Introduction to Computer Graphics and Animation

Exercise 4 of 5

Prof. Dr. Dennis Allerkamp – December 5, 2024

4.1 Representation of the Earth

In this task, you are to combine different textures with lighting information. The required image data can be found in the course materials (originally from NASA Visible Earth). You will also be provided with the file `earth.obj`.

- Create a suitable framework program for the example, which should load the OBJ file and display it in the scene. You should now see a sphere or a disc on the screen.
- Use the texture coordinates of the OBJ file to overlay the texture from `earth_day.png` on the sphere.
- Use the model matrix to continuously rotate the object around the y-axis.
- Also create a texture object for `earth_night.png`. Add another sampler to the fragment shader, so that a separate texture unit can be used for each texture. Bind each texture to a different texture unit before drawing the object. Test whether you can actually access all textures in the fragment shader by trying them all once.
- Calculate the diffuse lighting component in the shaders. However, here we do not need a distinction of the different color components but only a brightness value. Test whether you have implemented the lighting correctly (e.g., you can display the brightness values `light` with `fragColor = vec4(light, light, light, 1.0);` as grayscale).
- Use the function `vec4 mix(vec4 x, vec4 y, float a)` to mix the color values from `earth_day.png` with the color values from `earth_night.png`. Use the calculated brightness values as the third parameter.
- Now use the texture `earth_ocean_mask.png` as a shine texture. By multiplying the specular light component with the value of the texture (either 0 or 1), it is ensured that only water surfaces reflect the light.
- Overlay the cloud texture `earth_clouds.png` over the other textures. For this, you have to implement the color mixing taking into account the alpha value of the cloud texture in the fragment shader.

Your result could now look like this:

4.2 Skybox

In this task, the environment of the scene should be represented by a Cube Map texture. You can find the textures in the course materials (originally from https://learnopengl.com/img/textures/skybox.zip)

- Implement a simple box. For this, you can copy the code from earlier exercises. However, only the positions are needed.
- Implement a camera ride around the box. You can also take this code from earlier exercises.
- Create a Cube Map with the previously downloaded textures and use it as a texture for the box. You use the position in object coordinates as the texture coordinates, which you pass directly from the vertex shader to the fragment shader as a varying variable. You use the texture coordinates in the fragment shader to read the color of the fragment from the Cube Map.
- The box should now be placed not in front of the camera but around the camera. This is easiest by setting the w-coordinate of the position to 0.

Your vertex shader might look like this:

```
uniform mat4 uModel;
uniform mat4 uView;
uniform mat4 uProj;
in vec3 aPosition;
out vec3 vTexCoord;
void main() {
  vTexCoord = aPosition;
  vec3 viewPos = (uView * uModel * vec4(vPosition, 0.0)).xyz;
  gl_Position = uProj * vec4(viewPos, 1.0);
}
```

And the fragment shader might look like this:

```
uniform samplerCube skybox;
```

```
in vec3 vTexCoord;
out vec4 fragColor;
void main() {
  fragColor = texture(skybox, fTexCoord);
}
```

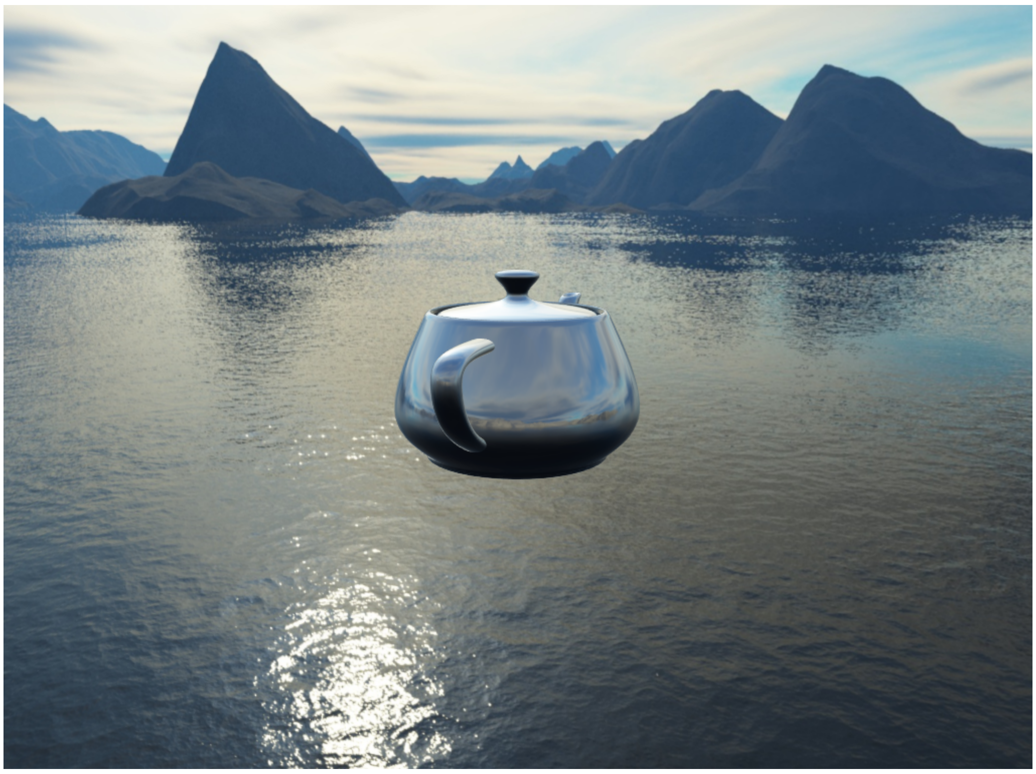
If you have difficulties, you can find an explanation of cube maps under LearnOpenGL - Cubemaps.

4.3 Environment Mapping

Now an object that is (perfectly) reflective should be added to the scene.

- Add another object to the scene. Note that a separate shader is needed for this. With the function `useProgram(…)`, you can change the shader program.
- Add a sampler for the Cube Map texture to the fragment shader of the new object.
- Since the environment of the scene is independent of the camera or projection, the texture coordinates relevant for the reflection must be calculated in the world coordinate system. For this, you need the direction $\vec{v}$ to the camera in world coordinates. This can be calculated from the view matrix V. It applies $V\vec{v} = \vec{z}$, where $\vec{z}$ is a vector pointing in the direction of the z-axis. (This results from the fact that the camera in eye point coordinates points in the direction of $-\vec{z}$.) The required vector $\vec{v}$ can thus be calculated as follows: $\vec{v} = V^{-1}\vec{z}$. Pass $\vec{v}$ as a uniform variable to the vertex shader.
- The direction to the camera in world coordinates is passed in the vertex shader as a varying variable unchanged to the fragment shader. There you can now calculate the texture coordinate via a reflection on the surface normal (analogous to specular lighting).
- Use the texture coordinate to read the color of the fragment from the Cube Map.

Your result could now look like this:



4.4 Render to Texture

For this optional task, you should acquire the necessary knowledge for using frame buffers yourself. You can find a good guide at LearnOpenGL - Framebuffers. Drawing into textures is an important prerequisite for several techniques (e.g., drop shadows).

- Create a Frame Buffer Object with a texture for the colors and a Render Buffer Object for the depth information.
- Draw the teapot into the created Frame Buffer Object.
- Then draw a box into the Frame Buffer of the Canvas. Use as texture the texture of the created Frame Buffer Object.

Your result could now look like this: