the visuals (see Figure 14) were constantly discussed with the team so that feedback could be applied before finalizing the task.

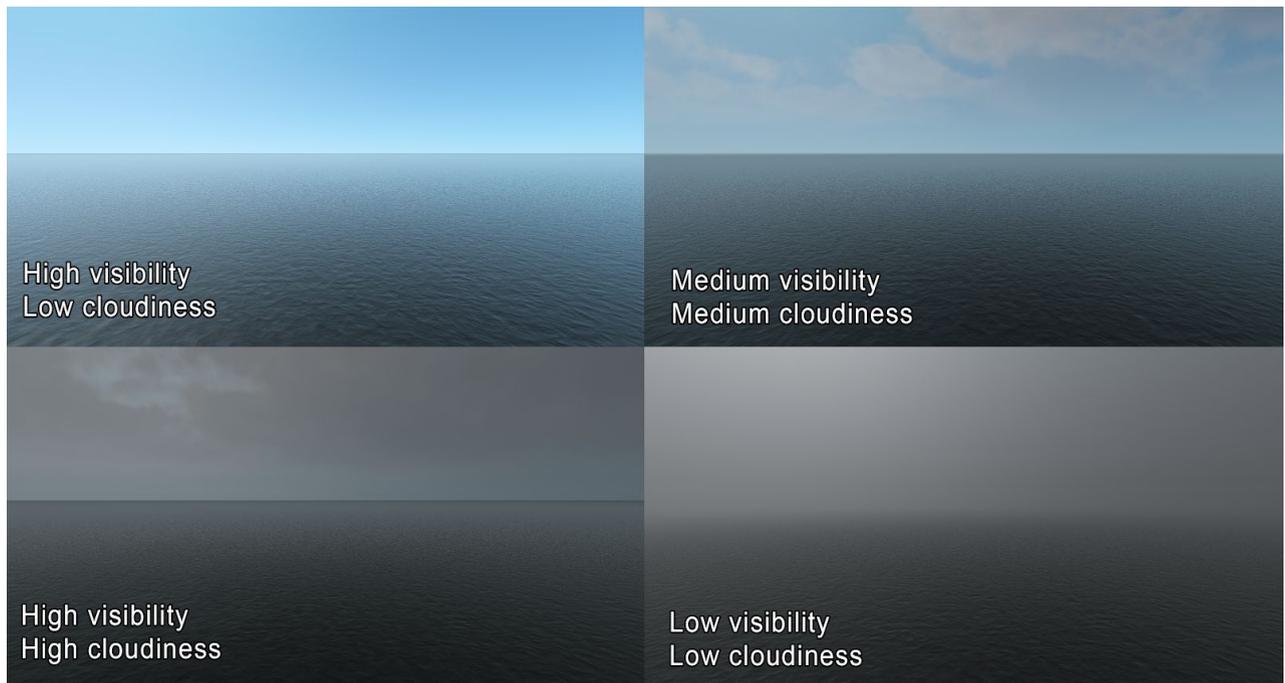


Figure 14. Fog visual results.

After creating a pull request with the fixes on DevOps, it was additionally reviewed by a team member to give feedback before the fixes were merged into the development branch in the repository. Though, as the project team and the naval expert decided that the old fog implementation was preferred over the proposed prototype due to its heavy disadvantages, it was not included in the pull request. Due to that, the bug with the bright sky at low visibility is still present in the current version of NAUTIS and will be attempted to be fixed when an update to the UNIGINE environment system is released. Other than that, the changes to the environment system were agreed on (see Appendix II). Similar to the evaluation of the moon visuals, a company internal survey (see Appendix III Figure 3 and Appendix III Figure 4) was conducted to assess the environment fog visuals before and after my work with a grade from one to nine based on the same criteria as for the moon. The average results presented in

4. Professional product

Figure 15 show that the proposed fog adjustments were generally perceived as improvements in all aspects by the survey participants. Moreover, additional comments have shown that the bright sky bug, which could not be addressed, is the main issue in the new fog implementation.

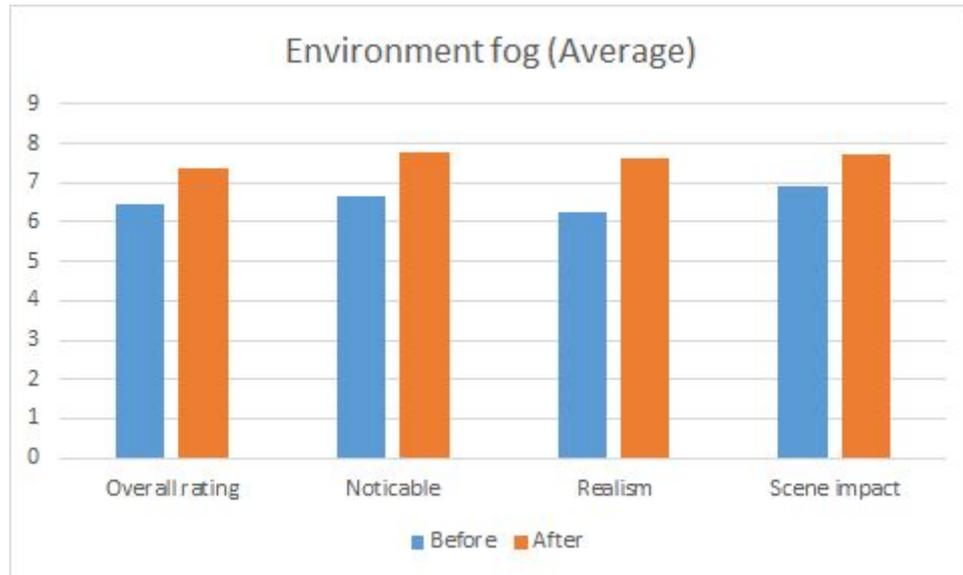


Figure 15. Average company survey results of the environment fog.

4. Professional product

4.3 Warfare particle effects

The work on this task was explicitly requested by a client of VSTEP, as they wanted to have working warfare effects for both day- and nighttime in their simulator. Additionally, the client is going to use NAUTIS mainly as a visualizer for their own simulation which means that the proposed particle effect implementations should barely have any logic on their own, since it is all provided by the client's simulator. Because of that, it is only necessary to spawn effects based on the requested effect type, its position in the world and other parameters. To test the particles ourselves, however, a weapon system was implemented for the warfare sounds by a team member, which could also be used to spawn custom particles whenever a ship is firing its weapon. Though, to make those effects visible at night, they have to be altered in such a way, that they actually illuminate the geometry around them when an explosion related effect is used (see Figure 16). Aside from that, options to reflect particles on the ocean surface had to be investigated for future development.



