Project 2 report

By Fedor Ivachev 费杰 2019280373

Terrain Engine

1) Render a skybox and the sea.

2) Simulate the waves on the sea.

3) Load the terrain model and render it with texture.

4) Implement the reflection of sky and terrain in water.

5) Can wander around the scene by keyboard.

The required materials and goal effect refer to the attachment project2.zip.

Submit code, report and screen recording of program running.
Use modern (core-profile) opengl taught in tutorial.

In my solution each of these tasks are implemented. Also, extra features are added: (1) multitexture model of terrain, (2) own method of plane clipping in the shader, (3) speeding up and slowing down and (4) collision detection. The extra features (1) – (3) can be found as requirement in TerrainDoc.pdf, while feature (4) cannot.

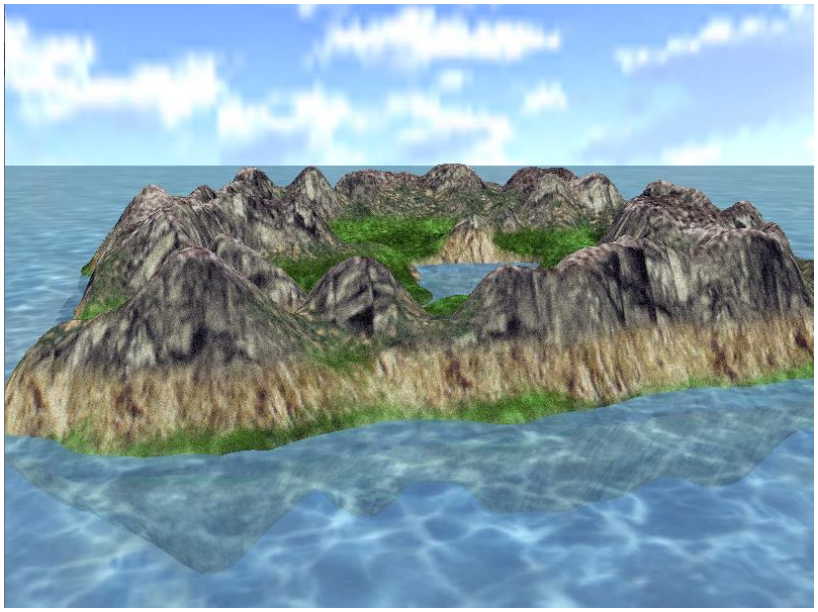
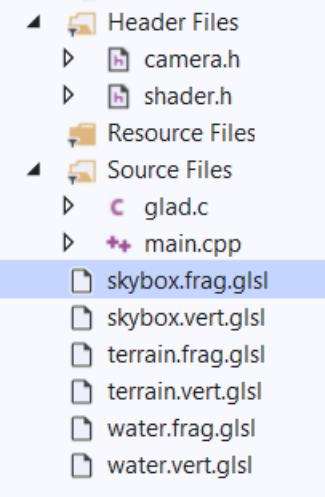
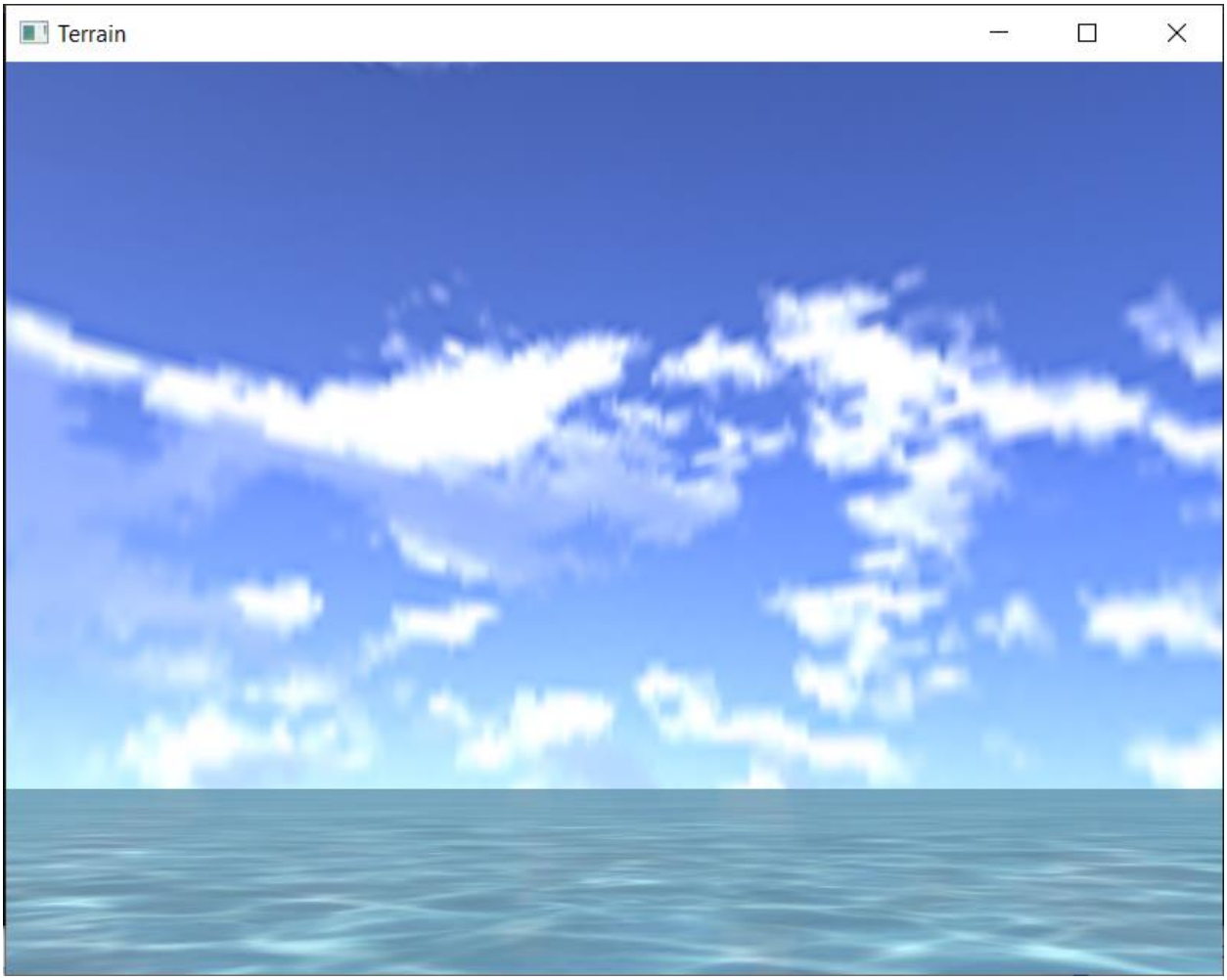


