Karlsruher Institut für Technologie

Department of Informatics

Institut für Visualisierung und Datenanalyse,
Chair of Computer Graphics

24912 – Praktikum Graphics and Game Development

Planet Exploration

Personal Report:
Joans Heinle

September 16, 2021

Contents

List of Figures

List of Tables

1 Proposal

Our team consists of Jonas Heinle and Kansei Hara

Goal: Develop a spaceship simulation where the planet terrain is generated procedurally and the focus is on the graphics. You can change the terrain with parameters for light, Perlin noise and graphic parameters.

Milestone 1:

- Simple OpenGL application (Jonas)
- GUI for Graphic settings, Light change, procedural Terrain setting (Jonas)
- Shader loading/phong (directional point und spot) (Jonas)
- Camera system: transformations (Jonas)
- Texture loading (Jonas)
- Mesh generation and loading (Jonas, Kansei)
- Prototype/Information Gathering: Procedural Terrain generation (Kansei)

Milestone 2:

- improving mesh loading (Jonas)
- Sky box, Atmospheric effects information gathering (Jonas)
- starting with shadow mapping (Jonas)
- Light API (Jonas)
- Deferred rendering (Jonas)
- Texturing Terrain(Kansei)
- Procedural Terrain generation (Kansei)
- search .obj .mtl files(Both) (Both)

Milestone 3:

- add minimal game logic to it; be able to control the spaceship(Jonas)
- PBR (Physically based rendering → Materials, Lighting, BRDF, Microfacettenmodell, Normal Distribution Function, GGX/Smith)(Jonas)
- Cascaded Shadow Mapping with PCF(Jonas)
- Maintaining the GUI (Jonas)
- Point lights with omnidirectional shadow maps(Jonas)
- View frustum culling (Jonas)
- Atmospheric effects: procedural generated clouds(Jonas)
- Place objects on the terrain(Kansei) for example trees that can grow on that planet effects/colors of snow sand ....

Milestone 4:

- Everything must work correctly, tweaking(Both)
- Cascade tweaking (number of cascades and overlapping) (Jonas)
- improvement of procedural clouds (Jonas)
- add linear interpolation between cascade points (Jonas)
- Polishing: if there is time Anti Aliasing: FXAA(Jonas), SSAO(Jonas)
- Optional: Terrain with Water(Transmission, Reflection)(Kansei)

2 First Milestone

2.1 Task 1: Setting up

What I've implemented
To speed up the process of window creation, setting up the OpenGL context,etc. I used GLEW [10] and GLFW [3] since they are very popular in the community and well known to me from projects in the past [12].

Problems, Difficulties and Challenges
Since there is not that much to alter for each application this was rather easy and pretty fast to set up. No problems occured.

2.2 Task 2: GUI

What I've implemented
I used ImGUI [4] for User I/O handling.

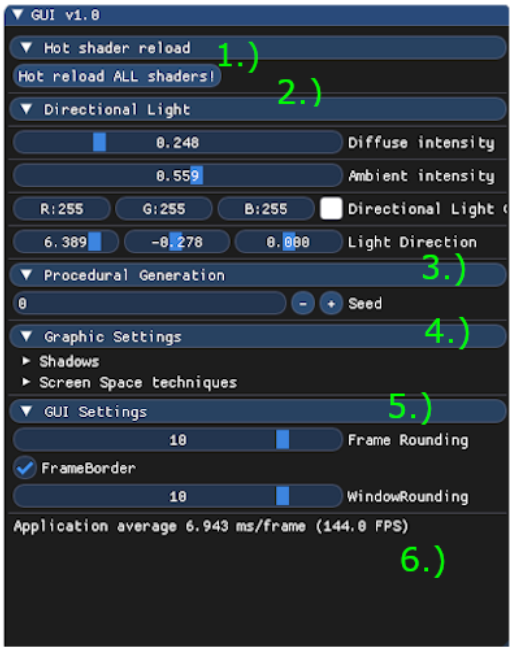

Figure 1: The GUI

1. I can hot reload shaders now. This safes a lot of time (truly wasn't much work and saves a lot of work) :)
2. Control all light settings. Very helpful for tweaking graphics.
3. Should become the interface to the procedural generation
4. Incomplete Graphic settings
5. We can also alter the interface at runtime
6. Observe the fps; does not replace nsights graphics [18] but gives first insight

Problems,Difficulties and Challenges
First, I didn't take into account ImGUI [4] isn't threadsafe. That cost me 30 minutes to realize and fix. Once this was done the use is pretty straight forward.

2.3 Task 3: Mesh and Textures

What I've implemented
The mesh is straightforward implemented as an vector of Vertices, each consisting of position, normal and texture coordinates. Furthermore you must provide the mesh with an array of indices for supporting indexed rendering. Non indexed rendering is not efficient and not supported by the

mesh!
Using the famous STB library[5] loading textures has become straight forward. One must know whether the texture has an alpha channel. Consider the fact that currently you will have to explicitly call the "use_texture()" functions before every use of a texture so it is bind to the right texture unit.
Problems/Difficulties
More complex models might need more complex texture handling. It might be useful to encapsulate the texture usage into the model. Therefore you don't need to further explicitly call the function. But for a first usage it is working.

2.4 Task 5: Lights

What I've implemented
Directional light is supported with an good structure for adding spot/point lights later on. For now one can interact with all light attributes at run-time (see my results 8)(I am passing the light as an uniform from the application to the fragment shader).
Problems/Difficulties No further problems occured.

2.5 Task 6: Camera System

What I've implemented
I used a system the relies on a camera position and on a yaw/pitch/roll-approach. (inspired by [30, learnopengl.com]
Problems/Difficulties
Finding a good starting moving/rotation speed was a bit difficult. Since there are coded as a pure number without further interpretation one has to tweak them a bit.

2.6 Task 4: Shaders

What I've implemented

Passing in uniforms from the application and Buffer Objects (location is defined in the mesh with the glVertexAttribPointer() function). Phong shading with diffuse term (see results 9)

```
#version 330

layout(location = 0) in vec3 in_position;
layout (location = 1) in vec3 in_normal;
layout (location = 2) in vec2 in_tex_coord;

out vec2 tex_coord;
out vec3 normal;

//uniform variables
uniform mat4 projection;
uniform mat4 view;
uniform mat4 model;

void main() {
        gl_Position = projection *  view *
                    model *
                    vec4(in_position, 1.0f);
        tex_coord = in_tex_coord;

        normal = mat3(transpose(
                      inverse(
                      model))) *
                      in_normal;
}
```

Figure 2: Minimalist vertex shader

```
#version 330

in vec2 tex_coord;
in vec3 normal;

out vec4 color;

struct Light {

    vec3 color;
    float ambient_intensity;
    float diffuse_intensity;

};

struct DirectionalLight {

    Light base;
    vec3 direction;

};

uniform sampler2D brick_texture;
uniform DirectionalLight directional_light;

void main() {

        float kd = directional_light.base.
                   diffuse_intensity *
                   max(0.0f, dot(normalize(
                   directional_light.direction),
                   normalize(normal)));
        color += texture(brick_texture,
                         tex_coord);
    color += kd * vec4(directional_light.base.
                 color, 1.0) *
        directional_light.base.ambient_intensity;

}
```

Figure 3: Minimalist fragment shader

Figure 4: My shaders

Problems/Difficulties
I reconsidered some calculations. One should make calculations in the latest possible pipeline stage.[20] Therefore I am passing now the light to the fragment shader and calculate the dot product there. This would have become otherwise a bottleneck in the future when the amount of vertices will increase further.

