02561 Computer Graphics – Project

Olav Nørgaard Olsen
s184195

19. December 2021

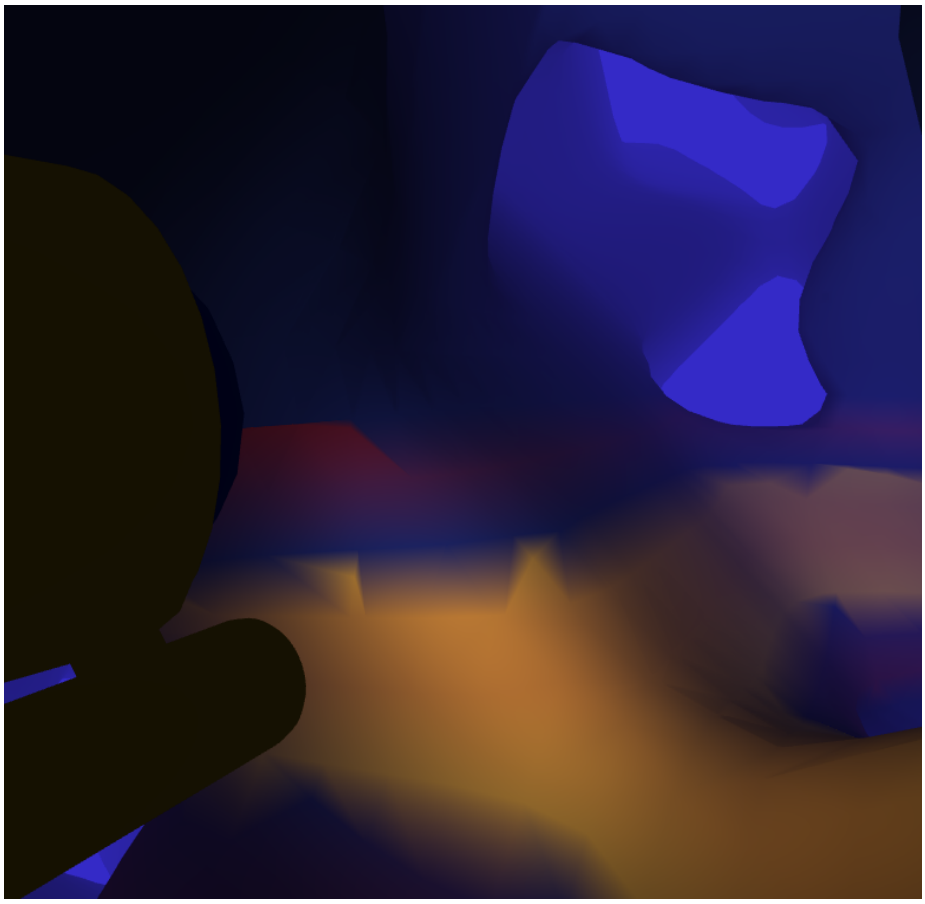


Figure 1: Screenshot from the project of the submarine in an underwater environment

Preface

All images and models are created by the author unless explicitly stated otherwise. The implementation of the project is located at: `http://www.student.dtu.dk/~s184195/02561_Computer_Graphics/Project/`. The exercises for the course are located at: `http://www.student.dtu.dk/~s184195/02561_Computer_Graphics/Project/`.

1 Introduction

By combining concepts from computer graphics together with procedural generation, one can create interactive demonstrations, allowing the user to experience various visual results.

This project experiments with three dimensional, procedural world generation, whilst applying multiple computer graphics techniques to render it. The user controls a submarine, navigating an unending underwater environment. Mounted to the submarine is 2 front-facing spotlights, illuminating the world.

2 Method

The following sections will give an overview of the features that have been used in the project as well as a brief description of their functionality.

2.1 Terrain generation

To enable the exploration of a vast underwater world, a large terrain must be added. The terrain is procedurally generated, instead of having a predefined mesh model of the terrain. This allows for an infinitely large world without having to repeat parts of a model over and over.