Figure 1. The whole effect

The program structure:



Skybox



To render skybox, first need to create VAO and VBO for it and load textures (using GL_CLAMP_TO EDGE to not draw the texture border). Skybox vertices are defined in the array for cube vertices with texture coordinates. Then the five sides of the cube are drawn.

```
glBindVertexArray(skyboxVAO);
glBindTexture(GL_TEXTURE_2D, backTexture);
glDrawArrays(GL_TRIANGLES, 0, 6);
glBindVertexArray(skyboxVAO);
glBindTexture(GL_TEXTURE_2D, leftTexture);
glDrawArrays(GL_TRIANGLES, 6, 6);
glBindVertexArray(skyboxVAO);
glBindTexture(GL_TEXTURE_2D, rightTexture);
glDrawArrays(GL_TRIANGLES, 12, 6);
glBindVertexArray(skyboxVAO);
glBindTexture(GL_TEXTURE_2D, frontTexture);
glDrawArrays(GL_TRIANGLES, 18, 6);
glBindVertexArray(skyboxVAO);
glBindTexture(GL_TEXTURE_2D, topTexture);
glDrawArrays(GL_TRIANGLES, 24, 6);
```

The vertex and fragment shader are pretty standard.

terrain.frag.glsl | terrain.vert.glsl | skybox.vert.glsl | water.vert.glsl | water.frag.glsl

```
#version 330 core

layout (location = 0) in vec3 position;
layout (location = 1) in vec2 texCoord;

out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform float scale;

void main()
{
    gl_Position = projection * view * model * vec4(position * scale, 1.0f);
    TexCoords = vec2(texCoord.x, 1.0f - texCoord.y);
}
```

```
#version 330 core

in vec2 TexCoords;

out vec4 color;

uniform sampler2D texture1;

void main()
{
    color = texture(texture1, TexCoords);
}
```

Also, reverse skybox is also rendered (to get the effect of reflection of skybox on water). Each of the cube's vertices is reflected relative to (0,1,0,0) plane to get a cube for reflected skybox.