2.7 Task 7: Model loading

What I've implemented
I used the tinyobjectloader [6] to ease the process of model loading.The example viking room model (see picture 8) with its attached textures are stolen from a Vulkan tutorial [28] I've already known. and the documented tutorial for implementing model loading. [28]
Problems,Difficulties and Challenges
I had problems with efficiently implementing indexed rendering. For they are absolutely necessary for not wasting GPU-Memory I tried many things. What seems to work quite good is a hash map. ([2, VulkanToturial], since model loading is render independent i looked it up here because i used the tutorial earlier in a project with Vulkan)

```
std::unordered_map<Vertex, uint32_t> vertices_map;
...
/*
* here is code for loading vertex data from .obj data
*/
...
//here we want to buffer all
if (vertices_map.count(vert) == 0) {
    vertices_map[vert] = vertices.size();
    vertices.push_back(vert);
}
```

Figure 5: Save indices in hash map

Furthermore it can be problematic saving all vertices to one mesh and rendering it. For upcoming milestones I need to rethink the strategy and rather load shapes separately in a mesh list, index them with a material id, a texture before rendering.

2.8 My Results

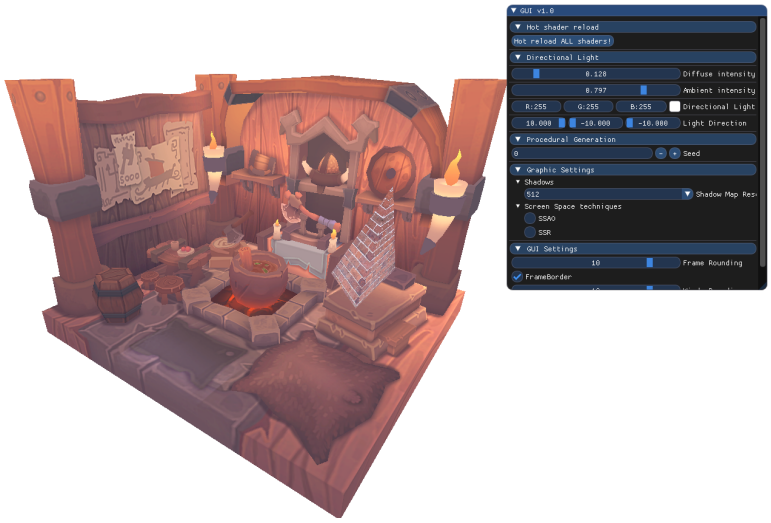

Figure 6: Reddish room

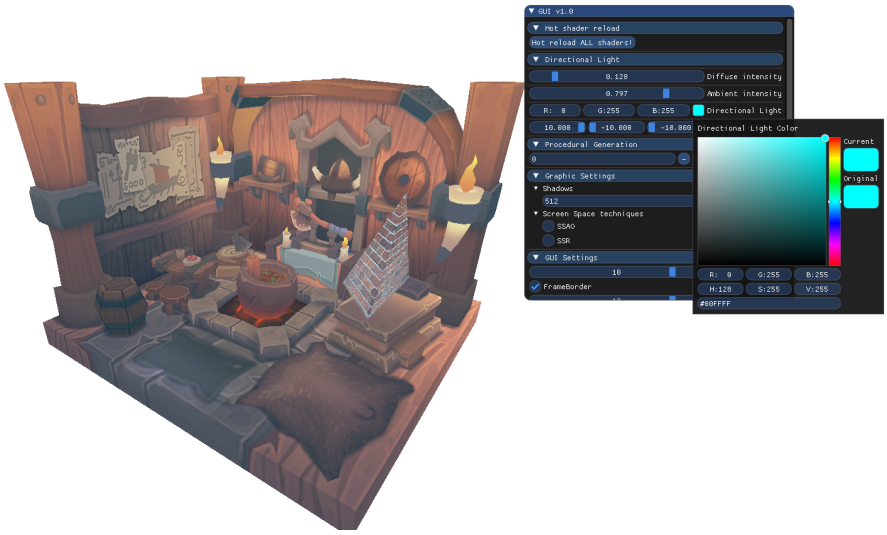

Figure 7: Bluish Viking room

Figure 8: Viking room with example interaction

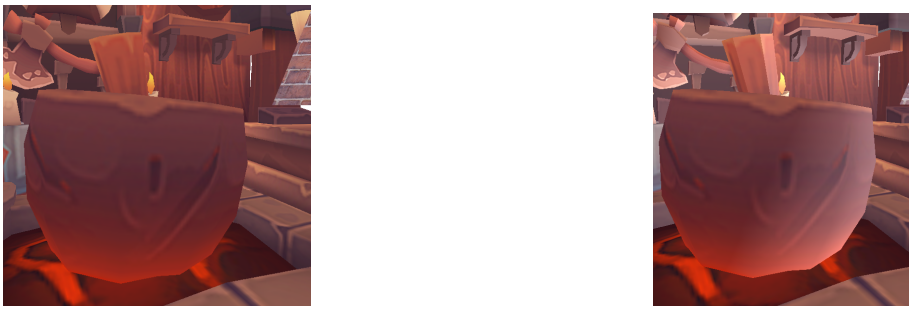

Figure 9: Phong shading (left:no diffuse, right: with diffuse)

3 Second Milestone

3.1 Improved model loading

One problem in the past milestone has been the model loading. We didn't support per face texturing and material information loading. Now with this problems being solved we are better prepared for the next milestones goals where we will implement PBR (physically based rendering) with the help of proper materials.