2.1.1 Terrain functions

The terrain is a binary terrain; either there is no terrain (air/water) or else there is. It is defined by a function $f(x, y, z)$, $f : \mathcal{R}^3 \rightarrow \mathcal{R}$, where positive values denote terrain and negative values of f gives no terrain. The surface of the terrain is located a $f(x, y, z) = 0$. The terrain is entirely dependent on how f is defined. By having f as a signed distance function, one can define the terrain to take the form of simple geometric shapes (see fig. 2).

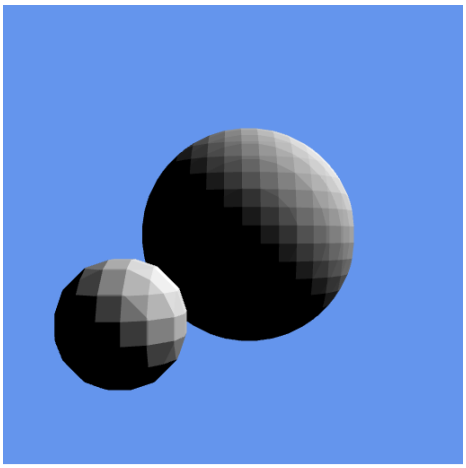


Figure 2: Terrain generated when f is defined as the (negative) shortest distance to the surface of 2 spheres.

However, for the intents of allowing an infinite terrain, f should be defined over all of $\mathcal{R}^3$ and should return coherent values for locations close to one another. To this end, Simplex noise is used.[6] It produces coherent pseudorandom values in points of a given dimensionality. Naturally, three dimensions are used her.

2.1.2 Marching cubes

The terrain specified by f must be converted into a triangle mesh. By sampling f at fixed intervals in the world, one gets a series of points that are either inside or outside the terrain. Consider a cube, whose vertices are the sampled points of f. Depending on which corners are inside or outside the terrain, one can determine how the triangles should be placed. This is the essence of the marching cubes algorithm.[3],[5] By combining multiple cubes together (and thus more sampled points) one can make arbitrarily complex shapes, only limited by the size of the cubes. Applying the marching cubes algorithm together with simplex noise enables for generating vast amounts of terrain.

2.1.3 Managing complexity

Allowing the user to navigate the world freely means that no bound can be placed on which parts of the terrain that must be generated from the start – and generating an infinitely large terrain is computationally intractable. As such, only some finite amount of terrain must be present at any given time. A system must be designed that can determine exactly which parts of the terrain that must be generated at a given time. The implementation section 3.2 will go into greater detail of its functionality.

2.2 Lighting & Shadows

To enhance the visual fidelity of the world, lighting must be added. The submarine features 2 spotlights. They are practically the same as point lights, but with a direction. Here, the intensity received by a point also depends on the angle between the normal and the direction of the point light. How the angle θ affects the intensity is called the attenuation $\alpha(\theta)$ of the light source. [1], chap. 6.2.4. The intensity of light at an object a distance d away is then

$$i = \frac{\alpha(\theta)I}{d^2} \tag{1}$$

where I is the intensity of the light source. Furthermore, each light must also be able to cast shadows onto the terrain. Concrete workings of the shadows and light are covered in section 3.3.

2.3 Distance fog

To put emphasis on distances in the water, a distance fog is added. When an object is shaded a distance d away from the camera, then distance fog blends between the computed color c_0 and the background color c_b according to d to produce the final color c. The closer the object is, the closer the resulting value will be to the original computed color, whereas farther objects will be closer to the background color, or:

$$c = c_0 + (c_b - c_0)f(d) \tag{2}$$

where $f_g(d)$ is a fog formula. The distance fog in the project uses the exponential fog formula [4]:

$$f_g(d) = 1 - \frac{1}{e^{(d \cdot k)^2}} \tag{3}$$

where k describes how quickly the fog strengthens.

