Water Surface

This document will explain how I finish this project. In this project, what I have done are sine wave and skybox reflection.

First of all, I defined a class “Drawable” that stored all information of an object that need to be drawn, e.g. wave.

```
#pragma once

#include "RenderUtilities/Shader.h"

class Drawable {
public:

    Shader* shader = nullptr;

    GLuint vao;
    GLuint vbo[4];  // position, normal, texture, color
    GLuint ebo;
    unsigned int element_amount;

    GLuint fbo;
    GLuint textures[4]; //attach to color buffer
    GLuint rbo;
};
```

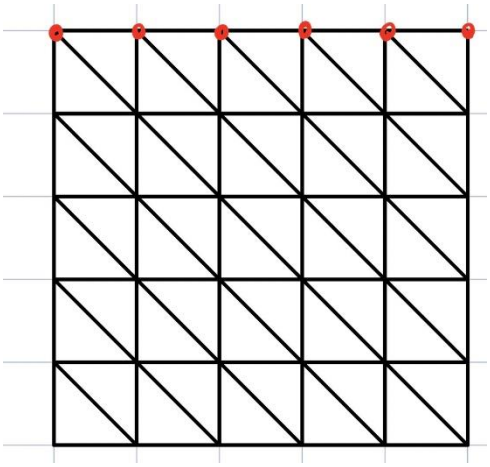
1. Sine wave

1-1. Attribute

I cut the plane into 10 x 10 grid and define the vertices information as below.

```
// Position attribute
GLfloat  vertices[11 * 11 * 3] = { 0.0f };
int index = 0;
for (float x = -5.0f; x <= 5.0f; x += 1.0f) {
    for (float z = -5.0f; z <= 5.0f; z += 1.0f) {
        vertices[index * 3 + 0] = x;
        vertices[index * 3 + 1] = 0.0f;
        vertices[index * 3 + 2] = z;

        index++;
    }
}
```



```
// Normal & Color attribute
GLfloat  normal[11 * 11 * 3] = { 0.0f };
GLfloat  mycolor[11 * 11 * 3] = { 0.0f };
for (int i = 0; i < 363; i++) {
    if ((i + 1) % 3 == 0) {
        mycolor[i] = 1.0f;
    }
    else if ((i + 1) % 3 == 1) {
        mycolor[i] = 0.8f;
    }
    else {
        mycolor[i] = 0.8f;
        normal[i] = 1.0f;
    }
}
```

```
//Element attribute
GLuint element[10 * 10 * 6] = { 0 };
int i = 0;
for (int row = 0; row < 10; row++) {
    for (int col = 0; col < 10; col++) {
        int start = row * 11 + col;

        element[i + 0] = start;
        element[i + 1] = start + 1;
        element[i + 2] = start + 12;

        element[i + 3] = start;
        element[i + 4] = start + 12;
        element[i + 5] = start + 11;

        i += 6;
    }
}
```

1-2. Shader

In sinewave.vert (vertex shader for sine wave), I referred https://jayconrod.com/posts/34/water-simulation-in-glsl and calculated the height of the wave according to the x and z.

```
const float pi = 3.14159f;
float wavelength = 1.0f * pi;
float speed = 3.0f;
float amplitude = 0.5f;
float angle = fract(sin(pi)*100000.0); // simulate random
vec2 direction = vec2(cos(angle), sin(angle));
```

```
float wave(float x, float y) {
    float frequency = 2.0 * pi / wavelength;
    float phase = speed * frequency;
    float theta = dot(direction, vec2(x, y));

    return amplitude * sin(theta * frequency + time * phase);
}
```

and this is the final position of the vertex

```
vec4 mywave = vec4(position.x, wave(position.x, position.z), position.z, 1.0f);
v_out.position = vec3(u_model * mywave);
```

To perform the right lighting, we have to adjust the normal of the vertices.

```
vec3 waveNormal(float x, float y) {
    float frequency = 2.0 * pi / wavelength;
    float phase = speed * frequency;
    float theta = dot(direction, vec2(x, y));

    float dx = amplitude * direction.x * frequency * cos(theta * frequency + time * phase);
    float dy = amplitude * direction.y * frequency * cos(theta * frequency + time * phase);

    vec3 n = vec3(-dx, 1.0, -dy);
    return normalize(n);
}
```

```
vec3 mynormal = waveNormal(position.x, position.z);
v_out.normal = mat3(transpose(inverse(u_model))) * mynormal;
```

In sinewave.frag (vertex shader for sine wave), I referred https://learnopengl.com/Advanced-OpenGL/Cubemaps to make reflection of the skymapping. I first got the color of corresponding sky color and mix it with the color of water.

```
void main()
{
    vec3 I = normalize(f_in.position - camera_pos);
    vec3 reflect_ray = reflect(I, normalize(f_in.normal));

    vec4 reflect_color = vec4(texture(skybox, reflect_ray).rgb, 1.0);
    vec4 water_color = vec4(f_in.mycolor, 0.5f);

    f_color = reflect_color * water_color;
}
```