3.2 Skybox

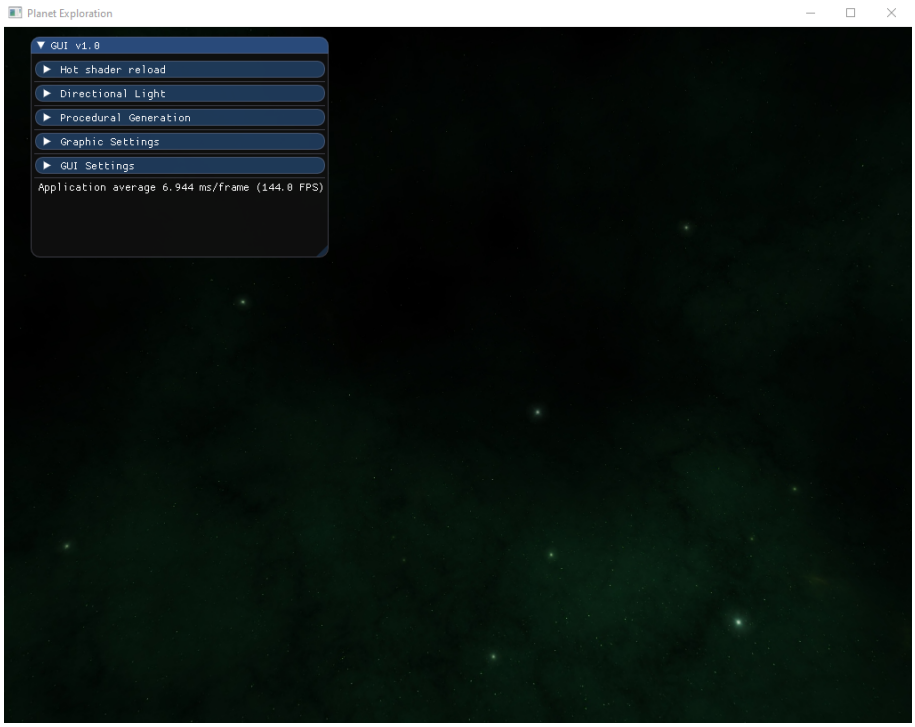

Figure 10: Skybox

The rendering of a skybox [31] is now the first step in our rendering pipeline (see 3.3). For the skybox the first step is a 3D-texture [27] where we will shade all pixels accordingly after a lookup with a direction vector in the texture. In a second step for pixels where objects will be visible after the geometry and lighting pass the skybox is omitted and the objects are drawn to. Therefore there will always a space be visible on every place where is no object.
This contributes to the feeling of being in space rather than having just a black background.
The integration of the skybox into the deffered rendering approach has been a challenge. I implemented it in the way of completely rendering it into the albedo texture where in the geometry stage we will render everything of every object to and just "forget" what has been rendered in the skybox pass for that pixels.

3.3 New Pipeline Layout

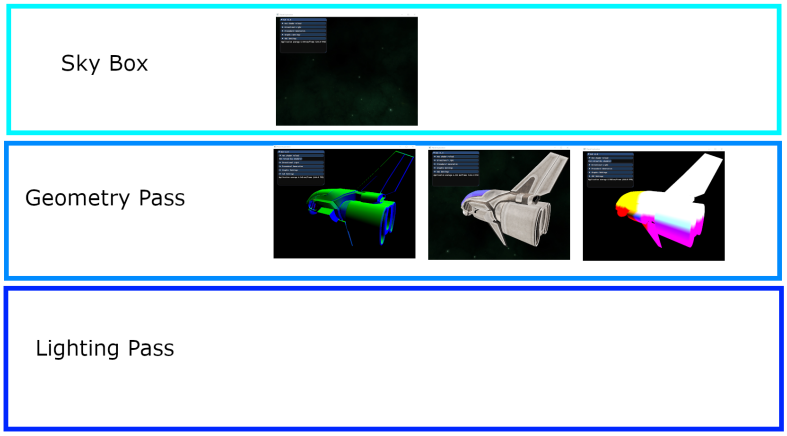

Figure 11: Pipeline

Additionally an extra buffer for the world position in light source coordinates is attached for implementing cascaded shadow mapping. With this extra information and an extra depth only pre-rendering pass into a shadow map we can achieve good shadows.

3.4 Deffered Rendering

As described in Figure 3.3 we have switched to the state-of-the-art technique "deferred rendering" which will allow us in further milestones to use modern screen space techniques like SSAO, etc. With splitting our rendering into two basic steps, the geometry and the lighting pass, we hope to decouple the shading from complex geometry and simultaneously having the Gbuffer for later steps.

3.5 Atmospheric Effects

Due to the absence of the terrain I decided to move this section to the next milestone. [8]

4 Shadow Mapping

I already started with shadow mapping with PCF. Due to my waiting for the terrain generation I started with implementing shadow mapping. As mentioned in the previous section 3.3 I've set up an extra depth rendering pass into the shadow map. Here all depth values from the light position will be stored.
Then in the following geometry rendering pass I will set up an extra texture with the fragment world position in light coordinates so we can use those position in the lighting pass for comparing the current fragment depth to the previous rendered depth to the light source.
So we have 5 textures as input to our lighting pass: position, albedo, normal, position in light source coords and the shadow map.
Currently there is a problem in the shadow map. Somehow the rendered shadow map can't be displayed and/or wrong depth values. So I question myself whether the depth texture is correctly rendered at all or the copying is false.

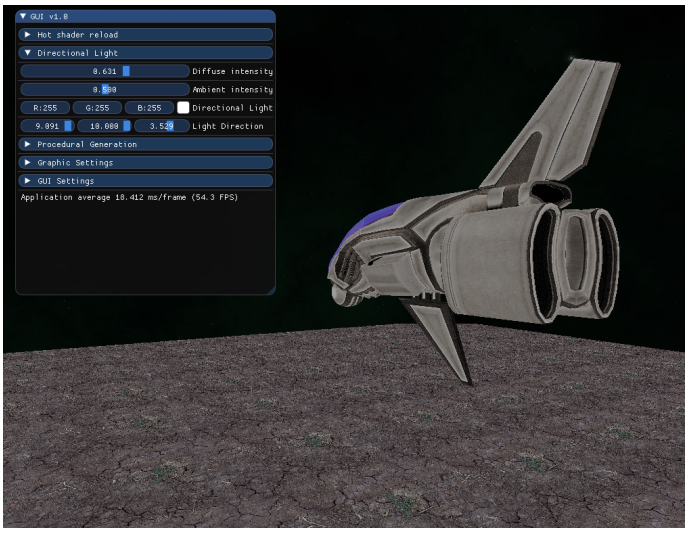

Figure 12: Test floor for shadows

5 General fixes

- The window wasn't truly resizeable. I forgot to change the framebuffersize when the user changed window size.
- If there is an other clear color than vec4(0.0f,0.0f,0.0f,1.0f) in the geometry stage it will mess up everything