2.4 Models

The submarine must be added to the world as well. A model has been created for this purpose (see fig. 3). The user must also be able to control the submarine; forward/backwards movement (along local z-axis), rotating to the sides (rotation around global up) and looking up/down (rotation around local x-axis). These operations are done utilizing quarternions to avoid risks of a gimbal lock. (Note that, in the implementation, the submarine is prevented from doing full rotation around the local x-axis, turning upside down. It is not a technical limitation, but disabled as it is not desirable for the user experience.)

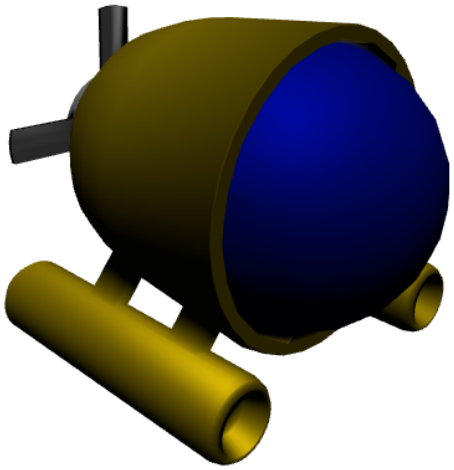


Figure 3: Submarine model.

3 Implementation

The following sections will describe different implementation details of the project, that required further details from the method section.

3.1 Overall Design

The program enters into an unending loop after initialization. First, the management of which parts of the terrain should be loaded is done. Then, user interaction is handled and, lastly, the world is rendered. An overview is given in fig. 4.

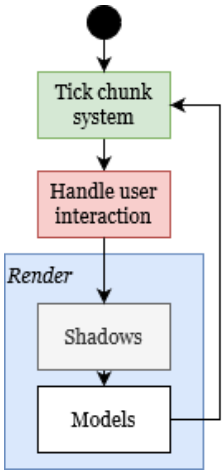


Figure 4: Interaction loop in project

Working with both multiple modelds and multiple shaders (discussed in section 3.3) requires a lot of bookkeeping to ensure the attributes and uniforms are set properly in the shader. Therefore, a framework as been implemented such that a shader exposes which attributes/uniforms it requires from the models. When the model is to be drawn using that shader, it then uploads the appropriate values to the shader. This makes it easy to swap the active shaders, as well as rendering multiple models.

3.2 Terrain generation & chunk system

Only some of the terrain around the submarine is generated, as to fixate the memory consumption and computational work. The challenge is then which parts of the terrain that must be computed. The terrain is built up of 1×1 cubes as described in section 2.1.2. Then $16 \times 16 \times 16$ cubes are combined together to form a *chunk*. The responsibility of the chunk system is to then figure which chunks that must be generated as the user moves around. In essence, all chunks within a certain radius r if the submarine is loaded at any given time. However, it is too computationally expensive to generate every chunk each frame - so as long as the user does not move too far away from where the chunks were computed last, no new generation is done. When the user moves sufficiently far away, some new chunks must be generated – but most chunks are still within a radius of r away. The chunks that are now further away than r are determined. These can be unloaded. The chunks that previously more than r away (but are now within range) take the place of the unloaded chunks in the buffers. The radius r is set to the size of 3 chunks.

When generation occurs, each chunk is generated one after another. The computation of a single chunk is spread over multiple frames as to reduce drops in the framerate when generation occurs.

3.3 Lights & Shadows

2 spotlights are in the project. They are mounted by the "feet" of the submarine. Both spotlights have the attenuation function:

$$\alpha(\theta) = \begin{cases} cos^4\theta & \text{if } -\pi/3 \leq \theta \leq \pi/3 \\ 0 & \text{Otherwise} \end{cases} \tag{4}$$

This means that objects at an angle greater than 60° receives no light from a given spotlight. The attenuation function is visualized on fig. 5, where the black curve is the defined interval, shown by the black vertical lines.

