Laboratory work 10

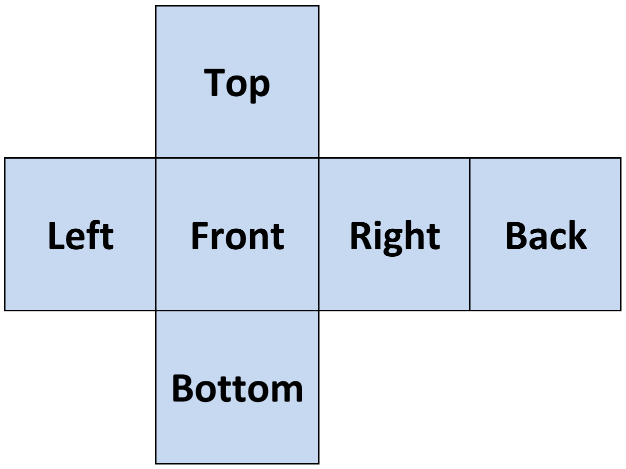
1 Objectives

This laboratory presents you with basic notions about the concept of cubemaps (that are used for defining skyboxes) and reflection properties of 3D objects.

2 Theoretical background

2.1 Cubemap

In many situations we want to display a “background” image representing the part of the 3D scene that we didn’t model. We can accomplish this by “including” our 3D scene of objects inside a cube and map different textures on each face. This cube is typically referred as a skybox and consists of 6 textures following this pattern:



The OpenGL representation of this kind of texture is a **cubemap**, containing 6 individual 2D textures. By combing these textures together and mapping them on a unit cube, we can very easily sample a texture value using a direction vector. This vector can be retrieved from the interpolated vertex position of the cube.

To create a cubemap we have to generate a texture id and bind it to the GL_TEXTURE_CUBE_MAP target.

```
GLuint textureID;
glGenTextures(1, &textureID);
glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);
```

For each face of the cubemap, we have to call the glTexImage2D function and set a proper target parameter specific to the current face:

Target	Orientation
GL_TEXTURE_CUBE_MAP_POSITIVE_X	Right
GL_TEXTURE_CUBE_MAP_NEGATIVE_X	Left
GL_TEXTURE_CUBE_MAP_POSITIVE_Y	Top
GL_TEXTURE_CUBE_MAP_NEGATIVE_Y	Bottom
GL_TEXTURE_CUBE_MAP_POSITIVE_Z	Back
GL_TEXTURE_CUBE_MAP_NEGATIVE_Z	Front

We can linearly increment the target in order to set all the textures of the cubemap:

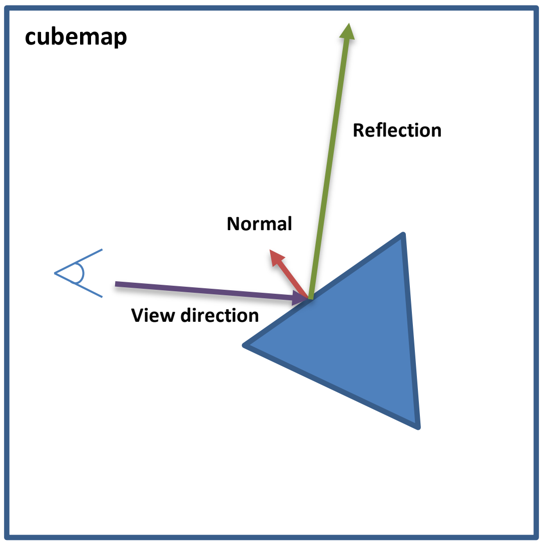
```
for(GLuint i = 0; i < skyBoxFaces.size(); i++)
{
	image = stbi_load(skyBoxFaces[i], &width, &height, &n, force_channels);
	if (!image) {
		fprintf(stderr, "ERROR: could not load %s\n", skyBoxFaces[i]);
		return false;
	}
	glTexImage2D(
		GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0,
		GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);
}
```

In the fragment shader, we use a different sampler in this case (**samplerCube**) but we use the same function (texture) to sample the color value. Here, the textureCoordinates variable represents the direction vector.

```
#version 410 core

in vec3 textureCoordinates;
out vec4 color;

uniform samplerCube skybox;

void main()
{
    color = texture(skybox, textureCoordinates);
}
```

2.2 Reflections

Having now a surrounding environment (represented by the cubemap) we can specify reflective properties to various objects. The reflection is based on viewer’s angle. This reflection vector can be computed using the GLSL function called **reflect**. We have to specify the normal vector and the view direction vector. The resulting vector is used to sample from the cubemap texture.



```
vec3 viewDirectionN = normalize(viewDirection);
vec3 normalN = normalize(normal);
vec3 reflection = reflect(viewDirectionN, normalN);
vec3 colorFromSkybox = vec3(texture(skybox, reflection));
```

3 Tutorial

Start by downloading the resources and include them in your project. The SkyBox class will be used to load and draw the cubemaps.

Add in a vector the textures (namely their path) that will be mapped on the cubemap. Create a void function called **initSkybox()** for the following code and call it after initializing the objects in your scene.

```
std::vector<const GLchar*> faces;
faces.push_back("skybox/right.tga");
faces.push_back("skybox/left.tga");
faces.push_back("skybox/top.tga");
faces.push_back("skybox/bottom.tga");
faces.push_back("skybox/back.tga");
faces.push_back("skybox/front.tga");

mySkyBox.Load(faces);
```

Define the following global variables that will represent our skybox and the shader used to draw it.

```
gps::SkyBox mySkyBox;
gps::Shader skyboxShader;
```

In the **initShaders()** function, load the skybox shader.

```
skyboxShader.loadShader("shaders/skyboxShader.vert", "shaders/skyboxShader.frag");
skyboxShader.useShaderProgram();
```

In the **renderScene()** function call the drawing function of the skybox after you called all the other drawing functions of the rest of the objects. If your project has shadows enabled using the shadow mapping technique, make sure not to draw the skybox in a depth map render pass, but rather in the normal render pass.

```
mySkyBox.Draw(skyboxShader, view, projection);
```

4 Assignment

- Follow the tutorial in order to display a skybox.
- Load another 3D object in your scene and compute the object's color taking into account also the reflections coming from the skybox.