6 Third Milestone

6.1 Cascaded shadow mapping

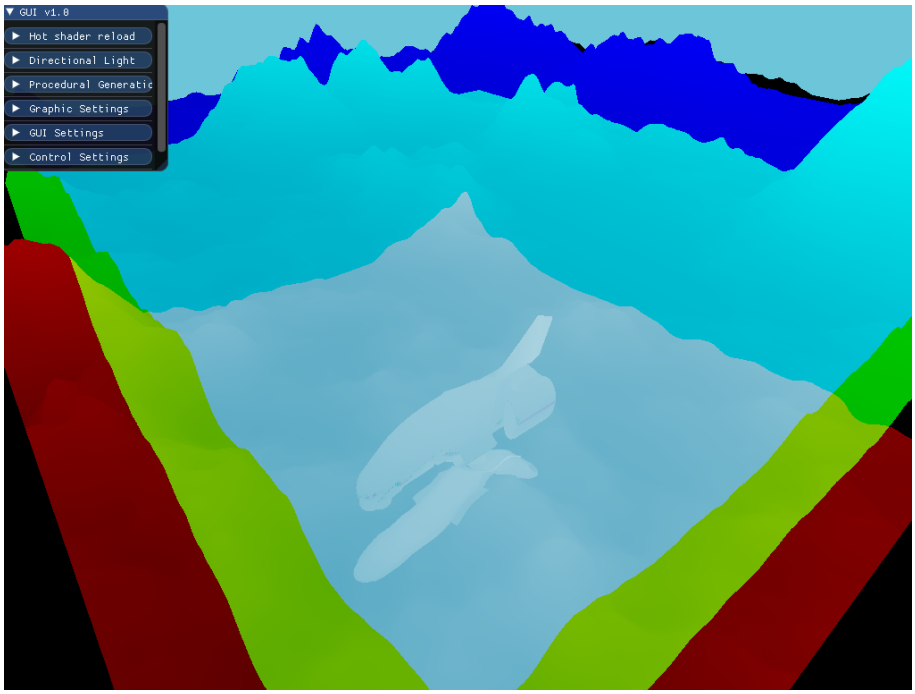

Figure 13: Shadow cascade visualized

Conventional shadow maps have a problem with covering big areas. Shadow maps result in surface acne and aliasing when covering huge scene spaces. A well-known technique to encounter this problem is to split the view frustum into different cascades and calculate separate shadow maps for each part.
We plan to have big procedural generated terrain in our game with the sun as directional light source. Therefore we decided to use this technique. Furthermore it is a already good explored technique [17] [26] [20] and quite handy to implement.
For now we go with 3 cascades (see fig.13. One covering the very front of our view frustum(red), one the middle(green) and one the very end(blue). Right now the closest cascade endpoint is chosen for selecting the appropriate shadow map. But in the next milestone I want to add a linear interpolation between them.

6.1.1 Errors/problems

As there is now linear interpolation between the shadow maps one can see popping artefacts which are unacceptable. It should be quite easy to implement and is postponed due to time constraints to the next milestone.
An other artifact is the vanishing of shadows(see fig. 14) due to sub-optimal chosen cascade start and end points. This will be an other task for milestone 4 to fix.
Renderdoc [13] has been a huge help in this milestone.

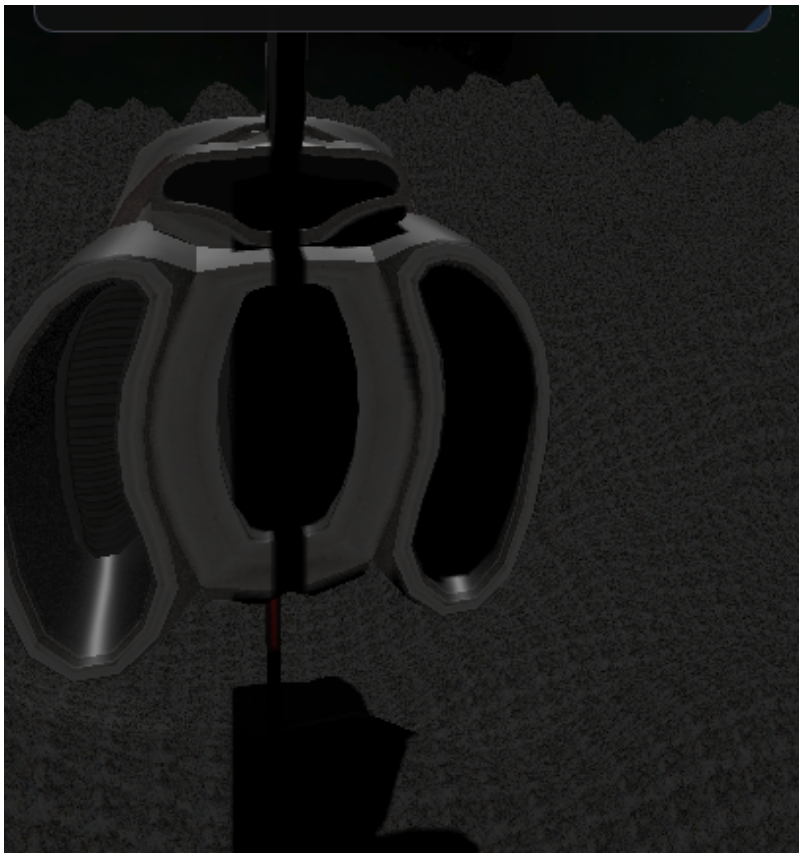

Figure 14: Sub-optimal cascade start and end points; In specific camera configuration some areas are not covered with a shadow map from the cascade and therefore incomplete/no shadows are appearing

But nevertheless we have nice realistic shadows now (see fig. 15).

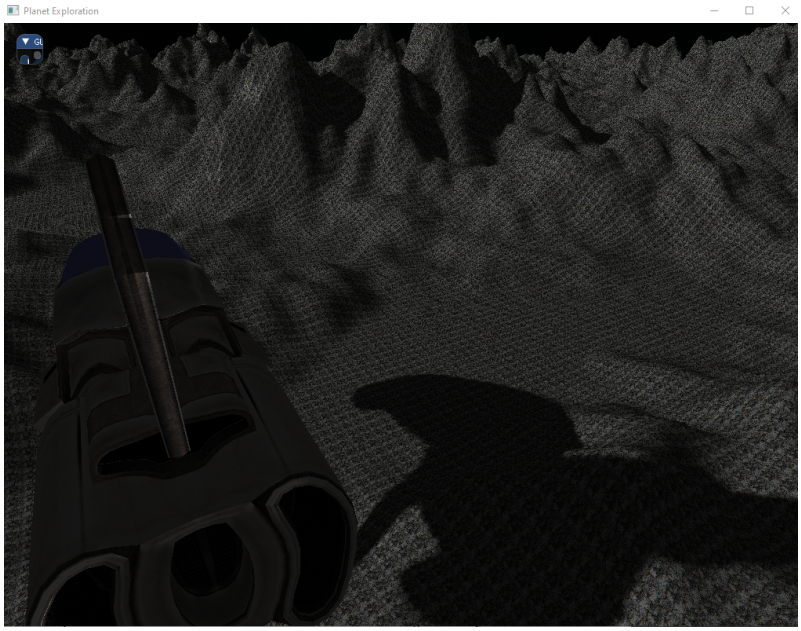

Figure 15: Shadow

6.2 Point Lights

Point lights contribute to the atmosphere of the game. Therefore I added them to the game and set 3 on the space ship (see fig. 16).

