Project ”No Man’s Galleon”

Realtime Rendering

186.140

Simon Derflinger and Maximilian Scheiber

Matriculation Numbers:
12244572 and 12144091
Study Code: 066 932
Semester: WS 2024

18. January, 2024

1 Description

Here we present our demo "No Man's Galleon". It is implemented in C++ and OpenGL (4.6 Core Profile). The technical setup is as proposed in the previous submission.
The demo was tested on a NVIDIA GeForce RTX 3060 Laptop GPU.

Our basic framework uses a main file "Application.cpp" which contains all the setup logic for the rendering context and the main rendering loop. Additionally, we wrote some helper classes to encapsulate certain elements of the rendering context like the camera, textures, buffers etc.

We use the following additional libraries:

- assimp for model loading
- stb_image for image loading
- irrKlang for audio (though we decided to not include sound in the final demo because of copyright concerns)
- glad for opengl function loading
- glfw for window context management
- glm for vector, matrix and other mathematical operations

The demo implements the following features and effects:

- A free floating camera, which is controllable and also has auto movement that follows a predefined path
- Model loading supporting Vertex-[Position/Normals/Texture Coordinates/Tangent/Bitangent] and [Diffuse/Specular/Normal/Height/Glow]-Textures. The Tangent and Bitangent Vectors get computed by assimp in the loading process.
- Blinn-Phong illumination for Point Lights
- Normal-Mapping using precomputed Tangent and Bitangent Vectors
- Omni-Directional Shadowmapping for Point Lights
- A water effect created with dynamic environment mapping, normal mapping and a distortion map
- HDR with tone mapping
- Volumetric light froxel based approach
- Tessellation with beziér triangle interpolation that affects the ships sails
- Wind-effect that affects the ships sails

All effects above support multiple point lights, however, we ran into slight performance issues, with only having about 50FPS. Therefore, we only activated two lamps. Additionally, the moonlight can be turned off, if more FPS are required.
The Omni-directional Shadowmapping is done based on Cube map Depth Textures. The shadows can be observed on the ship (they also move together with the sails), on the island, also under water and in the reflection on the water. The Shadows are filtered by sampling the shadowmap multiple times per fragment and taking the mean value.
The water effect is done using dynamic environmental mapping. The water surface is modeled by a simple flat plane. On this plane, 2 Textures, one depicting the underwater-surface and the other one depicting the water-reflection, are rendered and blended together. To create the waves, the textures get distorted and a normal map is applied to create the highlights.
The volumetric light is implemented as a froxel-based approach, using a light-injection compute shader that calculates the lights' influence to each froxel. Additionally, we use a raymarching compute shader to traverse the voxel grid and accumulate the light values for each froxel. In our forward rendering pipeline we apply these light values in the fragment shader to each object and the skybox in the scene. The density and the anisotropy of the light can be modified using the keybinds below. However, the raw implementation of the fog was strongly affected by noise. To reduce the noise, we sample from a blue noise texture to introduce a small jitter to the world position. Additionally, we blur the output in the fragment shader using three 1D Gaussian blurs.
The wind effect affects the sails of the ship and is implemented in a geometry shader that moves the vertices along a modified sine wave in the z direction. We created a wind texture to determine the influence of the wind on the vertices. Vertices close to a fixed point should move less than vertices with a greater distance to such a fixation point. As the model has a small vertex count, we decided to implement tessellation that affects only the sails of the ship. We use Beziér triangles as interpolation method in the tessellation evaluation shader. The effect can be observed best by looking at the sails from the side and toggling the tessellation.
Every object in the scene is rendered with regards to the point lights (lantern light and moonlight). The model of the moon is the only exception, it is rendered with a constant color independent of the lights and the rest of the scene. It is also left out for the reflection in the water.
Some effects are toggleable and some parameters can be changed (See Keybinds section).

1.1 Keybinds

The following inputs can be used to configure the demo.

- WASD: Camera movement
- Mouse Position: Viewing direction
- Mousewheel: FOV
- Shift: Increase camera movement speed

- CTRL: Decrease camera movement speed
- C: Toggle automatic camera movement (default: on)
- R: Restart camera auto pathing
- M: Toggle moonlight (default: on)
- T: Toggle tessellation of the ships sails (default: on)
- U: Toggle the wind effect of the ships sails (default: on)
- 1,2: Decrease/Increase tone mapping exposure
- 3,4: Decrease/Increase tone mapping gamma
- V: Toggle the volumetric light (default: on)
- B: Toggle blue noise filter of volumetric light (default: on)
- F: Toggle gaussian blur of volumetric light (default: on)
- Keypad -/+: Decrease/Increase volumetric fog density
- Keypad 6/9: Decrease/Increase volumetric fog anisotropy