Water

Water texture is loaded with GL_REPEAT, and vertices are taken from the bottom side of the cube, so uses skybox VAO and VBO. But the shaders are different:

terrain.frag.glsl | terrain.vert.glsl | skybox.vert.glsl | water.vert.glsl | water.frag.glsl

```
#version 330 core

layout (location = 0) in vec3 position;
layout (location = 1) in vec2 texCoord;

out vec2 TexCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform float scale;
uniform float u_time;


void main()
{
    gl_Position = projection * view * model * vec4(position * scale, 1.0f);
    TexCoords = vec2(texCoord.x + sin(u_time * 0.05), 1.0f - texCoord.y);
}
```

In the vertex shader float u_time is responsible for changing the texture coordinates to make the effect of flowing water. Fragment shader same as in skybox.

Terrain

Heightmap is loaded to the program as an array.

```
float h = heightimage[x * depthimage_width + z];
```

Height in each point is retrieved from this array and then normalized.

```
for (int x = 0; x < depthimage_height; x++) {
    for (int z = 0; z < depthimage_width; z++) {
        float h = heightimage[x * depthimage_width + z];

        /* Normalize height to [-1, 1] */
        h = h / 127.5;
        h = (h < 0.41) ? 0.41 : h;
        heights_init.push_back(x * 1.0 / depthimage_height);
        heights_init.push_back(h / scale * 3.0);
        heightmap[x][z] = h / scale * 3.0;
        heights_init.push_back(z * 1.0 / depthimage_width);

        // Texture coords
        heights_init.push_back(x * 1.0 / depthimage_height);
        heights_init.push_back(z * 1.0 / depthimage_width);

    }
}
```

Each of the texture coordinates on u,v mapping in equal to x and z values. Although for the detail noise texture it is not, but either 0.0 or 1.0.

To handle both of the textures, terrain vertices array contains:

- X-value
- Y-value
- Z-value
- Terrain texture u-value
- Terrain texture v-value
- Detail texture u-value
- Detail texture v-value

The VAO and VBO:

```
// setup terrain VAO and VBO
GLuint terrainVAO, terrainVBO;
glGenVertexArrays(1, &terrainVAO);
glGenBuffers(1, &terrainVBO);
glBindVertexArray(terrainVAO);
glBindBuffer(GL_ARRAY_BUFFER, terrainVBO);
glBufferData(GL_ARRAY_BUFFER, heights.size() * sizeof(float), &heights[0], GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 7 * sizeof(GLfloat), (GLvoid*)0);
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 7 * sizeof(GLfloat), (GLvoid*)(3 * sizeof(GLfloat)));
glEnableVertexAttribArray(2);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 7 * sizeof(GLfloat), (GLvoid*)(5 * sizeof(GLfloat)));
glBindVertexArray(0);
```

To draw both textures, we their uniform location in fragment shader is first collected, and then activate and deactivate each of the texture units.

```
GLint terrainTexLocation = glGetUniformLocation(terrainShader.Program, "texture0");
GLint detailsTexLocation = glGetUniformLocation(terrainShader.Program, "texture1");
glUniform1i(terrainTexLocation, 0);
glUniform1i(detailsTexLocation, 1);

glActiveTexture(GL_TEXTURE0 + 0); // Texture unit 0
glBindTexture(GL_TEXTURE_2D, terrainTexture);

glActiveTexture(GL_TEXTURE0 + 1); // Texture unit 1
glBindTexture(GL_TEXTURE_2D, detailsTexture);

glDrawArrays(GL_TRIANGLES, 0, heights.size());
glDisable(GL_TEXTURE_2D);
glActiveTexture(GL_TEXTURE1);
glDisable(GL_TEXTURE_2D);
glActiveTexture(GL_TEXTURE0);
glDisable(GL_TEXTURE_2D);
```

The main problem is that terrain texture has water drawn on it. This water should be cut from the terrain but with the programmable shaders clipping plane becomes complicated.