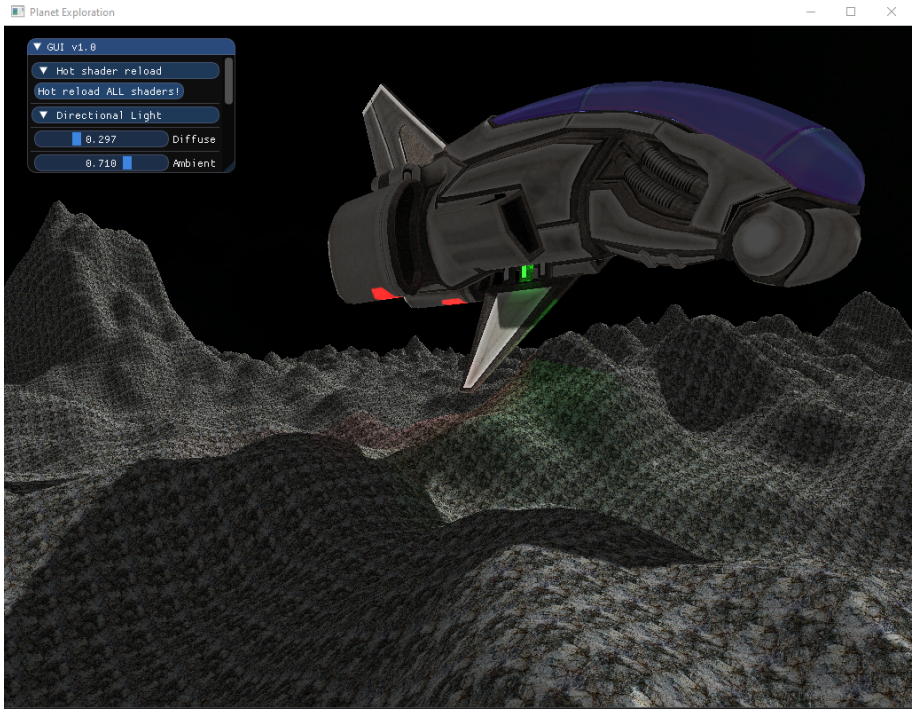

Figure 16: Point Lights with omnidirectional shadow maps

6.3 Game Logic

Our goal is to have a quite minimal game logic. We want to explore the planet with the space ship. Therefore I added a mode for controlling the space ship and move with it in the scene in a third person manner(see fig. 17).

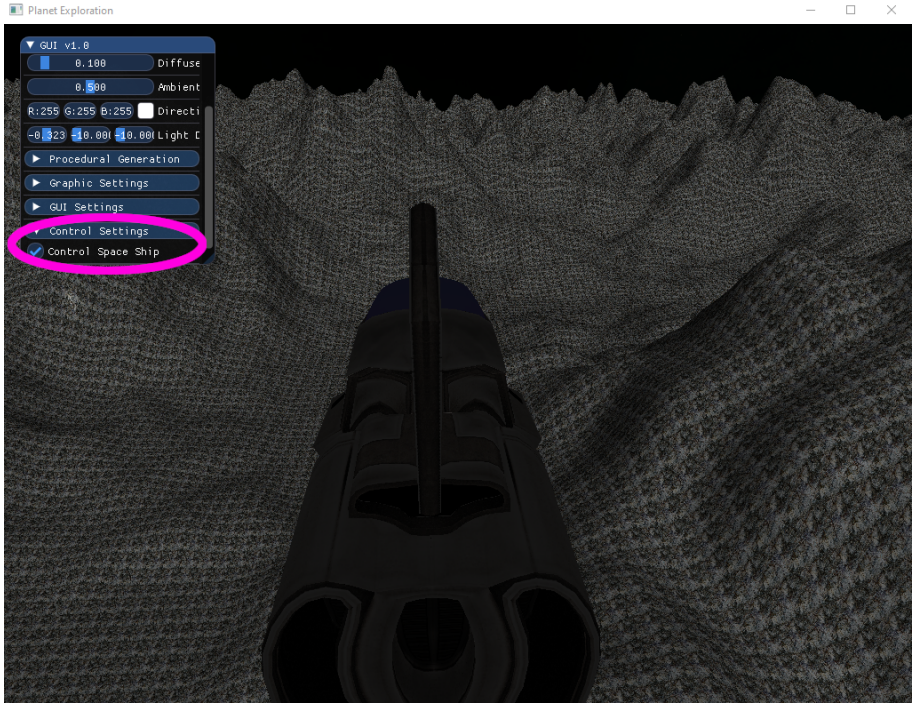

Figure 17: Shadow Cascade visualized

6.4 PBR

Furthermore it is our goal to support physically based rendering with multiple material support [30] [29]. We support a micro facet model with a Cook-Torrence BRDF. A Schlick-Approximation for the reflected term, a Trowbridge-Reitz GGX for the normal distribution function and Smith's Schlick GGX as geometry term.
We can support a much more realistic metallic appearance of the space ship. (fig. 18)

Figure 18: Our space ship looks metallic now

6.5 View frustum culling

Without we could experience a drastic frame drop. The procedural terrain generation leads to huge scene geometry and in combination with the PBR 6.4 approach to a way too large computaional overhead. Therefore I implemented the planed view frustum culling algorithm (see [20] [14]). For our models we generate AABB's for intersect them with our view frustum (see fig 19).

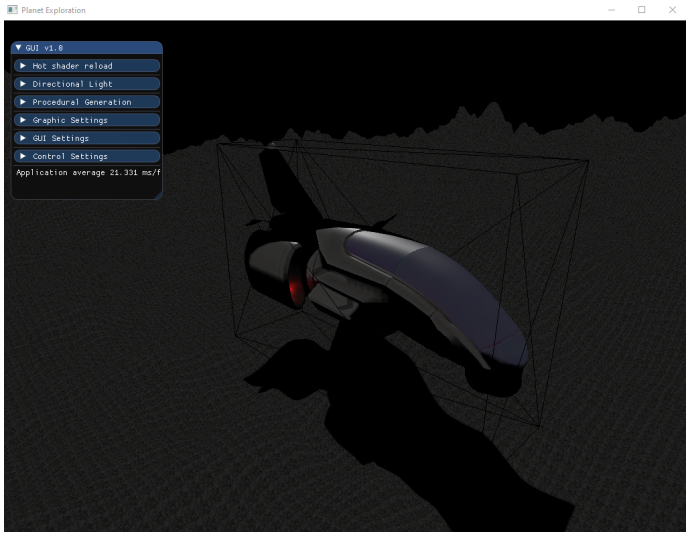

Figure 19: AABB as wire frame around our loaded models

As we load the terrain in chunks I assign one AABB to each chunk and intersect them with the view frustum (see fig. 20).