Figure 16. Warfare particle that lacks illumination of surrounding geometry.

4.3.1 Theory

A particle system object in UNIGINE consists of a single particle emitter node which is used to control certain behaviour of the emitted particles, such as their velocity or their growth over time (UNIGINE Corp., 2019). Its spawned particles can react to physical forces and can act independently from its emitter node. Besides that, particles consist of point masses instead of a vertex mesh which allows them to change their form over time (UNIGINE Corp., 2019). Furthermore, a parent particle node can be used to control and sync all of its children emitters.

4. Professional product

To fake the surface illumination effect, an approach is to combine the usage of emissive textures and the Bloom post-processing effect. When emission is used, the affected parts of the object surface ignore the lighting calculations completely and only color it according to the emission texture (UNIGINE Corp., 2019). Combined with the Bloom post-processing effect, the bright areas in the scene are getting blurred which makes the glow even more pronounced. However, while this technique allows objects to have a glow effect on themselves, it does not illuminate any geometry around it as seen in Figure 17. An approach to have emissive materials contribute to the lighting is to bake them into an environment probe. Although, this can only be used for static objects and is therefore not applicable for particles (UNIGINE Corp., 2019).

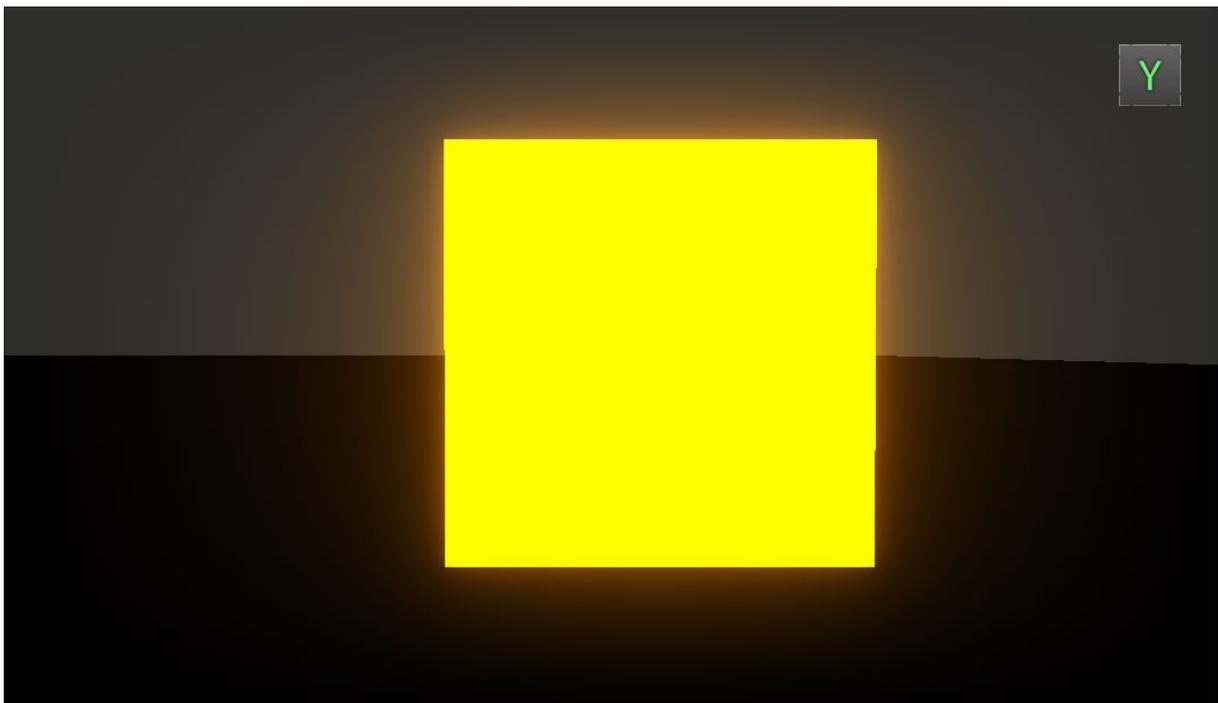


Figure 17. Bloom and emission effects used to fake self-glow of the plane.

Another option is to use a real-time omnidirectional light source and add a custom animation for it. While this approach allows to illuminate geometry in real-time, this also takes away some performance especially when forward rendered objects have to be shaded. Furthermore, enabling shadows for omni lights is even more hardware intensive, as the scene has to be rendered six times from different angles to generate a shadow cubemap that covers all areas affected by the omni light (UNIGINE Corp., 2019).

In order to have particles contribute to the reflection on the ocean surface, UNIGINE offers to either use their screen-space reflections (SSR) or planar reflections implementation (see Figure 18).

