Documentation Mazeballer

Mazeballer

Lea Aichner 01226600
Julian Pegoraro 01225472

Description of the implementation

The implemented effects can be found in the section “Effects”. The camera movement was implemented without any specification. The desired positions as well as the pitch and the yaw of the camera are selected at the beginning of the program. The duration of each camera movement as well. Than by using the delta time, two positions are interpolated. When the last position is reached, this position is interpolated with the first one and the whole process starts again. Mipmapping was implemented, by telling it to the graphics card when adding the textures.

Additional libraries

Object loader Assimp (http://assimp.org/)
Image loader stb_image (http://nothings.org/)
Render on the Screen (http://www.glfw.org/download.html)
Manage OpenGL Drivers (https://github.com/Dav1dde/glad)
OpenGL Mathematics (https://glm.g-truc.net/0.9.8/index.html)

Effects

Reflection/Refraction:
- Slides of the lecture
- Old code snippets for cubemap

Omnidirectional Shadow Maps:
- https://learnopengl.com/#!Advanced-Lighting/Shadows/Shadow-Mapping
- https://learnopengl.com/#!Advanced-Lighting/Shadows/Point-Shadows
- Slides of the lecture
- https://users.cg.tuwien.ac.at/husky/RTR/OmnidirShadows-whyCaps.pdf
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.19.6747&rep=rep1&type=pdf

Normal Mapping:
- Slides of the lecture
- https://learnopengl.com/#!Advanced-Lighting/Normal-Mapping
- http://ogldev.atspace.co.uk/www/tutorial26/tutorial26.html

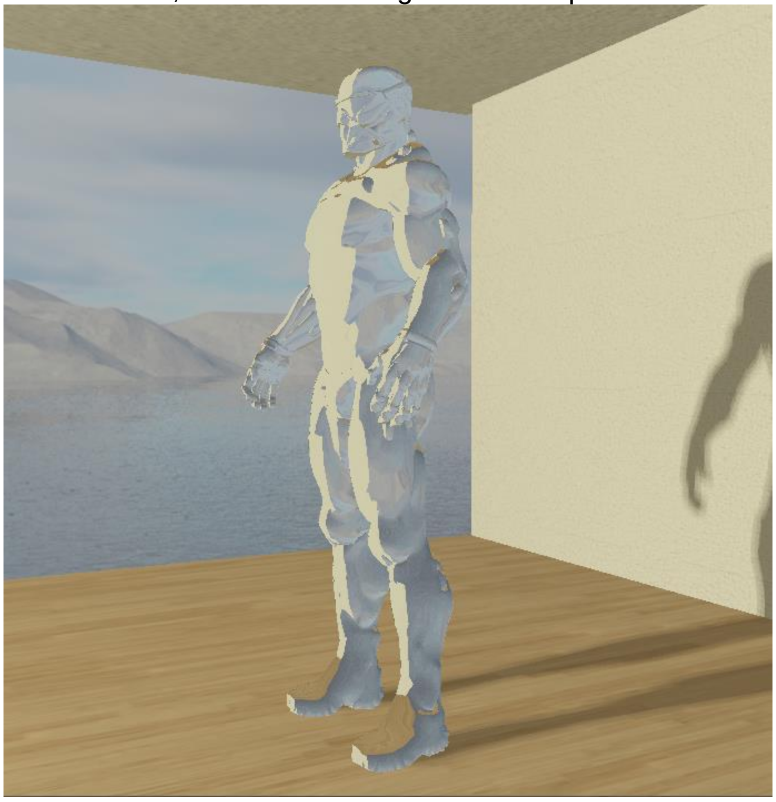
Bloom:
- http://rastergrid.com/blog/2010/09/efficient-gaussian-blur-with-linear-sampling/
- https://learnopengl.com/#!Advanced-Lighting/HDR
- https://learnopengl.com/#!Advanced-Lighting/Bloom

Skybox:
- https://www.khronos.org/opengl/wiki/Skybox
- Slides of the lecture (cubeMaps)

Phong Shading:
- http://graphics.wikia.com/wiki/Phong_shading

How are the Effects implemented?