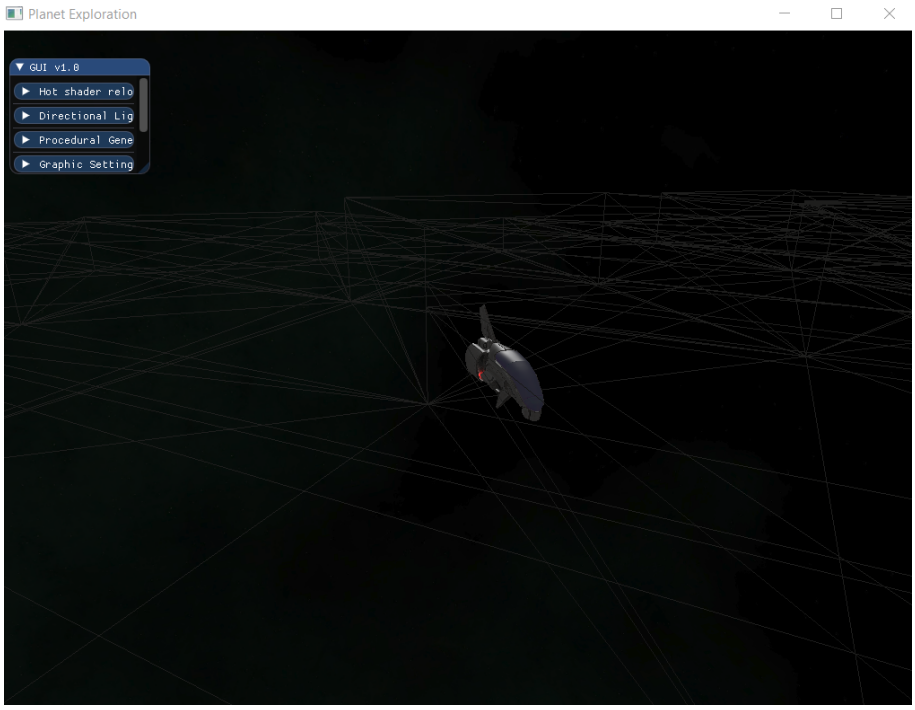

Figure 20: AABB as wire frame around our chunks of the terrain

6.6 Atmospheric Effects

A game needs atmospheric effects. Otherwise it would not be as immersive as it should be. I added procedural generated cloud volumes based on a technique from Horizon: Zero Dawn[24] [21] [16] [23] which layers different perlin-worley noise frequencies.

Figure 21: Procedural generated clouds

7 Fourth Milestone

7.1 Game Logic

One can now control 4 different space ships (see 22). This creates a much better game experience for games live from variety.

Figure 22: You can choose between 4 different ships now

7.2 GUI

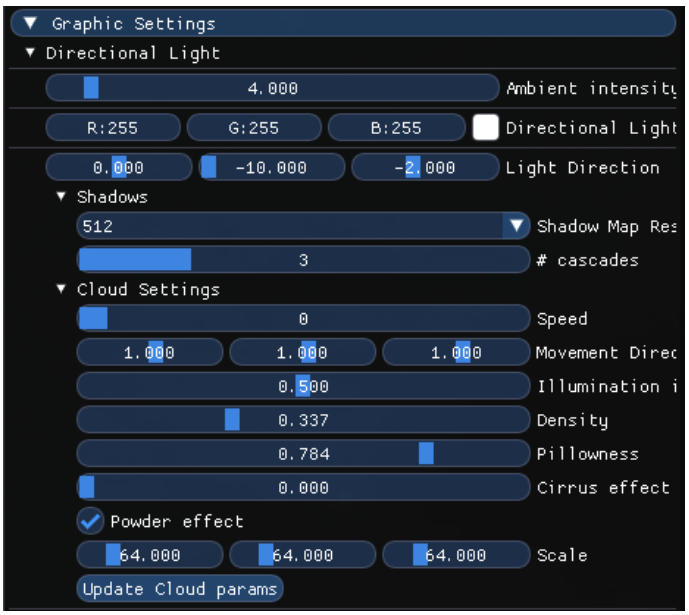

Figure 23: You can interactively change graphic settings without reloading

One can now choose between different shadow settings (see fig. 23) and therefore the user is able to adjust the graphics for his purpose. This is very important. Having no adjustable graphics is not acceptable in a game. Also you are able to control the cloud appearance.
Audio is adding the necessary atmosphere to a game. Therefore I added audio to this game (24).

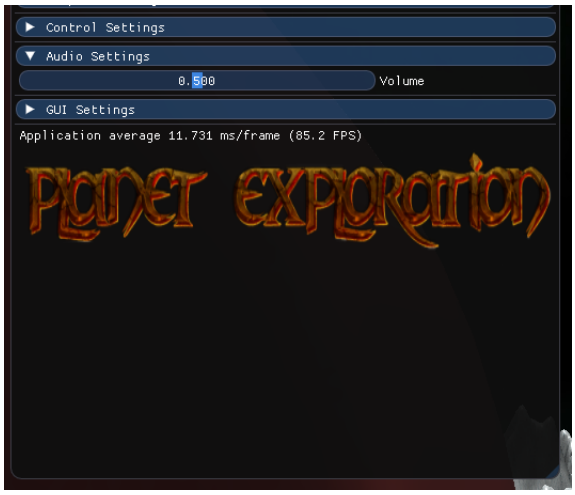

Figure 24: You can interactively change graphic settings without reloading

7.3 Clouds

7.3.1 Noise

I completely overworked my cloud system and developed a cloud system which is good adjustable for different cloud shapes (see GUI in fig. 23).

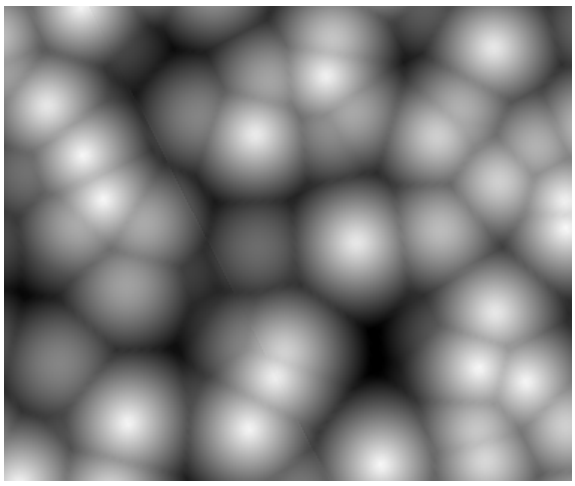

Figure 25: Inverted worley noise gives us already good pillow shapes

In the last milestone I experimented with third-party 2D/3D-Worley noise [23] [21] whereas in this milestone I created my own 3D worley noise (see 25) based on good 2D-worley noise work [7] [19]. Many previous work [25] [24] is dilating worley noise with different frequencies in a FBM (=Fractional Brownian motion) Manner(IQ's blog is a very good resource [22]) for getting some plausible cloud shapes (see fig. 26)