4. Professional product

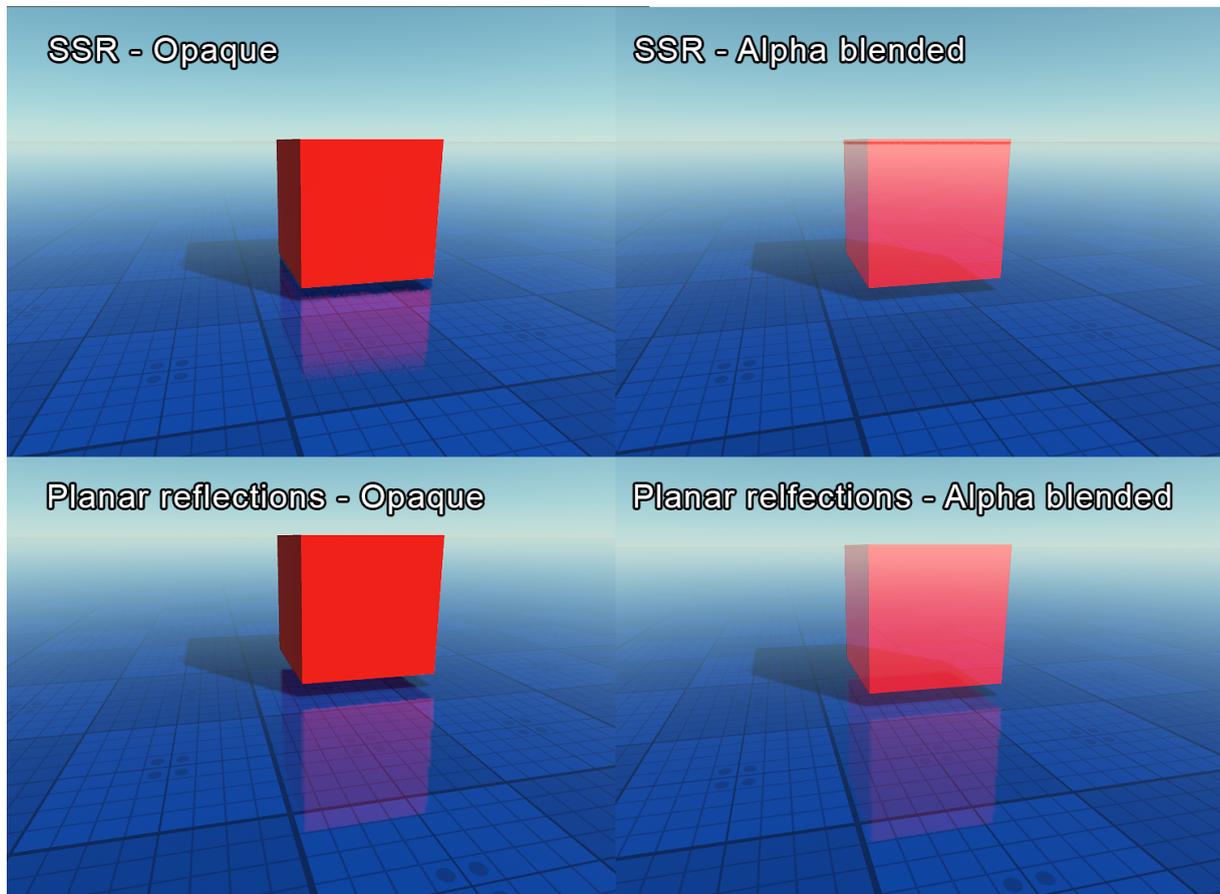


Figure 18. Reflection capabilities of screen-space reflections and planar reflections in UNIGINE.

SSR is an effect that approximates reflections on smooth surfaces by only using limited data that is currently available on the screen. Planar reflections on the other hand, render the entire scene from the perspective of the reflective surface to generate a reflection texture. Because of that, SSR usually costs less performance than planar reflections whereas the latter results in more precise reflections (Epic Games Inc., 2020). Though, since SSR generally does not work with transparent objects due to its dependency on the G-Buffer, planar reflection was the only applicable option. The downside, however, is that planar reflections only account for vertex meshes instead of point masses used in particle systems. To account for that, particle meshes have to be created to have emitters spawn mesh based particles (UNIGINE Corp., 2019).

4. Professional product

4.3.2 Implementation

Since this task was mainly focussed on the visuals of the weapon particles, the main work consisted of creating particle systems in the UNIGINE editor and tweaking them constantly based on real world references until they looked satisfactory.

Since the previously developed weapon system already took care of firing, shooting and other weapon behaviour, only the particle effects had to be managed by the implementation. To trigger the particle emission, it was necessary to hook to a specific event in the weapon system which is triggered when the respective weapon is fired. As an initial approach, the particles were implemented by introducing an object pool to reduce the amount of memory allocations. When the weapon was fired, it would either create a new particle emitter or reuse the oldest one if the pool was full. Although, after learning that the particles act independently from their emitter, it was clear that only one particle emitter per effect is necessary which is why the pooling implementation was removed again.

In order to have the particle effects illuminate surfaces around them, an omni light was attached to the weapon node and animated based on the extra parameters that were exposed to the weapon config. As the particles were initially only affected by real-time lighting of directional lights, their material settings had to be adjusted to allow for omni lights as well.

When the particle behaviour was tested at different fire rates, it was noticeable that using a huge fire rate value on the cannons resulted in some visual artifacts which made it look like the particles were spawned with a long delay, even though multiple ones were expected to be spawned at once. To address this issue, the spawn threshold and the maximum amount of particles of the emitters had to be increased. While this change made the particles look better at a high fire rate, the emitters were still limited to spawn a maximum of one particle per frame. Because of that, after meeting the threshold of about 600 rounds per minute (RPM), the particle effects started to look similar to when significantly higher fire rate values are used.

To make it easy for visual effects artists to use the weapon particle system, a configuration preset was created for different cannons that can be used as a starting point. In the config file itself, only the light animation parameters have to be adjusted and up to three particle emitters for the respective weapon have to be selected.

4. Professional product

4.3.3 Evaluation

Similar to the other tasks, the intermediate and final results were frequently shown to the project team to gather their feedback and iterate on the implementation. To prove their viability in dark environments, demos at noon, dawn and midnight (see Appendix II) were presented to see the impact on the scene illumination under different conditions. On one hand, the overall team's feedback on the particles was positive, but on the other hand, it was noticed that the omni light source sometimes illuminates surfaces that should be occluded by other geometry (see Figure 19).



Figure 19. Particle lighting passing geometry and illuminating unwanted areas.

To tackle this issue, shadows for the light have to be enabled which would result in significant performance losses when a multitude of weapon particles are used at the same time. Alternatively, the size of the light source could be decreased to illuminate a smaller area and therefore make it less likely to affect a lot geometry. In any case, the team agreed that the implementation is sufficient.

Since this feature was requested by a client of VSTEP and is going to be included in the next version of update of NAUTIS, the QA department had to run several tests based on the given requirements to check its validity. While they were also generally happy with the visuals, they had a few suggestions to improve the realism of the effects. For instance, they recommended making the smoke particles more grey instead of black as black smoke is only characteristic for gunpowder based weapons whereas the test ship uses gas powered weapons.

4. Professional product

A potential improvement for the future is to create particle meshes to have them contribute to the planar reflections on the ocean surface. This would require no additional work by a programmer as a VFX artist could tweak or even create entirely new particle systems in the UNIGINE editor and select them in the weapon config (see Figure 20).