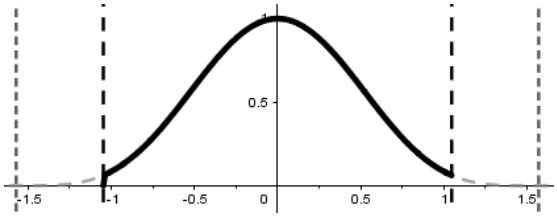


Figure 5: Attenuation function $\alpha(\theta)$ for spotlights

Shadows are created using shadow mapping. Each spotlight has its own texture (of size 1024×1024). As the spotlights only light 60° away from the direction of it, one can conveniently have the field of view to be 120° for the projection matrix when rendering to the shadow texture. To increase the precision of the shadow texture, bitmasking is utilized. Furthermore, percentage-closer filtering is applied to each lookup in the shadow texture to reduce aliasing artifaces even further.[2].

4 Results

Exploring the underwater world created by the implementation yields some nice looking scenes. Some of these are shown on fig. 6 and fig. 7.

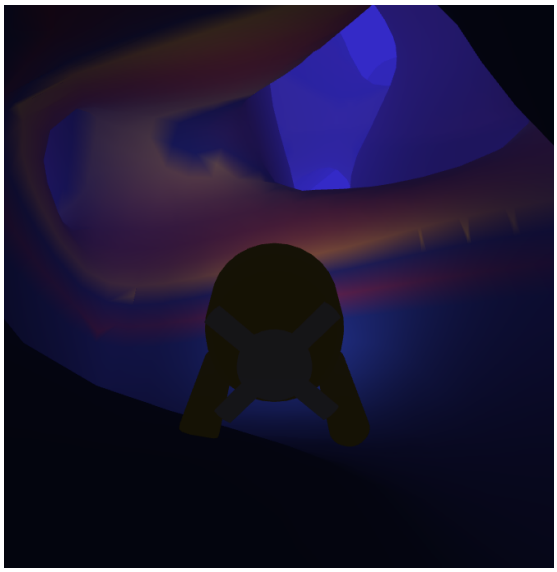


Figure 6: Showcase of explored environment 1.

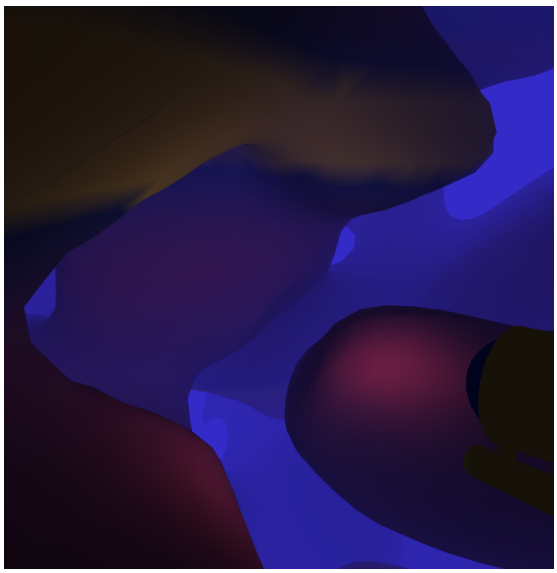


Figure 7: Showcase of explored environment 2.

4.1 Spotlights

To examine how each spotlight contributes to the overall lighting, each spotlight was given its own emissive color (left is green and right is red). By shining on a surface, as on fig. 8, one can see that they both are strongest directly in front of them, gradually decreasing as the angle to the spotlight gets larger. Locations illuminated by both spotlights end up with an additive combination of the emission (here, yellow).