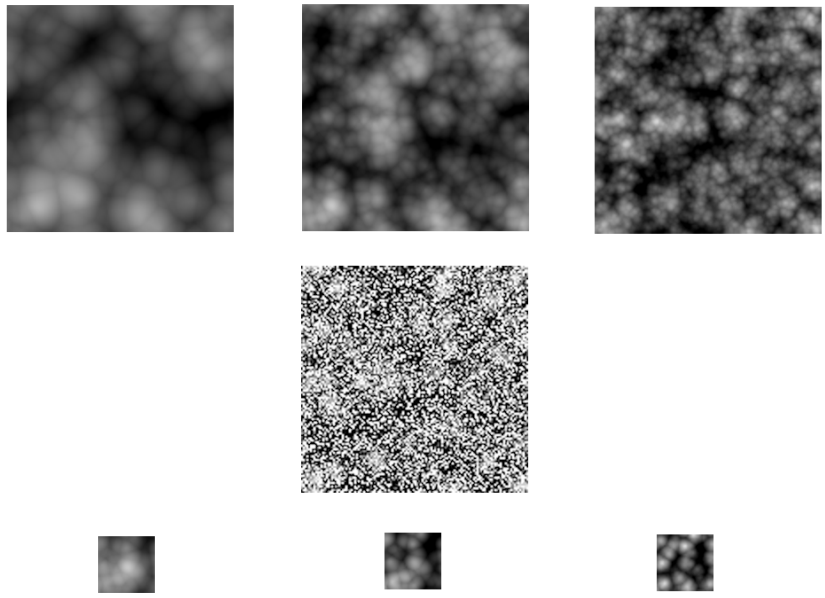

Figure 26: Clouds

Figure 26 shows the resulting textures I use for my cloud system. **The first row** shows the first slice of 3 128x128x128 textures which are made with an FBM like layering of worley noise with increasing frequencies.**The second row** is a worley noise dilated with classic perlin noise (for perlin noise I use a existing very robust glsl implementation [1]). **The last row** shows dilated worley noise with a lower resolution to add extra detail. I use an extra compute shader (good toturial [9]) pass to create the noise.

7.3.2 Implementation

With the noise textures we can now do ray marching for calculating clouds. A AABB serves as a placeholder to do an intersection test (fast box-ray intersection test from IQ's blog [22]). Inside the cube one can now calculate the transmittance and the received light energy accordingly to Beer's Law and the Henyey-Greenstein- phase function. [15]

7.3.3 Problems

I encountered many problems. One important part is to make the worley noise tileable. If they are not tileable many very unpleasant artefacts appear.(see 27)

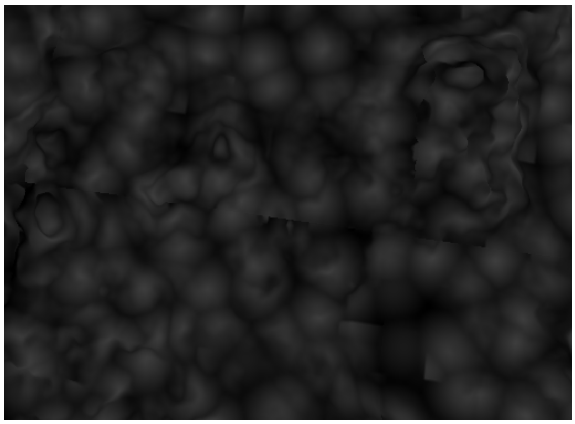

Figure 27: Making noise tileable is very important for the visual appearance

Too many ray marching steps causing significantly performance decreases whereas to+ less steps are causing visible band effects 28

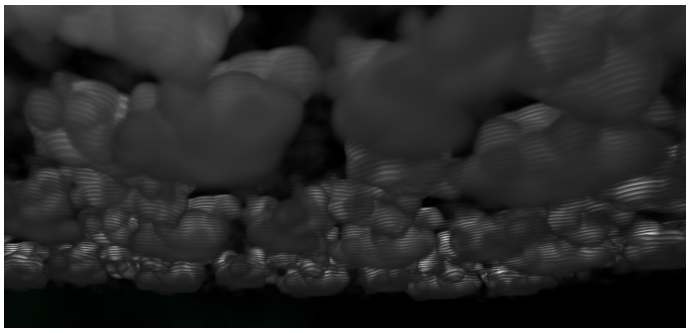

Figure 28: Too low step size while ray marching causing artefacts

7.3.4 Results

This approach with the many different parameters to influence (see 23) gives us the ability to create many different cloud types.

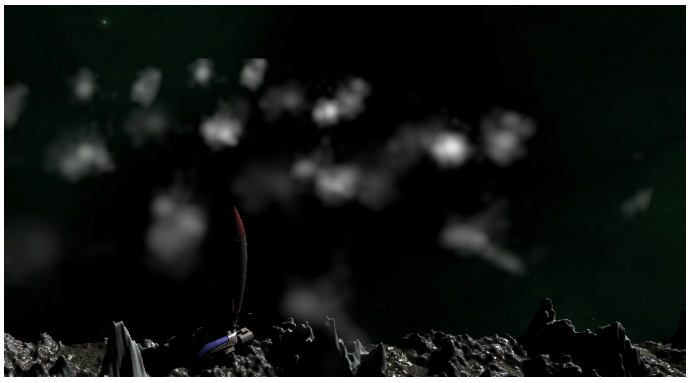

Figure 29: Good weather clouds

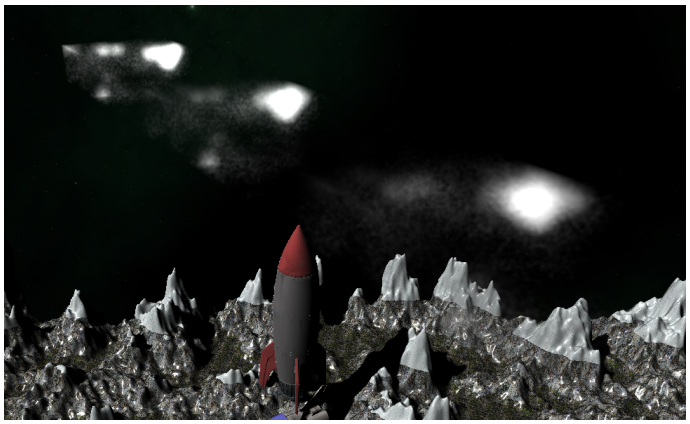

Figure 30: Cirrus style

Figure 31: Bad weather clouds

7.4 Loading Screen

Until now we only had one thread doing all work. This was causing very unpleasant effects. Heavy work like loading models has lead to a frozen window not able to render anything to screen and not able to handle user input. Therefore introducing multi-threading for handling time consuming calculations was very important for the game experience. I also introduced a loading screen (see fig. 32). Attached to the loading screen a progress bar (see fig 33) gives the user progress feedback. So when you have to make time consuming tasks (loading models, generate terrain) the user receives feedback.

Figure 32: Loading Screen [11]

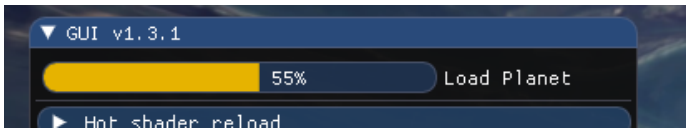

Figure 33: Progress Bar

7.5 General Bugs

This milestone was also about vanishing out all bugs hindering the player to experience a good game. I can't enumerate all my bugs due to the page count limit. In the following only a few important bug fixes in this milestone.