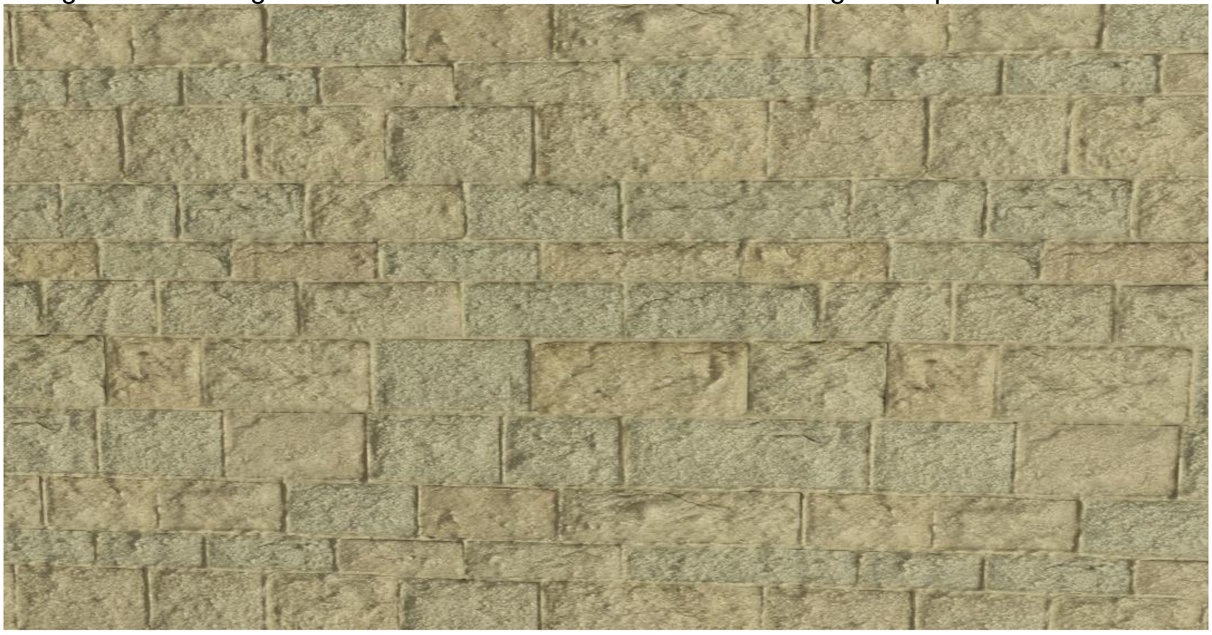
Reflection/Refraction: Since we are using Shadows which are moving in the scene, the cubemap has to be updated every frame, but the resolution is very low. Snells law can be adapted by the user on the refraction part. This is done by creating an uniform in the fragment shader which then uses the refract method in order to calculate the refracted vector and the reflect function to calculate the reflection vector. This vector is used to get the color at the cubemap which is calculated. The vertex shader is the same for both effects. The normal and the position are calculated and given to the fragment shader. The position of the vertex is simply calculated with the model, the view and the projection matrix. To calculate the cubemap, a camera is needed with a FOV of 90 degrees and a ratio of 1. Each face is then rotated in an other direction. For each side of the cubemap, the scene and the shadows are rendered, but no effects e.g. normal maps.



Omnidirectional Shadow Maps: In the scene, you can see a light source that moves from one side of the room to the other. The color of the light source changes over time. The point light casts an omnidirectional shadow. You can see the shadow on the table and chairs and on the model of the man. To render the shadows, the whole scene is first rendered from the position of the light source. The distance from a fragment position to the light source is saved in a depth map. When rendering the final scene, the application compares the current distance to the light source with the distances saved in the depth map. Because we are dealing with point lights we need to use cube maps. To transform the vertices to the space of the cube map we used a geometry shader.