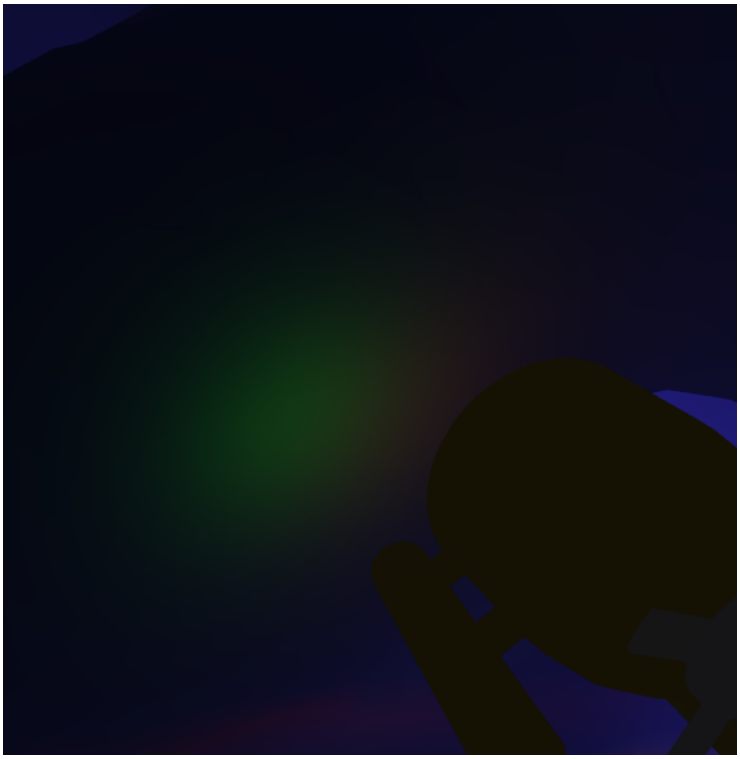


Figure 8: Spotlights with individual colors shining on close surface

4.2 Shadow mapping

By placing the submarine along the edge of some terrain, one can test the functionality of the shadow map. As can be seen on fig. 9, an extrusion in the terrain blocks either none, some or all of the light. The program correctly handles these accordingly. The jagged nature of the shadow may be due to the grid size of the terrain.

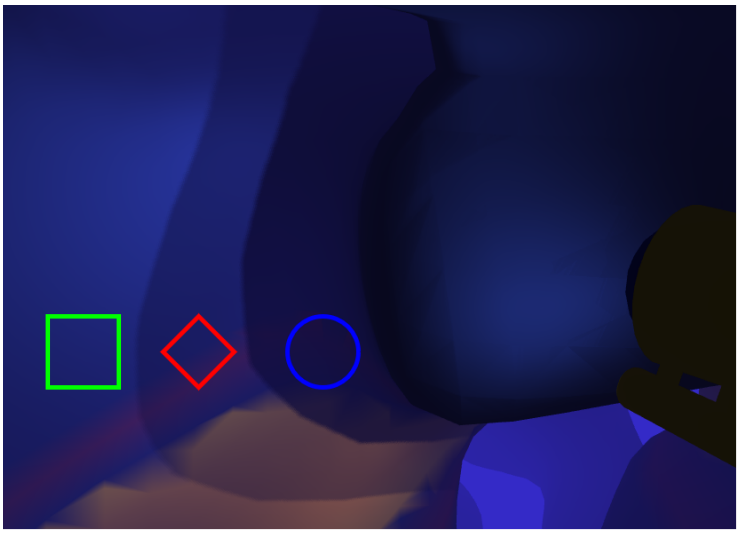
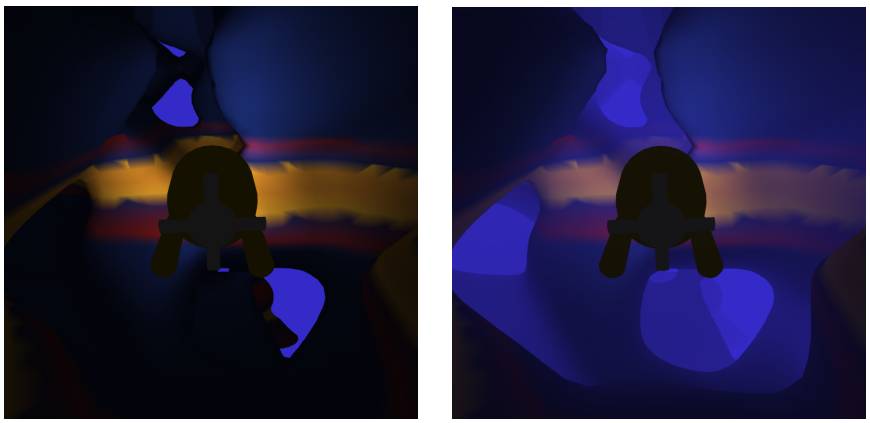