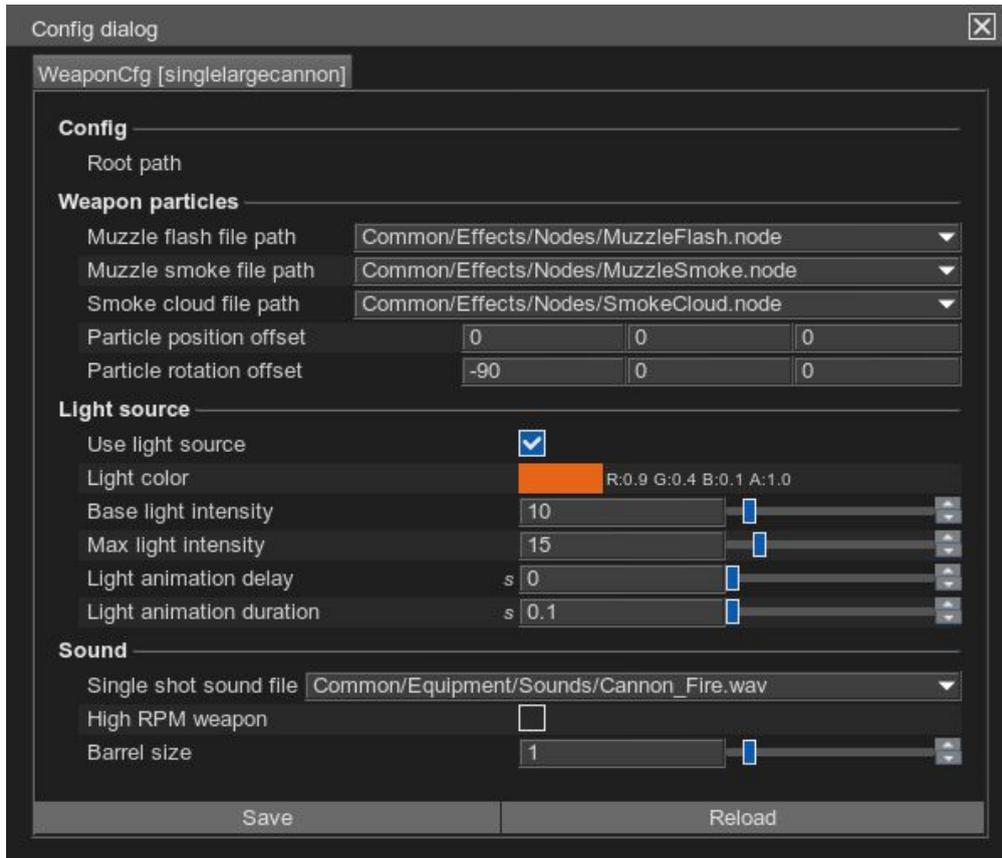


Figure 20. Options to adjust the particle effects and light animation.

In addition to that, a company internal survey (see Appendix III Figure 5 and Appendix III Figure 6) was conducted to rate the warfare particles with two different configurations where one uses a light source and the other does not. Similar to the evaluation of the moon visuals and the environment fog, the assessment criteria were the overall impression of the visuals, the noticeability, the realism and their impact on the mood of the scene. As seen in the results (see Figure 21), enabling the use of a light source for the particle effects increases the visual rating in all assessed aspects. The main remark given was that the particles' smoke color is too dark as previously discussed in the QA feedback. Besides that, the only small remarks were given on the light duration or light color which can be addressed by altering the configuration values based on the given use-case of a ship vessel.

4. Professional product

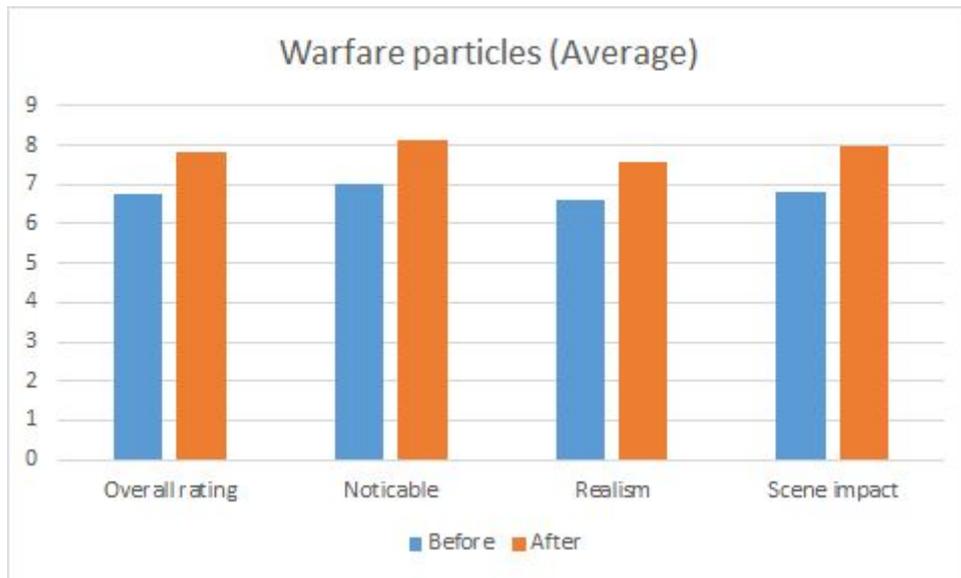


Figure 21. Average company survey results of the warfare particles.

4. Professional product

4.4 Nix light system

In the current light system of the Nix ocean, only custom searchlights can be used to dynamically illuminate the water surface. This is achieved by filling an array of searchlight structs, uploading it to the water fragment shader and individually calculating the light contribution for each water pixel, as it is known from traditional forward shading. However, this implementation does not account for lights being obstructed by other geometry and does not generally allow for many different light sources due to the performance costs of the individual calculations. To account for these issues, a dynamic light optimization technique for forward rendered objects is desired to be implemented into NAUTIS. This optimization should mainly be tested and applied in regards to the illuminating warfare particle effects so that it can be used in other fields as well in the future.

Due to time constraints, it was not possible to implement an approach to improve the lighting system for Nix. However, since the research part has already been carried out, the theory behind the light system optimization approach is already given.