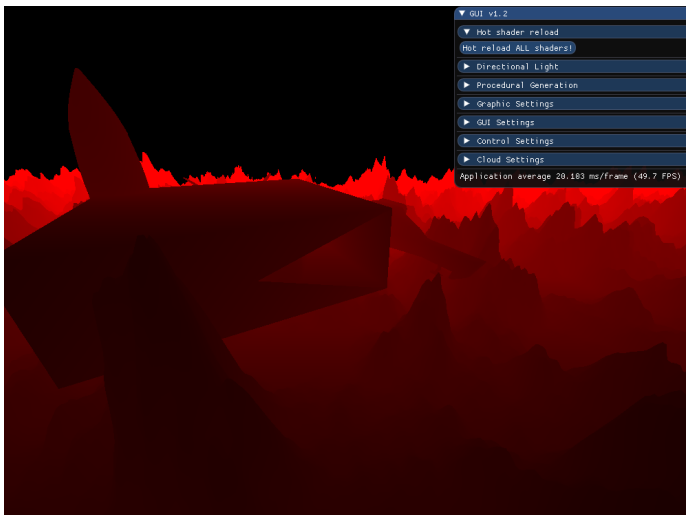

Figure 34: Wrong indices when drawing my AABB

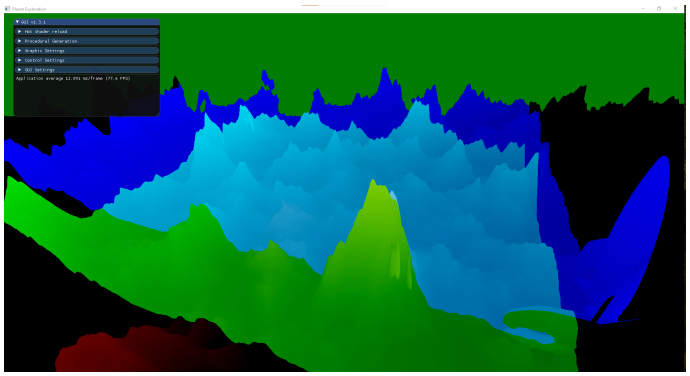

Figure 35: Wrong aspect ratio was causing disappearing shadows

Furthermore I had to deal with memory management issues. Especially all functionality regarding resetting graphic settings (like shadow map resolution, number of cascades) were causing memory problems. I learned a lot about memory management on CPU and GPU site.
It took a lot of time solving The problem of the occasionally disappearing skybox. In the end it was a problem originated when calculating shadows. Every fragment who does not contain any object (visible skybox) also has a fragment depth of 0.
One can now resize the window with an automatic adjusting of the viewport.
A wrong aspect ratio (width and height inverted) was causing my shadows to disappear on the sides of the screen (see 35).

References

[1] ashima. *Worley noise*. URL: https://github.com/ashima/webgl-noise (visited on July 5, 2021).

[2] Brotcrunsher. *load models*. URL: https://www.youtube.com/watch?v=KedrqATjoy0&list=PL58qjcU5nk8uH9mmlASm4SFy1yuPzDAH0&index=106 (visited on May 2, 2021).

[3] GLFW contributors. *GLFW*. URL: https://www.glfw.org/docs/3.0/group__input.html (visited on May 2, 2021).

[4] GUI contributors. *GUI*. URL: https://github.com/ocornut/imgui (visited on May 2, 2021).

[5] stb contributors. *Texture loading*. URL: https://github.com/nothings/stb (visited on May 2, 2021).

[6] tinyobjlader contributors. *tinyobjloader*. URL: https://github.com/tinyobjloader/tinyobjloader/ (visited on May 2, 2021).

[7] Erkaman. *Worley noise*. URL: https://github.com/Erkaman/glsl-worley (visited on July 5, 2021).

[8] Formski. *Skyboxes*. URL: https://www.gamedev.net/forums/topic/461747-atmospheric-scattering-sean-oneill---gpu-gems2/ (visited on May 11, 2021).

[9] Anton Gerdelan. *Compute Shader example*. URL: https://antongerdelan.net/opengl/compute.html (visited on July 5, 2021).

[10] GLEW. *GLEW*. URL: http://glew.sourceforge.net/ (visited on May 2, 2021).

[11] Golem. *Loading Screen*. URL: https://www.golem.de/news/raumfahrt-spacex-macht-sicherheitstest-bei-hoechster-belastung-2001-146124.html (visited on July 5, 2021).

[12] Jonas Heinle. *My github page with all projects*. URL: https://github.com/Kataglyphis (visited on May 2, 2021).

[13] Baldur Karlsson. *Renderdoc*. URL: https://renderdoc.org/ (visited on June 16, 2021).

[14] Lighthouse3D. *View Frustum Culling*. URL: http://www.lighthouse3d.com/tutorials/view-frustum-culling/geometric-approach-extracting-the-planes/ (visited on June 16, 2021).

[15] Wenzel Jakob Matt Pharr and Greg Humphreys. *Clouds*. URL: https://www.pbr-book.org/3ed-2018/Volume_Scattering/Phase_Functions (visited on July 8, 2021).

[16] nojima. *High quality noise functions*. URL: https://www.shadertoy.com/view/ttc3zr (visited on June 21, 2021).

[17] Nvidia. *NvidiaCascadedShadowMaps*. URL: https://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf (visited on June 16, 2021).

[18] Nvidia. *Vulkan Toturial*. URL: https://developer.nvidia.com/nsight-graphics (visited on May 2, 2021).

[19] Jen Lowe Patricio Gonzalez Vivo. *Clouds*. URL: https://thebookofshaders.com/12/ (visited on July 10, 2021).

[20] Dr. Christoph Peters. *Lecture notes in Interactive Computer Graphics*. Apr. 2020.

[21] piyushslayer. *Clouds in shadertoy*. URL: https://www.shadertoy.com/view/WddSDr (visited on June 20, 2021).

[22] Ingo Quilez. *Resource for many good topics*. URL: https://www.iquilezles.org/www/index.htm (visited on July 5, 2021).

[23] Andrew Schneider. *Clouds 3D based on Andrew Schneider*. URL: https://www.shadertoy.com/view/XlKyRw (visited on June 20, 2021).

[24] Andrew Schneider. *SIGGRAPHCOURSE2015*. URL: http://advances.realtimerendering.com/s2015/ (visited on June 20, 2021).

[25] SebastianLague. *Clouds*. URL: https://www.youtube.com/watch?v=4QOcCGI6xOU&t=97s (visited on July 5, 2021).

[26] Unknown. *Cascaded Shadow Maps*. URL: https://ogldev.org/ (visited on June 9, 2021).

[27] Unknown. *Skyboxes 3D Texture*. URL: https://doc.babylonjs.com/toolsAndResources/assetLibraries/availableTextures#cubetextures (visited on May 11, 2021).

[28] Unknown. *Vulkan Toturial*. URL: https://vulkan-tutorial.com/Loading_models (visited on May 2, 2021).

[29] Unreal. *SIGGRAPHCOURSE2013*. URL: https://blog.selfshadow.com/publications/s2013-shading-course/ (visited on June 16, 2021).

[30] Joey de Vries. *Learn OpenGl*. URL: https://learnopengl.com/ (visited on Apr. 30, 2021).

[31] Joey de Vries. *Skyboxes*. URL: https://learnopengl.com/Advanced-OpenGL/Cubemaps (visited on May 11, 2021).