Figure 9: Functionality of shadow mapping. The area marked with the green square is fully lit, the red diamond is in shadow from exactly one of the light sources and the area marked with blue is in complete shadow.

4.3 Distance fog

The visual effect the distance fog has on the rendered scene is compared on fig. 10. As the background color is different from black, it artificially brightens the scene, according to how far away the objects are. This also helps enhance the perception of depth. It also aids in concealing long tunnels that end outside the loaded terrain.



(a) Distance fog turned off.

(b) Distance fog turned on.

Figure 10: Visual comparison when having distance fog turned off/on.

5 Discussion

The project enables user navigation through a vast underwater world, enhancing the experience with concepts from computer graphics. However, there is still room for improvement. Below are some of the different points that could either be added or improved upon:

Additional lighting The submarine is only lit by some ambient lighting, giving it a flat appearance. This could be remedied By adding another light source to the submarine. Another possibility is to allow the terrain to generate light sources, adding more interesting features to it.

Noise functions In the project, the terrain is generated by only sampling the simplex noise once. As described in [5], there are a plethora of ways to create more sophisticated terrain. Adding such to the program could be interesting.

View based chunks Currently, all chunks within a distance r of the submarine are generated – even though some may never be looked at (for instance, if the user moves in a straight line). By using the view volume, one would only have to generate exactly those chunks which the user looks at. This would have a larger spread on when chunks are generated, instead of doing them in batches. This could reduce the drop in framerate even further if a chunk can be generated quickly, improving performance

Collision detection Currently, no collision detection exists between the user and the terrain. The vertices are only stored long term on the GPU – making collision detection impossible. All of the vertices of the chunk the submarine is currently inside could be located in memory as well. Another option is to sample the terrain function around the submarine to determine how close it is to hitting the terrain – and preventing it from passing through.

Automatic navigation Instead of having the user control the submarine, one could add an automatic navigation of the underwater world. This would turn the program into some passive animation.

References

[1] Edward Angel. *Interactive computer graphics : a top-down approach with WebGL*. Boston: Pearson, 2015. ISBN: 9780133574845.

[2] Randima Fernando. *GPU gems : programming techniques, tips, and tricks for real-time graphics*. Boston, MA: Addison-Wesley, 2004. Chap. 11. ISBN: 978-0321228321. URL: https://developer.nvidia.com/gpugems/gpugems/part-ii-lighting-and-shadows/chapter-11-shadow-map-antialiasing.

[3] William E. Lorensen and Harvey E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 163–169. ISSN: 0097-8930. DOI: 10.1145/37402.37422. URL: https://doi.org/10.1145/37402.37422.

[4] Microsoft. *Fog Formulas (Direct3D 9)*. Accessed: 19-12-2021. 2021. URL: https://docs.microsoft.com/en-us/windows/win32/direct3d9/fog-formulas.

[5] Hubert Nguyen. *GPU gems 3*. Upper Saddle River, NJ: Addison-Wesley, 2008. Chap. 1. ISBN: 9780321515261. URL: https://developer.nvidia.com/gpugems/gpugems3/part-i-geometry/chapter-1-generating-complex-procedural-terrains-using-gpu.

[6] Ken Perlin. *Noise hardware*. 2001. URL: https://www.csee.umbc.edu/~olano/s2001c24/ch09.pdf.