4.4.1 Theory

UNIGINE already provides a variety of options to optimize the lighting in a scene, although some of these optimizations, such as using static lights, are not applicable in NAUTIS due to its dynamic simulations. For dynamic lighting, the most relevant optimization techniques in the engine are voxel probes, interleaved lights rendering and tile rendering. A voxel probe is essentially a light source that applies physically correct lighting to all objects inside it while also providing diffuse reflections (UNIGINE Corp., 2019). However, since a voxel probe uses prebaked 3D lighting maps, it is only applicable to use them when lighting dynamic objects within a static area in the scene. When using interleaved lights rendering, only the dynamic lighting for opaque objects that participate in the deferred rendering pipeline is optimized (UNIGINE Corp., 2019). This is achieved by only calculating the lighting at $\frac{1}{2}$ or $\frac{1}{4}$ of the screen resolution and reconstructing the remaining lighting data based on the previously rendered frames. Because of that, this technique needs a warm-up time and a high framerate of 60 or more frames per second to produce the best results. Another lighting optimization that is used by UNIGINE is the tile rendering technique for omnidirectional lights without shadows, which allows a great number of omni lights to be rendered at once (UNIGINE Corp., 2019). When the optimization is performed, the light sources are grouped together and rendered in batches in order to reduce the amount of draw calls which therefore reduces the performance losses. However, once again this optimization is only present in the deferred

4. Professional product

rendering pipeline. Because of that, none of the presented light optimizations can be applied to the Nix ocean as it is present in the forward rendering pipeline.

As the ocean occupies a big portion of the scene most of the time, it is important to optimize the lighting for it as well as it would otherwise result in a bottleneck if too many lights are used. To tackle this issue, the clustered shading optimization can be implemented for forward passes, which is also known as Forward+ as seen in Figure 22 (Ortiz, 2018). In this approach, it is required to introduce a depth pre-pass which is responsible for rendering the depth of all transparent objects into a texture buffer, similar to how data is rendered into the G-Buffer during the deferred rendering pipeline. On top of that, the frustum of the camera is subdivided into smaller 3D clusters which are used to check for intersections with the light sources in a compute shader to store data about which areas of the scene are affected by which light source in an array (Olsson et al., 2012). With the help of this data, it is possible to perform light culling by only calculating the lighting of a pixel based on the light sources that actually contribute to its illumination, instead of calculating the lighting based on all light sources similar to how it is done in traditional deferred and forward shading (Ortiz, 2018).

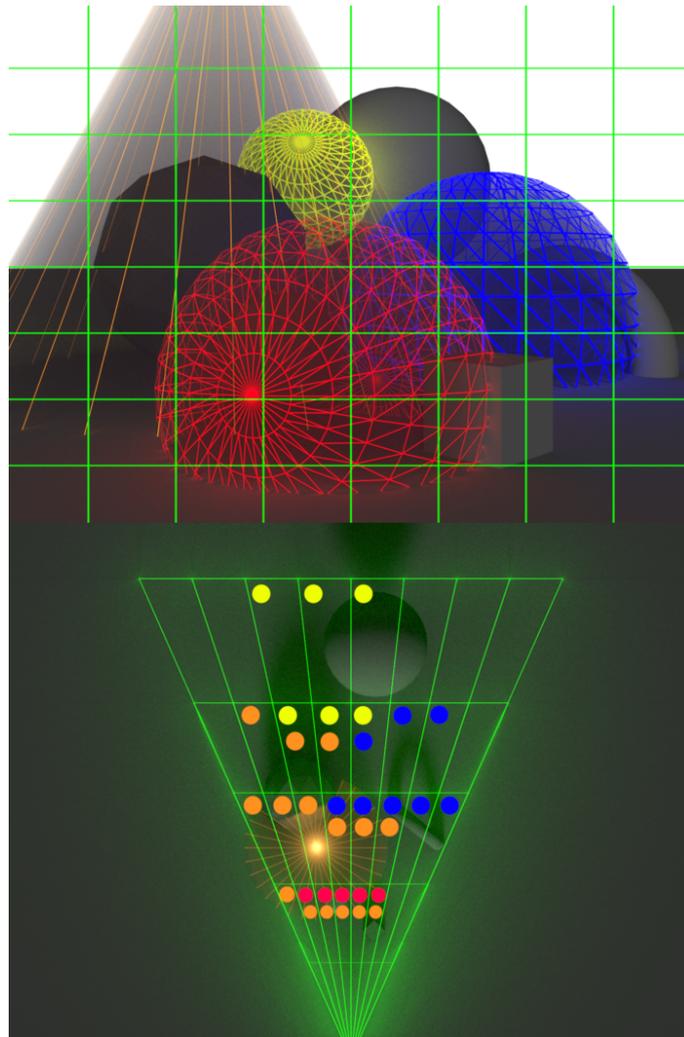


Figure 22. Visualization of Forward+ shading approach (Ortiz, 2014).

5. Conclusion

When it is desired to integrate custom effects into an application based on UNIGINE, the engine offers a multitude of options to do so. Based on the requirements, developers can choose to introduce custom materials and shaders into the existing rendering pipeline or to use existing effect objects, such as particle systems or volume boxes. Additionally, external rendering pipelines can be merged into the engine's rendering pipeline with the drawback of only getting access to limited shading data. To construct visual effects that achieve the desired visual quality and realism, real-life references, such as videos and photos, should be utilized and frequent feedback from experts should be acquired to iterate on the visuals. When effects for dark scenes are worked on, a special emphasis has to be put on the lighting and environment illumination as it draws more attention during the absence of sunlight. However, when using a large number of lights on many or big transparent surfaces, an additional forward rendering optimization technique, such as clustered forward shading, is recommended to be implemented into the engine's rendering pipeline to improve the performance.

While much theory was gained to propose improvements for NAUTIS, it is possible that different, and potentially more limited, implementation approaches have to be found in the future as UNIGINE no longer offers source code access for VSTEP by the end of 2020. Moreover, an in-depth insight from a dedicated computer graphics expert at VSTEP would have been desired to find proper solutions for environmental effect issues. Aside from that, a substantial amount of theory was necessary to understand most discussed rendering concepts due to their complexity. Though, even more knowledge on these topics could have led to other or potentially better implementations and proposals.

The main contributions of this project to the NAUTIS simulator involve the improvement of the moon shading by introducing tidal-lock and celestial body obstruction on its unlit side as well as the adjustment of the specular moonlight water reflections. Aside from that, the environment fog was improved by addressing various visual artifacts and by introducing a new environment present blending system that takes both the visibility distance and the cloudiness condition parameters into account. Other than that, a system to allow the usage of real time surface illumination on weapon particle effects was introduced to make these effects behave realistic in dark scenarios. The viability of these contributions was mainly verified by comparing the effects to real-life references and by presenting them to the project team or QA for further feedback until they were approved. On top of that, a company internal survey (see Appendix III) was conducted which also confirmed that the given implementations are improvements to the visuals.