I have come up with my own solution:

terrain.frag.glsl | terrain.vert.glsl* | skybox.vert.glsl | water.vert.glsl | water.frag.glsl

```
#version 330 core

layout (location = 0) in vec3 position;
layout (location = 1) in vec2 texCoord;
layout (location = 2) in vec2 texCoord2;

out vec2 TexCoords;
out vec2 TexCoords2;
out float cl;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform float scale;
uniform int rev;

void main()
{
    gl_Position = projection * view * model * vec4(position * scale, 1.0f);
    if ((rev == 1 && position.y > -0.05) || (rev == 0 && position.y < 0.043))
    {
        cl = 0.0;
    }
    else
    {
        cl = 1.0;
    }
    TexCoords = vec2(texCoord.x, 1.0f - texCoord.y);
    TexCoords2 = vec2(texCoord2.x, 1.0f - texCoord2.y);
}
```

If the position of vertex is on the height of the water, then it should not be drawn on discarded in the fragment shader. Same for the reflection of terrain: if the height of vertex is too big, then in the middle of the island opposite side of the reflection will be drawn forming “water bubble”.

terrain.frag.glsl* | terrain.vert.glsl* | skybox.vert.glsl | water.vert.glsl | water.frag.glsl | skybox.frag.glsl | main.c

```
#version 330 core

in vec2 TexCoords;
in vec2 TexCoords2;
in float cl;

out vec4 color;

uniform sampler2D texture0;
uniform sampler2D texture1;

void main()
{
    vec4 texcolor = texture(texture0, TexCoords);
    if (cl < 0.5)
        discard;
    else
        color = clamp(texture(texture1, TexCoords2) + texcolor - vec4(0.5, 0.5, 0.5, 1.0), 0.0, 1.0);
}
```

To count the color of terrain with detail texture applied, we sum the texture colors and subtract 0.5 (equal to GL_ADD_SIGNED).

Interaction

To move between the scene smoothly, the time of each last pressed button is stored. After releasing this button, the player continues moving in this direction for a certain amount of time (100 * deltatime), slowing up gradually.

```
    if (lastFrame - lastPressed[0] < 100 * deltaTime)
        camera.ProcessKeyboard(FORWARD, 0.1 * deltaTime / ((lastFrame - lastPressed[0] + 0.1)));

    if (lastFrame - lastPressed[1] < 100 * deltaTime)
        camera.ProcessKeyboard(BACKWARD, 0.1 * deltaTime / ((lastFrame - lastPressed[1] + 0.1)));

    if (lastFrame - lastPressed[2] < 100 * deltaTime)
        camera.ProcessKeyboard(LEFT, 0.1 * deltaTime / ((lastFrame - lastPressed[2] + 0.1)));

    if (lastFrame - lastPressed[3] < 100 * deltaTime)
        camera.ProcessKeyboard(RIGHT, 0.1 * deltaTime / ((lastFrame - lastPressed[3] + 0.1)));

    if (lastFrame - lastPressed[4] < 100 * deltaTime)
        camera.ProcessKeyboard(UP, 0.1 * deltaTime / ((lastFrame - lastPressed[4] + 0.1)));

    if (lastFrame - lastPressed[5] < 100 * deltaTime)
        camera.ProcessKeyboard(DOWN, 0.1 * deltaTime / ((lastFrame - lastPressed[5] + 0.1)));
```

I have also decided to add collision, so the player could not get inside the terrain or under the water. Since for each coordinate x and z we know the value of height h, we can add a small value h_offset to it and define player coordinate be always bigger than value h + h_offset:

```
void is_collide(int heightmap_width)
{
    GLfloat cur_y = camera.Position.y;
    if (cur_y < 1.0) {
        //camera.ProcessKeyboard(UP, 1.0);
        camera.Position.y = 1.0;
    }
    else {
        float h_offset = -0.053f * scale;
        if (camera.Position.x > 0.0 && camera.Position.z > 0.0 &&
            (camera.Position.x - scale  * 1.0 < 0.0) &&
            (camera.Position.z - scale * 1.0 < 0.0)) {
            int x_ind = static_cast<int>(std::round(camera.Position.x / scale * heightmap_width));
            int z_ind = static_cast<int>(std::round(camera.Position.z / scale * heightmap_width));
            GLfloat ter_height = heightmap[x_ind][z_ind] * scale;
           if (cur_y + h_offset - ter_height - 0.2 < 0.0) {

                camera.Position.y = ter_height - h_offset + 0.2;
            }
        }
    }
}
```

The video demonstration of the program can be found in the same folder. Source code can also be downloaded from https://github.com/FedorIvachev/GraphicsCourse