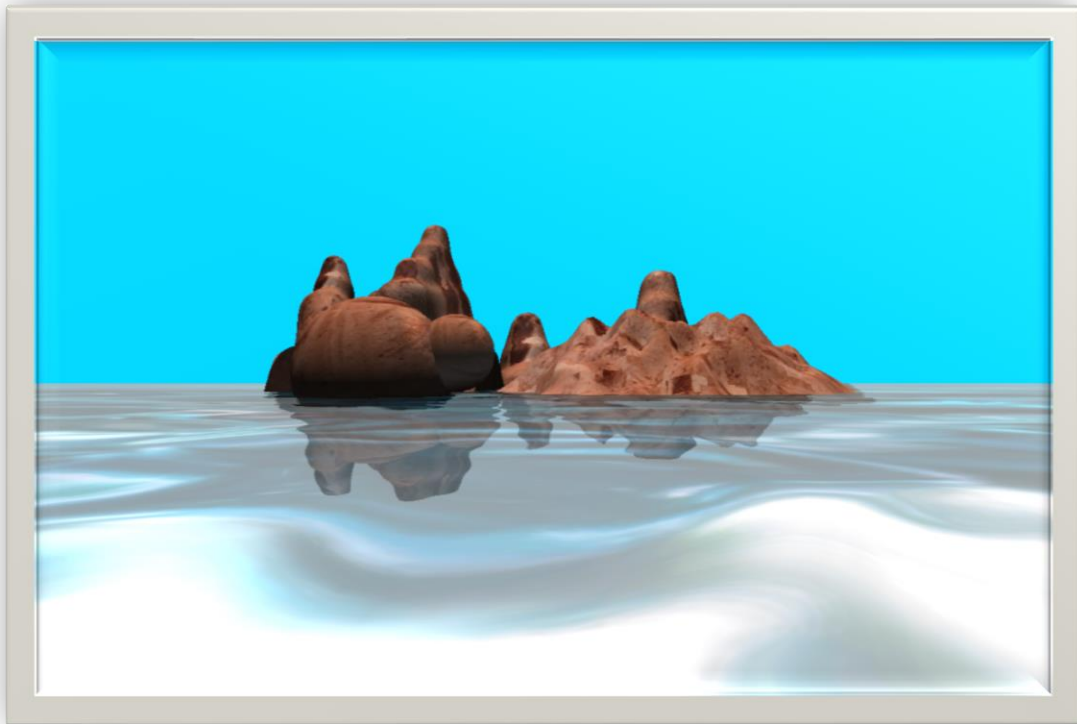


186.140 VU Echtzeitgraphik

Submission 2 – Final Demo

TreasureHunter



Takaki Hagen	11701261	066 932
Sarah El-Sherbiny	01126592	066 932

1. Story of the demo:

Our goal is to find a treasure, which is located in the center of an island. We have to fly across the water to reach the island. Then we fly over the island, to find the treasure – it's a special coin.

2. Scene description:

We start in the middle of the water and fly to an island, which is far away. After reaching it, we fly through the mountains and find the treasure in the center of a mountain. The story ends, when the camera stops in front of the treasure.

3. Implemented effects:

Water with reflection and refraction [1]: For the wave of the sea.

Environment mapping [2]: For the treasure.

Glow/Bloom [18]: For the green lucky sun.

4. Implementation details:

4.1 Effects

Environment mapping:

We implemented the reflection of the treasure (Fig.1b) with cube mapping [1, 14]. The idea of this reflection method is, each point of the reflecting object has the color of a point of the cube map, where the reflection ray and the cube map intersects. Fig 1 shows the simple graph of this method. The injection vector i from the camera to a point t on the treasure is calculated and the reflection vector r is calculated with normal vector n and i . The vector r is elongated till it intersects with the environment cube map. The point t of treasure gets the color of point c on the environmental map.

The environment cube map is generated with rendering the scene 6 times for different directions (plus minus x , y and z .) The camera is located at the position of the reflection object. Then, in every rendering loop, the camera renders 6 times the scene and saves it as a cube map texture. In this project, as a first step, we allocated the cube map texture. Then we created a frame buffer and render buffer. We attached the render buffer to the frame buffer so that the textures are saved to the render buffer and we can refer the textures with the frame buffer. The size of each surface of the cube map is 1024x1024.