5. Conclusion

Based on the feedback for these contributions, further improvements for them can be implemented in the future. These include the corona and a lunar eclipse for the moon, working cloud light scattering for the fog at low visibility distances and water reflections as well as visuals improvements for the warfare particles. Besides that, further research is recommended to determine if the clustered light shading optimization approach for Nix can be implemented for the deferred and forward rendering pipelines of the engine. As it was not possible during the corona crisis, the effect implementations should additionally be tested thoroughly with end users on the targeted simulator hardware to gather valuable feedback for future iterations.

When it comes to integrating custom environmental and nighttime effects into UNIGINE, the engine provides a broad variety of options to integrate custom effects by allowing the introduction of new rendering passes and offering existing effect solutions. Aside from that, allowing for accurate and performant dynamic lighting is most important for visual effects in dark scenes due to the great impact of the environment shading and individual light sources. Because of that, effects that contribute a lot to the overall environment lighting have to be improved and implemented for convincing nighttime scenes. On top of that, the previous discussions on the light system have shown that the most apparent bottleneck regarding the assignment scope is the dynamic illumination of many and large forward rendered surfaces by a multitude of light sources which is recommended to be addressed.

6. Bibliography

6. Bibliography

De Vries. J. (2014, June). Deferred Shading. Retrieved from <https://learnopengl.com/Advanced-Lighting/Deferred-Shading>

De Vries. J. (2014, June). Shaders. Retrieved from <https://learnopengl.com/Getting-started/Shaders>

Epic Games Inc. (2020). Planar Reflections. Retrieved from <https://docs.unrealengine.com/en-US/Engine/Rendering/LightingAndShadows/PlanarReflections/index.html#:~:text=Screen%20Space%20Reflections%20VS%20Planar%20Reflections,are%20also%20much%20less%20reliable.&text=Screen%20Space%20Reflections%3A%20The%20image,limitation%20of%20Screen%20Space%20Reflections>

Epic Games Inc. (2020). Visibility and Occlusion Culling. Retrieved from <https://docs.unrealengine.com/en-US/Engine/Rendering/VisibilityCulling/index.html>

Johnson, B. R. (2014, April). The Camera View Frustum. Retrieved from <http://courses.washington.edu/arch481/1.Tapestry%20Reader/2.The%20Camera/4.View%20Frustum.html>

Microsoft Corp. (2020). Shader Constants. Retrieved from <https://docs.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl-constants>

NVIDIA Corp. (2007, September). GPU Gems - Chapter 39. Volume Rendering Techniques. Retrieved from https://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch39.html

NVIDIA Corp. (2005, April). GPU Gems 2 - Chapter 16. Accurate Atmospheric Scattering. Retrieved from <https://developer.nvidia.com/gpugems/gpugems2/part-ii-shading-lighting-and-shadows/chapter-16-accurate-atmospheric-scattering#:~:text=16.2.&text=The%20two%20most%20common%20forms,Rayleigh%20scattering%20and%20Mie%20scattering.&text=Mie%20scattering%20is%20caused%20by,all%20wavelengths%20of%20light%20equally>

6. Bibliography

NVIDIA Corp. (2020). NVIDIA Nsight Graphics. Retrieved from

<https://developer.nvidia.com/nsight-graphics>

Olsson, O., Billeter, M., & Assarsson, U. (2012). Clustered Deferred and Forward Shading.

Retrieved from http://www.cse.chalmers.se/~uffe/clustered_shading_preprint.pdf

OpenGL Wiki. (2019, April 9). Geometry Shader. Retrieved from

http://www.khronos.org/opengl/wiki_opengl/index.php?title=Geometry_Shader&oldid=14512

OpenGL Wiki. (2019, April 9). Tessellation. Retrieved from

https://www.khronos.org/opengl/wiki_opengl/index.php?title=Tessellation&oldid=14629

Ortiz, A. (2018, December 21). A Primer On Efficient Rendering Algorithms & Clustered

Shading. Retrieved from <http://www.aortiz.me/2018/12/21/CG.html#tiled-shading--forward>

Pixar. (2013, February). Translucency and Subsurface Scattering. Retrieved from

https://renderman.pixar.com/resources/RenderMan_20/subsurface.html#introduction

UNIGINE Corp. (2019). 1. Basic Scene Objects and Coordinate System. Retrieved from

https://developer.unigine.com/en/docs/2.9/start/programmer/basics_objects?rlang=cpp

UNIGINE Corp. (2019). Environment - Haze Settings. Retrieved from

https://developer.unigine.com/en/docs/2.9/editor2/settings/render_settings/environment/?rlang=cpp#haze

UNIGINE Corp. (2019). Environment Probe - Adding Environment Probe. Retrieved from

<https://developer.unigine.com/en/docs/2.9/objects/lights/envprobe/#adding>

UNIGINE Corp. (2019). Interleaved Lights Rendering. Retrieved from

https://developer.unigine.com/en/docs/2.9/principles/render/interleaved_rendering/

UNIGINE Corp. (2019). Lights. Retrieved from

https://developer.unigine.com/en/docs/2.9/editor2/settings/render_settings/lights/

UNIGINE Corp. (2019). Lights Optimization. Retrieved from

<https://developer.unigine.com/en/docs/2.9/content/optimization/lights/>

6. Bibliography

UNIGINE Corp. (2019). Lights Optimization - Omni Light. Retrieved from https://developer.unigine.com/en/docs/2.9/content/optimization/lights/#omni_light

UNIGINE Corp. (2019). mesh_base - Emission. Retrieved from https://developer.unigine.com/docs/2.9/content/materials/library/mesh_base/#option_emission

UNIGINE Corp. (2019). Particle Systems. Retrieved from <https://developer.unigine.com/en/docs/2.9/objects/effects/particles/>

UNIGINE Corp. (2019). Particle Systems - Mesh-Based Particles. Retrieved from https://developer.unigine.com/en/docs/2.9/objects/effects/particles/#mesh_based_particles

UNIGINE Corp. (2019). Rendering Sequence - Deferred Pass for Opaque Objects - SSR. Retrieved from <https://developer.unigine.com/en/docs/2.9/principles/render/sequence/#ssr>

UNIGINE Corp. (2019). Rendering Sequence - Rendering Pipeline Overview. Retrieved from https://developer.unigine.com/en/docs/2.9/principles/render/sequence/#rendering_pipeline