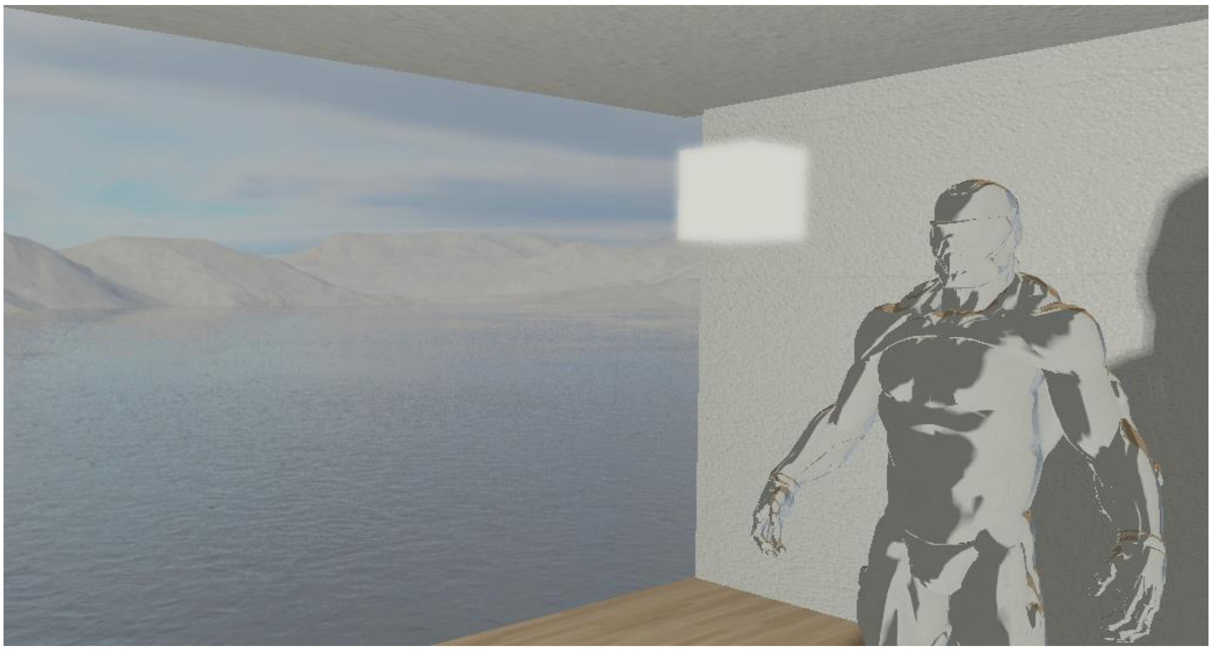
We also tried to implement perspective shadow maps, unfortunately we did not have time to finish it correctly. In comparison to normal shadow maps, when computing perspective shadow maps, the depth information is computed in the post perspective space.

Normal mapping: You can see the normal mapping on the room model, especially on the brick wall and when the light changes position. But also on the other walls, on the floor and on the ceiling normal mapping is applied. The program simply reads the information of the orientation of the normals out of a normal map. Then the position of the light source and the position of the camera are transformed to the Tangent Space. As a result, the normals of the triangles are looking to different directions. This influences the light computation.



Phong Shading: The light source illuminates the scene using a phong shading. The color of the light changes over time. Phong Shading is a simple BRDF model, where the diffuse and the specular part of a material can be implemented very easily.

Bloom: The bloom effect is applied to the model of the light source. The scene is rendered onto two frambuffers. On the first one the whole scene is rendered as usual, on the second one only those points which extend a specific threshold are rendered. This second one is than used as a texture for a blur shader where the image is blurred in vertical and in horizontal direction. This is done separately for performance reasons. The previously rendered scene and the blurred one are then added together in order to obtain a bloom effect.



Skybox: At initialization, the six faces of the skybox are loaded and a cubemap is created. by putting each of the face onto another GL_TEXTURE_CUBE_MAP_POSITIVE position of a GL_TEXTURE_CUBE_MAP. This generates a 3D texture where every point of the texture can be selected with a vertex. After rendering the whole scene, the skybox is rendered by the glDethFunc to GL_LEQUAL. By doing so the skybox is rendered to those pixels where nothing was rendered before, this saves a lot of rendering time.

Tools to create Models

The room was made using Blender. The human model was taken from an introduction tutorial from this site: https://learnopengl.com/#!Model-Loading/Mesh
The chair and the table are taken from this site: https://free3d.com/3d-models/texture

Controls

R: change between reflection and refraction, when in debug mode.
Q: go to debug mode (stop camera movement, can use R).
F: Print FPS on the console
C: Stop and start the changing of the color of the camera
W,A,S,D: can move camera when in debug mode.
Mouse: can change viewing direction when in debug mode.
Arrow Up/Down: can change refraction index when in debug mode.

Graphics card

Nvidia GeForce GT 640M