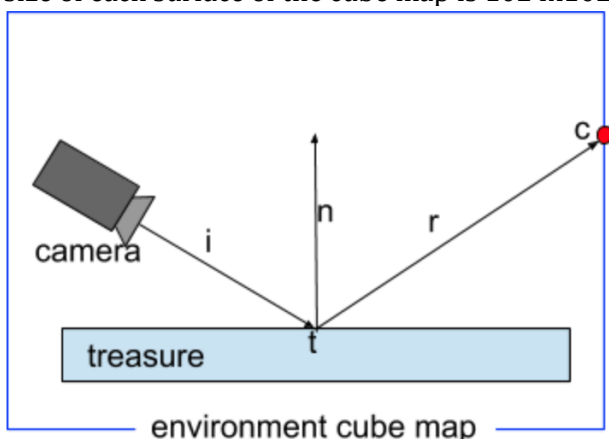


Fig.1a: Relation of camera and environment cube map



Fig.1b: Environment mapping

Water with reflection and refraction:

For the water effect [1, 15, 16], we started by rendering the scene to a texture twice, to create the reflection and refraction effect. Therefore we used two Framebuffers with a color buffer for the surface texture and a depth buffer for storing the water depth. We clipped the scene so that the area above the water is captured for the reflection effect, while a area under the water is used for the refraction effect. To capture the reflections, the camera is moved under the water. Both textures for the reflection and refraction are then mixed together. To create moving waves, we used a dudv Map, and derived distortions of the captured reflection and refraction textures from it. The distorted textures are blended together with some blue color. To increase the structure of the water waves, we additionally used a color texture, which we distorted and blended to the result. We used the view vector of the camera and the normal vector of the water surface to add the fresnel effect, so that the water is more reflective at lower angles (Fig.2b), while it is more transparent at higher angles (Fig.2c). To add highlights to the water, we used a normal map for the lighting calculations (Fig.2a).

The normal map and the dudv map are both taken from ThinMatrix [15]. The additional color texture is taken from Pexels [16].

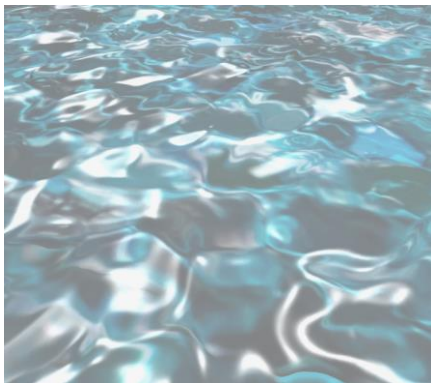


Fig.2a: Normal mapping

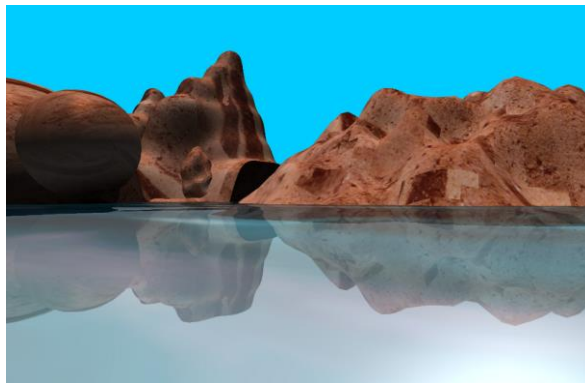


Fig.2b: Water reflection

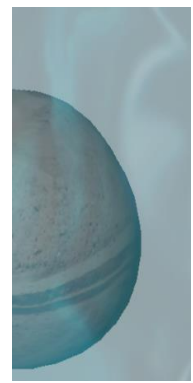


Fig.2c transp.

Glow/Bloom:

For the glow effect [18, 19], we first rendered the scene to a texture. Then we extracted the glowing parts out of it. We blurred the extracted parts with a gaussian filter, and combined them with the scene texture. The effect can be seen on the green lucky sun, shown at the beginning of the demo (Fig.3).