UNIGINE Corp. (2019). Rendering Sequence - Transparent Objects Rendering. Retrieved from <https://developer.unigine.com/en/docs/2.9/principles/render/sequence/#transparent>

UNIGINE Corp. (2019). UUSL Compute Shaders. Retrieved from <https://developer.unigine.com/en/docs/2.9/code/uusl/compute?rlang=cpp>

UNIGINE Corp. (2019). volume_fog_base [Image]. Retrieved from https://developer.unigine.com/en/docs/2.9/content/materials/library/volume_fog_base/

UNIGINE Corp. (2019). Voxel Probe. Retrieved from <https://developer.unigine.com/en/docs/2.9/objects/lights/voxelprobe/?rlang=cpp>

Unity Technologies. (2020). Understanding the View Frustum [Image]. Retrieved from <https://docs.unity3d.com/Manual/UnderstandingFrustum.html>

6. Bibliography

Wolfe, A. (2012, September 24). Bias And Gain Are Your Friend. Retrieved from <https://blog.demofox.org/2012/09/24/bias-and-gain-are-your-friend/>

Wolfe, A. (2012, September 24). HTML5 Bias and Gain Visualizer [Screenshot]. Retrieved from <http://demofox.org/biasgain.html>

7. Appendices

Appendix I. Basic engine principles in UNIGINE

To understand how UNIGINE functions, it is important to note that the engine uses a right-handed coordinate system where the X, Y and Z axes are considered right, forward and up respectively. Similar to other game engines, the objects in a scene are represented by a node graph that takes care of all the parent-child relationships of the engine. These nodes can either be contained in the world file to load them during start-up or in separate node reference files to spawn nodes at runtime. Furthermore, most engine files, such as files for nodes and materials, are written in XML which makes it easy to understand their properties without having to look up their behaviour in the editor. In terms of rendering, the node specifies a reference to a mesh and a material file that should be used. Within the material file, it is then possible to adjust which shaders should be used during the specified render pass, what uniform values or textures the shaders should receive and more (see Figure 1). With the help of these materials, custom objects and UUSL shaders, which are wrappers for both DirectX and OpenGL shaders, are allowed to contribute to UNIGINE's render pipeline. Besides that, the engine already offers a wide range of optimized rendering techniques that can be enabled and authored in the UNIGINE editor.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <base_material version="2.9" name="moon_material" editable="1">
3    <!-- Enable the custom transparency preset -->
4    <blend src="src_alpha" dest="one_minus_src_alpha"/>
5    <options transparent="2"/>
6
7    <!-- States -->
8    <state name="ambient">1</state>
9
10   <!-- Forward (ambient) rendering shaders -->
11   <shader pass="ambient" node="object_mesh_static"
12     ambient="1"
13     vertex="Common/Materials/Sky/Moon/moon_vertex_ambient.vert"
14     fragment="Common/Materials/Sky/Moon/moon_fragment_ambient.frag"/>
15
16   <!-- Bindings -->
17   <bind node="object_mesh_dynamic" to="object_mesh_static"/>
18   <bind node="object_mesh_skinned" to="object_mesh_static"/>
19
20   <!-- Textures -->
21   <texture unit="0" name="albedo" anisotropy="1" shader="fragment" pass="ambient">Common/Textures/Sky/2k_moon.jpg</texture>
22   <texture unit="1" type="scattering_sky_lut" shader="fragment" pass="ambient"/>
23
24   <!-- Parameters -->
25   <parameter name="lightDirection" type="color">-1.0f 0.0f 0.0f 0.0f</parameter>
26   <parameter name="visibilityScale" type="slider">1.0f</parameter>
27
28 </base_material>

```

Figure 1. Example of a custom base material file.

7. Appendices

Appendix II. Professional product demo database

The link below opens a Google Drive folder that contains demos of the professional product in the form of images, GIF animations and videos.

Link:

<https://drive.google.com/drive/folders/1K1yZzni9orl92krfRPnNKYF4KuC2ho37?usp=sharing>

Appendix III. Visual effects company survey

Link: <https://forms.gle/gdLJSqqkixQ7xs797>

Moon visuals (version 1)

The moon visuals are mainly represented by the specular reflections on the water. This is the first presented version of the moon visuals.

Full moon water reflections (GIF Animation)



How do you rate the moon visuals? (1 = very bad, 9 = very good) *

Choose ▾

How noticeable do you rate the moon visuals? (1 = not noticeable, 9 = very noticeable) *

Choose ▾

How realistic do you rate the moon visuals? (1 = not realistic, 9 = very realistic) *

Choose ▾

Half moon water reflections (GIF Animation)



How do you rate the impact of the moon visuals in the overall mood of the scene? (1 = no impact, 9 = very impactful) *

Choose ▾

Do you have any other remarks for version 1 of the moon visuals?

Your answer _____

Moon shading



Figure 1. Moon visuals part 1 of the company internal evaluation survey.

7. Appendices

Moon visuals (version 2)

The moon visuals are mainly represented by the specular reflections on the water. This is the second presented version of the moon visuals.

Full moon water reflections (GIF Animation)



How do you rate the moon visuals? (1 = very bad, 9 = very good) *

Choose ▾

How noticeable do you rate the moon visuals? (1 = not noticeable, 9 = very noticeable) *

Choose ▾

How realistic do you rate the moon visuals? (1 = not realistic, 9 = very realistic) *

Choose ▾

Half moon water reflections (GIF Animation)



How do you rate the impact of the moon visuals in the overall mood of the scene? (1 = no impact, 9 = very impactful) *

Choose ▾

Do you have any other remarks for version 2 of the moon visuals?

Your answer _____

Moon shading

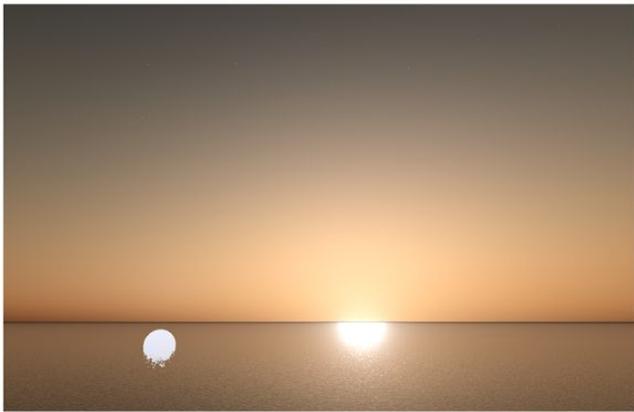


Figure 2. Moon visuals part 2 of the company internal evaluation survey.

7. Appendices

Environment fog (version 1)

The fog visuals are mainly represented by the environment blending. This is the first presented version of the environment fog visuals.

Medium visibility, medium cloudiness



Low visibility, high cloudiness



High visibility, low cloudiness



High visibility, high cloudiness



Low visibility, low cloudiness



How do you rate the environment fog? (1 = very bad, 9 = very good) *

Choose

How noticeable do you rate the fog effects? (1 = not noticeable, 9 = very noticeable) *

Choose

How realistic do you rate the fog effects? (1 = not realistic, 9 = very realistic) *

Choose

How do you rate the impact of the fog visuals in the overall mood of the scene? (1 = no impact, 9 = very impactful) *

Choose

Do you have any other remarks for version 1 of the environment fog?

Your answer

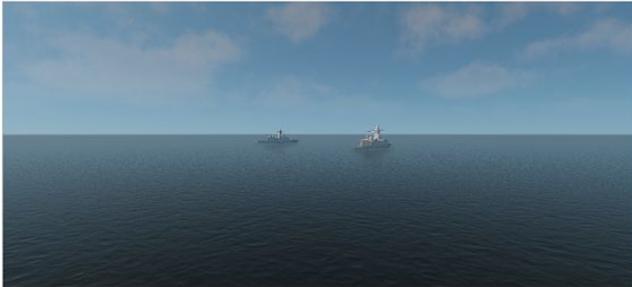
Figure 3. Environment fog part 1 of the company internal evaluation survey.

7. Appendices

Environment fog (version 2)

The fog visuals are mainly represented by the environment blending. This is the second presented version of the environment fog visuals.

Medium visibility, medium cloudiness



Low visibility, high cloudiness



High visibility, low cloudiness



High visibility, high cloudiness



Low visibility, low cloudiness



How do you rate the environment fog? (1 = very bad, 9 = very good) *

Choose

How noticeable do you rate the fog effects? (1 = not noticeable, 9 = very noticeable) *

Choose

How realistic do you rate the fog effects? (1 = not realistic, 9 = very realistic) *

Choose

How do you rate the impact of the fog visuals in the overall mood of the scene? (1 = no impact, 9 = very impactful) *

Choose

Do you have any other remarks for version 2 of the environment fog?

Your answer

Figure 4. Environment fog part 2 of the company internal evaluation survey.

7. Appendices

Warfare particles (version 1)

The warfare particles are mainly represented by a prototype particle effect for the main cannon a ship. This is the first presented version of the warfare particles.

Daytime, 60 RPM, Static (GIF Animation)



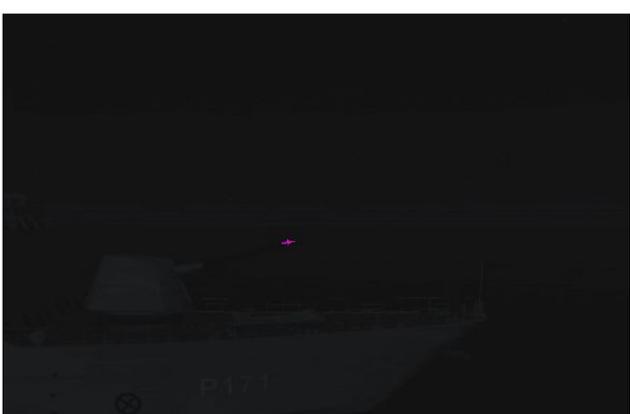
Screenshot of the particle effect during dawn



Dawn, 180 RPM, Moving (GIF Animation)



Nighttime, 1000 RPM, Static (GIF Animation)



How do you rate the warfare particles? (1 = very bad, 9 = very good) *

Choose

How noticable do you rate the warfare particles? (1 = not noticable, 9 = very noticable) *

Choose

How realistic do you rate the warfare particles? (1 = not realistic, 9 = very realistic) *

Choose

How do you rate the impact of the warfare particles in the overall mood of the presented scenes? (1 = no impact, 9 = very impactful) *

Choose

Do you have any other remarks for version 1 of the warfare particles?

Your answer

Figure 5. Warfare particles part 1 of the company internal evaluation survey.

7. Appendices

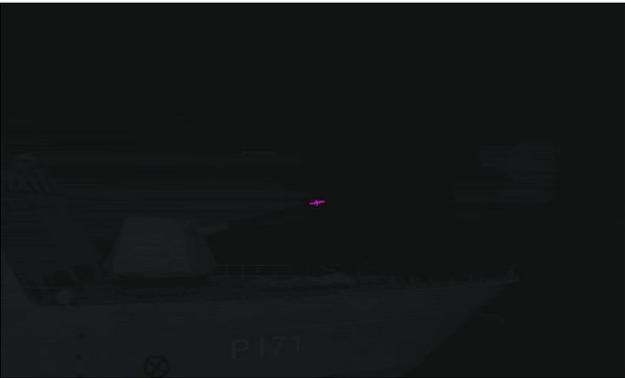
Warfare particles (version 2)	
<p>The warfare particles are mainly represented by a prototype particle effect for the main cannon a ship. This is the second presented version of the warfare particles.</p>	
<p>Daytime, 60 RPM, Static (GIF Animation)</p> 	<p>Screenshot of the particle effect during dawn</p> 
<p>Dawn, 180 RPM, Moving (GIF Animation)</p> 	<p>How do you rate the warfare particles? (1 = very bad, 9 = very good) *</p> <p>Choose <input type="button" value="v"/></p>
	<p>How noticeable do you rate the warfare particles? (1 = not noticeable, 9 = very noticeable) *</p> <p>Choose <input type="button" value="v"/></p>
	<p>How realistic do you rate the warfare particles? (1 = not realistic, 9 = very realistic) *</p> <p>Choose <input type="button" value="v"/></p>
<p>Nighttime, 1000 RPM, Static (GIF Animation)</p> 	<p>How do you rate the impact of the warfare particles in the overall mood of the presented scenes? (1 = no impact, 9 = very impactful) *</p> <p>Choose <input type="button" value="v"/></p>
	<p>Do you have any other remarks for version 2 of the warfare particles?</p> <p>Your answer <input type="text"/></p>

Figure 6. Warfare particles part 2 of the company internal evaluation survey.