1-3 Draw

I put all the thing about drawing the sine wave in the drawSineWave function.

```
void TrainView::drawSineWave()
```

Apart from the basic setting we need when we draw things, I enable GL_BLEND to perform transparency of the water.

```
// draw water surface
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glEnable(GL_LIGHTING);

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_CUBE_MAP, this->skybox->textures[0]);
glUniform1i(glGetUniformLocation(this->skybox->shader->Program, "skybox"), 0);

glDrawElements(GL_TRIANGLES, this->sineWave->element_amount, GL_UNSIGNED_INT, 0);

glDisable(GL_BLEND);
```

2. Skybox

2-1 Attribute

Building the skybox, I referred https://github.com/VictorGordan/opengl-tutorials/blob/main/YoutubeOpenGL%2019%20-%20Cubemaps%20%26%20Skyboxes/Main.cpp .

First, I need to assign the information about the vertices.

```
float skyboxVertices[] = {
    //   Coordinates
    -1.0f, -1.0f,  1.0f,//        7--------6
     1.0f, -1.0f,  1.0f,//       /|       /|
     1.0f, -1.0f, -1.0f,//      4--------5 |
    -1.0f, -1.0f, -1.0f,//      | |      | |
    -1.0f,  1.0f,  1.0f,//      | 3------|-2
     1.0f,  1.0f,  1.0f,//      |/       |/
     1.0f,  1.0f, -1.0f,//      0--------1
    -1.0f,  1.0f, -1.0f
};
```

```
unsigned int skyboxIndices[] = {
    // Right
    1, 2, 6,
    6, 5, 1,

    // Left
    0, 4, 7,
    7, 3, 0,

    // Top
    4, 5, 6,
    6, 7, 4,

    // Bottom
    0, 3, 2,
    2, 1, 0,

    // Back
    0, 1, 5,
    5, 4, 0,

    // Front
    3, 7, 6,
    6, 2, 3
};
```

In this part, I read my images using stb library.

```
for (unsigned int i = 0; i < 6; i++)
{
    int width, height, nrChannels;
    unsigned char* data = stbi_load(faces[i].c_str(), &width, &height, &nrChannels, 4);
    if (data){
        stbi_set_flip_vertically_on_load(false);
        glTexImage2D
        (
            GL_TEXTURE_CUBE_MAP_POSITIVE_X + i,
            0,
            GL_RGBA,
            width,
            height,
            0,
            GL_RGBA,
            GL_UNSIGNED_BYTE,
            data
        );
        stbi_image_free(data);
    }
    else{
        std::cout << "Failed to load texture: " << faces[i] << std::endl;
        stbi_image_free(data);
    }
}
```

2-2 Shader

In vertex shader, we simply transform the position to the screen and ignore the depth information (z).

```
#version 330 core
layout (location = 0) in vec3 aPos;

out vec3 texCoords;

uniform mat4 projection;
uniform mat4 view;

void main()
{
    vec4 pos = projection * view * vec4(aPos, 1.0f);
    gl_Position = pos.xyww;
    texCoords = vec3(aPos.x, aPos.y, -aPos.z);
}
```

In fragment shader, we only assign the texture and its coordinates.

```
#version 330 core
out vec4 FragColor;

in vec3 texCoords;

uniform samplerCube skybox;

void main()
{
    FragColor = texture(skybox, texCoords);
}
```

2-3 Draw

I put all the thing about drawing the skybox in the drawSkybox function.

```
void TrainView::drawSkybox()
```

First, I change the depth function as GL_LEQUAL, since for the skybox, the depth always remains the same.

```
glDepthFunc(GL_LEQUAL);
```

For the projection matrix and model_view matrix, we can get them from OpenGL. There is only one thing important, and that is the last row and the last column of the model_view matrix should be discarded because they will affect the translation.

```
// view matrix
glm::mat4 view_matrix;
glGetFloatv(GL_MODELVIEW_MATRIX, &view_matrix[0][0]);
view_matrix = glm::mat4(glm::mat3(view_matrix)); // get rid of last row & col (which affect translation)
glUniformMatrix4fv(glGetUniformLocation(this->skybox->shader->Program, "view"), 1, GL_FALSE, glm::value_ptr(view_mat

// perspective matrix
glm::mat4 projection_matrix;
glGetFloatv(GL_PROJECTION_MATRIX, &projection_matrix[0][0]);
glUniformMatrix4fv(glGetUniformLocation(this->skybox->shader->Program, "projection"), 1, GL_FALSE, glm::value_ptr(pr
```

Reference

Water simulation in GLSL
https://jayconrod.com/posts/34/water-simulation-in-glsl

Cubemaps
https://learnopengl.com/Advanced-OpenGL/Cubemaps

YoutubeOpenGL 19 - Cubemaps & Skyboxes
https://github.com/VictorGordan/opengl-tutorials/blob/main/YoutubeOpenGL%2019%20-%20Cubemaps%20%26%20Skyboxes/Main.cpp