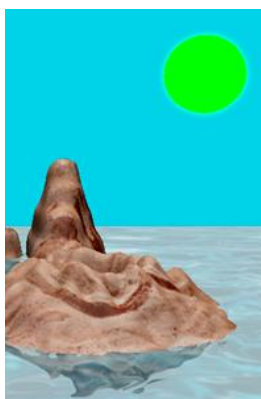


Fig.3: Glow

4.2 Base framework

We used the **ECG framework** as the basis for our assignment.

4.3 Development environment

- OpenGL 4.3 Core Profile
- Visual Studio 2015
- Windows 10 64-bit

4.4 Models

All models are selfmade with Blender [13].

For importing the models, the Assimp library [11] is used [9, 10].

4.5 Illumination

The scene contains a point light and a **directional light**. Both are based on the **phong** reflection model. Only the directional light is used as a sun light in our scene. The intensity of the specular light and diffuse light is based on the material of the models.

4.6 Texture

All objects in the scene – the mountains, the island, the water, the treasure – are textured. The uv coordinates of the island models are retrieved by a simple uv-unwrapping. Therefore they are not always perfectly aligned.

4.7 Camera

After starting the demo, a automatic camera movement can be seen. A manual camera can be activated by using the SPACE key. Then the ARROW keys and the WASD keys can be used to manually move around.

4.8 Scene navigation

Key	Action
SPACE	Switch between automatic camera mode and manual camera mode
ARROW UP	Move the manual camera forwards*
ARROW DOWN	Move the manual camera backwards*
ARROW LEFT	Move the manual camera to the left*
ARROW RIGHT	Move the manual camera to the right*
W	Move the manual camera upwards*
S	Move the manual camera downwards*
A	Rotate the manual camera to the left*
D	Rotate the manual camera to the right*
Q	Rotate the manual camera upwards
E	Rotate the manual camera downwards
ESC	Exit demo
F1	Wireframe on/off
F2	Culling on/off
F3	Glow on/off

*Can only be used, if the manual camera was activated before with the SPACE key

4.9 Libraries

ASSIMP [11]: For object loading.

GLM [12]: For mathematical functions.

4.10 Graphics card

We tested our demo on SHADOWCAT and AMAROK with NVIDIA GTX 1060 (and AMD RX580).

It works on both graphics cards, but we prefer **NVIDIA**.

5. References

- [1] <http://habib.wikidot.com/projected-grid-ocean-shader-full-html-version>, accessed 11.10.2018
- [2] <http://publications.lib.chalmers.se/records/fulltext/242141/242141.pdf>, accessed 11.10.2018
- [3] <https://chewitt.me/Papers/CTH-Dissertation-2017.pdf>, accessed 11.10.2018
- [4] https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch13.html, accessed 11.10.2018
- [5] <https://link.springer.com/article/10.1007/s11042-017-5267-8>, accessed 11.10.2018
- [6] https://link.springer.com/chapter/10.1007/11919476_17, accessed 11.10.2018
- [7] http://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch21.html, accessed 11.10.2018
- [8] <https://learnopengl.com/Advanced-OpenGL/Cubemaps>, accessed 11.11.2018
- [9] <https://nickthecoder.wordpress.com/2013/01/20/mesh-loading-with-assimp/>, accessed 11.11.2018
- [10] <http://www.nexcius.net/2014/04/13/loading-meshes-using-assimp-in-opengl/>, accessed 11.11.2018
- [11] <http://www.assimp.org/>, accessed 14.11.2018
- [12] <https://glm.g-truc.net/0.9.9/index.html>, accessed 25.11.2018
- [13] <https://www.blender.org/>, accessed 25.11.2018
- [14] <https://ieeexplore.ieee.org/document/4056759>, accessed 11.12.2018
- [15] https://www.youtube.com/watch?v=HusvGeEDU_U, accessed 21.01.2019
- [16] <https://bonzaisoftware.com/tnp/gl-water-tutorial/>, accessed 21.01.2019
- [17] <https://www.pexels.com/de-de/foto/abstrakt-blau-glas-hell-753413/>, accessed 21.01.2019
- [18] http://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch21.html, accessed 24.01.2019
- [19] <https://learnopengl.com/Advanced-Lighting/Bloom>, accessed 24.01.2019