The exercises will take place in room G40 in Mühlenpfordtstrasse 23. Your y-account is sufficient to login and access all tools. Ctrl+Alt+T gives you a terminal and g++ is your GNU C++ compiler.

Each week you must complete the assignments and hand in your commented source code for the practical tasks, as well as your solutions to the theoretical tasks (with drawings/formulas). Please use different colors in your drawings and also make sure that formulas are recognizable in your source code. Your group must present the completed assignments on each Friday, 9:45.

From now on it is also required that you send me your source code (without the data folder) before Friday 9:45!

6.1 Environment Map Acquisition (20 Points)

In the lecture, you have learned about environment maps and their acquisition. You have already implemented a simple environment map in a previous exercise. Now it is time to create your own by implementing an OmnidirectionalCamera. This type of camera casts rays in every direction. The x and y coordinates that are passed to the camera in OmnidirectionalCamera::castRay are simply spherical coordinates, i.e. \( uv \) coordinates on your environment map. So all you have to do is to cast the ray in that direction. Take a look at camera/omnidirectionalcamera.h and implement the camera. If you have implemented everything correctly (including the other exercises), it will look like this:

![Environment Map Image]

6.2 Textures (10 Points)

In reality, e.g. in games, textures are used to make a surface look more geometrically complex. You have already used a texture as an Environment Map, where you did a simple mapping based on viewing direction. In order to map textures onto objects, texture coordinates are stored at each vertex of each triangle. These values are then interpolated (similar to the normals on a SmoothTriangle) to find the correct texel for a surface point. Implement TextureTriangle::uvFromRay(Ray const& ray) so that it calculates the correct texel position from the \( uv \) coordinates stored in the ray. You will also have to add an additional value TEXTURED to the TriangleStyle enum in the ObjModel and adjust the code so that it can also load textered triangles.
6.3 Material Shader (0 + 20 + 10 + 20 + 30)

You have already implemented various individual shaders. We now want to create a new shader, that combines various techniques you have learned before and uses image maps for additional detail.

**Diffuse Map** The diffuse map, using simple lambertian shading is already implemented in the MaterialShader, take a closer look at this implementation as it will give you a good idea on how to implement the other maps.

**Alpha Map** An alpha map describes the local opacity of the object. White means opaque and black means transparent. Implement alpha mapping in the MaterialShader. Remember that you only have to propagate those rays, where the object is not fully opaque. Also keep in mind that the reflection map does also propagate rays, so you cannot just change the ray passed to the function, but have to create a copy of it. If you have implemented this part correctly, you should see a cloud layer around the planet earth.

**Specular Map** A specular map is a grayscale image that represents the specularity at each point on the object. White means glossy and black means diffuse. Implement the specularity term you already know from the phong shader and use the specular map to scale the result. If you implemented this part correctly you should see the earths oceans as very glossy, and the continents as very diffuse.

**Normal Map** You may have noticed that the moon in the foreground looks quite boring and flat. In order to add additional detail we want to add a normal map. In a normal map each pixel color describes the normal at the corresponding position. Note that you do not have negative colors, instead normal values are mapped to the color channels in such a way that for each component $[0, 1]$ in color space corresponds to $[1, 1]$ in normal space. Additionally, you will have to transform your normal into tangent space. Don’t worry, most of that is already done for you. If you implemented this correctly you should see detailed craters appear on your moon.

**Reflection Map** A reflection map is a grayscale image that represents the reflective property of the surface. White means a perfect mirror, whereas black means no reflection. You already know how to implement a mirror shader, and the value in the reflection map is simply a linear interpolation factor between the color of the mirrored ray and the color of the surface. Remember that you have to use the normal map when calculating the reflected ray!
6.4 Bilinear Filtering (20 Points)

You may have noticed that your textures are pixelated, especially near the camera, where one texel covers multiple pixels on the screen. To alleviate this, we will implement bilinear filtering. This allows us to linearly interpolate between texels and thus get a smoother result.

Take a look at `Texture::color(float u, float v)` in `common/texture.cpp` and implement bilinear filtering. If everything is implemented correctly, you get smooth textures.