Project #2 – Terrain Engine

Yoke Kai Wen, 2020280598

June 5, 2021

Contents

1 Introduction

In this project, I implemented a terrain engine with the following basic features:

1. Skybox and sea rendered
2. Waves on the sea simulated
3. Terrain model loaded and rendered with texture
4. Reflection of sky and terrain in water rendered
5. Can wander around the scene by keyboard

Additional bonus features include:

1. 3D water waves
2. Setting forbidden regions for camera position so that user cannot move below the sea or inside the terrain
3. Smoother movement with gradual slowing down

A sample screenshot of the rendered terrain can be seen in Figure 1.

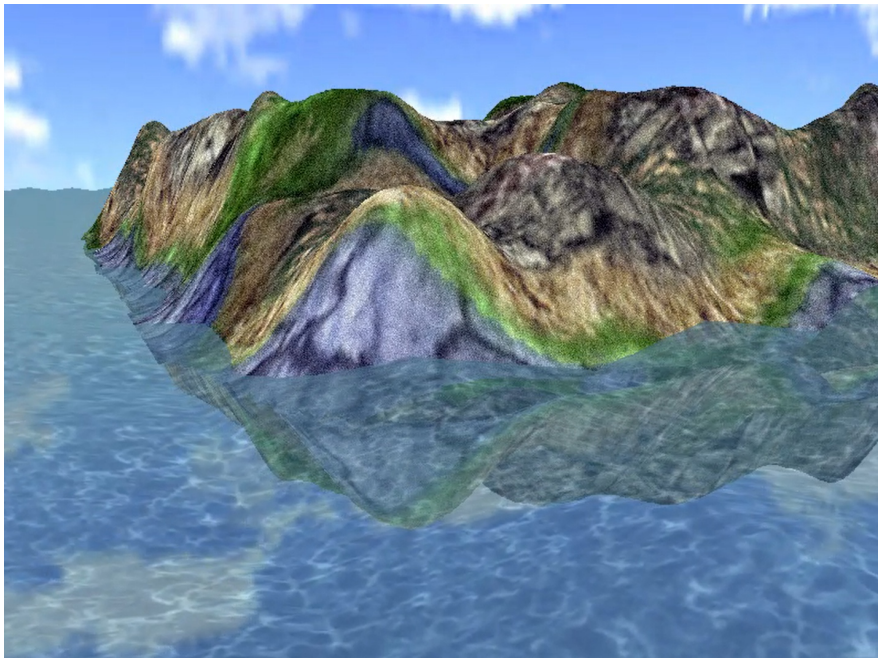

Figure 1: Screenshot from terrain island rendered

2 Files and directory structure

1. Main script: main.cpp
2. Shader files in shaders/ : sky.vert.glsl, sky.frag.glsl, wave.vert.glsl, wave.frag.glsl, terrain.vert.glsl, terrain.frag.glsl
3. Texture bitmaps in textures/ : detail.bmp, heightmap.bmp, terrain-texture3.bmp, SkyBox0.bmp, SkyBox1.bmp, SkyBox2.bmp, SkyBox3.bmp, SkyBox4.bmp, SkyBox5.bmp
4. Other header files: include/tut_headers/shader.h, include/tut_headers/camera.h
5. Set include/ and lib/ folders as include and library directories respectively in VS

3 Usage

1. Camera control: W, S, A, D, Z, X, or mouse movement
2. Toggle between 3D and 2D waves: C

4 Design

4.1 Rendering skybox

The skybox contains the blue sky with clouds and the sun. It is formed by mapping the five skybox texture images `SkyBox0.mp`, `SkyBox1.mp`, `SkyBox2.mp`, `SkyBox3.mp`, `SkyBox4.mp` onto the front, right, back, left and top sides of the cube respectively.

4.1.1 Skybox coordinates

The position and texture coordinates of each cube surface is determined as seen in the code fragment below, making sure that the position (0,0,0) is right in the centre of the skybox. The coordinates are basically those of a 8 units cube.

```
//skybox0
GLfloat skyFrontVertices[30] = {
    // Positions          // Texture Coords
    -1.0f, -1.0f,  1.0f,  0.0f,  0.0f, //front
    1.0f, -1.0f,  1.0f,  1.0f,  0.0f,
    1.0f,  1.0f,  1.0f,  1.0f,  1.0f,
    1.0f,  1.0f,  1.0f,  1.0f,  1.0f,
    -1.0f,  1.0f,  1.0f,  0.0f,  1.0f,
    -1.0f, -1.0f,  1.0f,  0.0f,  0.0f,
};
```

4.1.2 Binding vertex buffers and loading textures

The vertex buffers of each skybox surface are then bound to the vertex buffer objects with the helper function `setupSkyboxBuffers(skyFrontVAO, skyFrontVBO, skyFrontVertices, 30)`, and the skybox textures are loaded with the `loadTexture()` function. Note that to make the edge lines of the skybox invisible, we set the wrapping mode of the texture coordinate to `GL_CLAMP_TO_EDGE` in the `loadTexture` function.

4.1.3 Sky rendering shader

Before rendering the skybox, the buffer arrays are passed into the `shaders/sky.vert.glsl` shader, as shown in the code snippet below. It is the standard shader used in our tutorials for model-view transformation, but taking extra parameters such as `scale` and `rev` into account. `scale` is set to 30.0 in my project to enlarge the skybox, so that immediately the user view is completely confined inside the skybox and will see a paranomic view, instead of seeing a cube. `rev` is used for implementing reflection, which I will discuss later. Note that the camera `view` matrix for the skybox also has the translation component removed so that the skies remain static despite camera movement.

```
//sky.vert.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 int rev;

void main()
{
    vec3 pos = position;
    if (rev == 1) {
        pos.y *= -1;
    }
    gl_Position = projection * view * model * vec4(pos * scale, 1.0f);

    TexCoords = vec2(texCoord.x, 1.0f - texCoord.y);
}
```

4.2 Simulating waves on the sea

3D waves are simulated by producing a dynamic 3D water mesh, that is mapped to the sea texture `SKyBox5.bmp`. To make the waves look like they are moving, there are two key points: (1) varying the x texture coordinate to move wave in x-direction; (2) varying the y-coordinates of the wave to make the waves move up and down.

4.2.1 3D water mesh

To initialise the 3D water mesh, I first generate a 2D square grid (x,z coordinates) of dimensions 256 x 256 (waterPix=256), setting the x,z coordinates to a range of -10.0 to 10.0 (bigger than the dimensions of the 8 units cube skybox). I initialise the y-coordinate to zero, which would give a flat sea surface with no movement in the vertical direction. See code segment below.

```
// ------------------- water 3D mesh ---------------------
i = 0;
for (int r = 0; r < waterPix; r++) {
    for (int c = 0; c < waterPix; c++) {

```

```
        x = r * 1.0 / waterPix * 20.0 - 10.0;
        z = c * 1.0 / waterPix * 20.0 - 10.0;
        y = 0;

        waterMeshPos[i++] = x;
        waterMeshPos[i++] = y;
        waterMeshPos[i++] = z;

        waterHeightmap[r][c] = y;
    }
}
```

After obtaining the x, y, z coordinates of the mesh, I then connect adjacent points of the grid mesh to form patches, each composed of two triangles, so that I can render them in triangles later on. Each triangle is associated with texture coordinates such that each patch will be rendered with the entire `SkyBox5.bmp` sea texture. See code fragment below.

```
j = 0;
for (int r = 0; r < waterPix - 1; r++) {
    for (int c = 0; c < waterPix - 1; c++) {

        int start = r * waterPix + c;

        // triangle 1, vertex 1
        waterMesh[j++] = waterMeshPos[start * 3];
        waterMesh[j++] = waterMeshPos[start * 3 + 1];
        waterMesh[j++] = waterMeshPos[start * 3 + 2];
        waterMesh[j++] = 0.0f;
        waterMesh[j++] = 0.0f;

        // triangle 1, vertex 2
        waterMesh[j++] = waterMeshPos[(start + waterPix) * 3];
        waterMesh[j++] = waterMeshPos[(start + waterPix) * 3 + 1];
        waterMesh[j++] = waterMeshPos[(start + waterPix) * 3 + 2];
        waterMesh[j++] = 0.0f;
        waterMesh[j++] = 1.0f;

        // triangle 1, vertex 3
        waterMesh[j++] = waterMeshPos[(start + waterPix + 1) * 3];
        waterMesh[j++] = waterMeshPos[(start + waterPix + 1) * 3 + 1];
        waterMesh[j++] = waterMeshPos[(start + waterPix + 1) * 3 + 2];
        waterMesh[j++] = 1.0f;
        waterMesh[j++] = 1.0f;


        // triangle 2, vertex 1
        waterMesh[j++] = waterMeshPos[(start + waterPix + 1) * 3];
        waterMesh[j++] = waterMeshPos[(start + waterPix + 1) * 3 + 1];
        waterMesh[j++] = waterMeshPos[(start + waterPix + 1) * 3 + 2];
        waterMesh[j++] = 1.0f;
        waterMesh[j++] = 1.0f;

        // triangle 2, vertex 2
        waterMesh[j++] = waterMeshPos[(start + 1) * 3];
        waterMesh[j++] = waterMeshPos[(start + 1) * 3 + 1];
        waterMesh[j++] = waterMeshPos[(start + 1) * 3 + 2];
        waterMesh[j++] = 1.0f;
        waterMesh[j++] = 0.0f;

```

```
        //triangle 2, vertex 3
        waterMesh[j++] = waterMeshPos[start * 3];
        waterMesh[j++] = waterMeshPos[start * 3 + 1];
        waterMesh[j++] = waterMeshPos[start * 3 + 2];
        waterMesh[j++] = 0.0f;
        waterMesh[j++] = 0.0f;
    }
}
```

4.2.2 Updating vertical motion for 3D waves*

For every frame, I update the heights of the water waves in the function `updateWaterWaves()` as seen below. I first compute the y-coordinates of the water waves according to its x,z coordinates and the current time: $y = \frac{A}{2}(sin(\omega x + \phi t) + cos(\omega z + \phi t))$. The amplitude A, frequency ω, and phase ϕ of the water waves are also controlled by constants and should be carefully set to produce a nice wave effect. I then update the water mesh vertex buffers with the newly computed y-coordinates, and bind the vertex buffer object again using `glBufferSubData` which is useful for updating data buffer values. Also note that when binding the water vertex buffer initially with `glBindBuffer`, we use the `GL_DYNAMIC_DRAW` flag as the water coordinates are rendered differently every frame.

```
void updateWaterWaves()
{
    float x, z;
    for (int r = 0; r < waterPix; r++) {
        for (int c = 0; c < waterPix; c++) {
            if (wave3d == 1) {
                x = 4 * r;
                z = 4 * c;
                waterHeightmap[r][c] = amp * 0.5 * (sin(x * freq + lastFrame *
    phase) + cos(z * freq + lastFrame * phase));
            }
            else {
                waterHeightmap[r][c] = 0;
            }
        }
    }

    // update the y coords of water vertex buffer
    int r, c;
    for (int i = 1; i < numWaterVertices; i += 5) {
        r = std::round((waterMesh[i - 1] + 10.0) / 20.0 * waterPix);
        c = std::round((waterMesh[i + 1] + 10.0) / 20.0 * waterPix);
        waterMesh[i] = waterHeightmap[r][c];
    }

    glBindVertexArray(water3dVAO);
    glBindBuffer(GL_ARRAY_BUFFER, water3dVBO);
    glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(GLfloat)*numWaterVertices, &
    waterMesh[0]);
    glBindVertexArray(0);
}
```

4.2.3 Updating horizontal wave movement

To move the wave in the x-direction, I simply adjust the x-texture coordinate according to a given wave. I first compute the amount of shift in the x-texture coordinate `xshift += waterSpeed*deltaTime` based on the water speed and the time elapsed. Then I pass `xshift` into the wave shader `wave.vert.glsl`, and add it to the x-texture coordinate. This causes the wave to move in the x-direction at a uniform speed.

```
//wave.vert.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 xshift;
uniform float scale;

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

4.2.4 Setting colour and transparency of sea

The sea needs to be transparent so that the reversed skybox and the reversed terrain can be seen and perceived as reflections. This is done by blending the sea texture with a white colour vector of 0.7 transparency in the `wave.frag.glsl` shader as seen below.

```
//wave.frag.glsl
#version 330 core

in vec2 TexCoords;

out vec4 color;

uniform sampler2D textureSea;

void main()
{
    color = texture(textureSea, TexCoords) * vec4(1.0f,1.0f, 1.0f, 0.7f);
}
```

4.3 Loading the terrain model and rendering it with texture

Three main resources are required for loading and rendering the terrain: `heightmap.bmp, terrain-texture3.bmp, detail.bmp`. First, the height information is taken from `heightmap.bmp` to produce the 3D terrain mesh. The terrain texture and the detail texture are then mapped onto the 3D mesh with multi-texturing.

4.3.1 Loading height information

Using the `SOIL` library, `heightmap.bmp` is loaded as an image, where the row and column pixels of the image represent the x and z coordinates, while the grayscale pixel value indicates the terrain height (0 to 255) and should be mapped to the y-coordinate. Then, I scale the height value and floor the height value to achieve a more pleasant rendering effect. I also scale the x and z coordinate values to a range of 0 to 1, so that the island terrain takes up a space of a unit square relative to the (8units cube) skybox. Then, I add the terrain texture coordinates for each vertex, which is trivial since the texture coordinates correspond to the x,z positional coordinates. See code fragment below.

```
// ----------------------- terrain -----------------------------
int width, height, channels;
unsigned char* heightImage = SOIL_load_image("textures/heightmap.bmp", &width, &
    height, &channels, SOIL_LOAD_L);
GLfloat* heightInfo = new GLfloat[width * height * 5]; // (x, y, z, terrain1,
    terrain2), y is height
GLfloat x, y, z, t1, t2;
int i = 0;
for (int r = 0; r < height; r++) {
    for (int c = 0; c < width; c++) {
        y = heightImage[r * width + c]; //raw height value

        //scale height
        y = y / 127.5;
        y = (y < 0.41) ? 0.41 : y; // not sure how value was chosen

        //pos and texture coords
        x = r * 1.0 / height;
        y = y / scale * 3.0;
        z = c * 1.0 / width;
        t1 = r * 1.0 / height;
        t2 = c * 1.0 / width;

        heightmap[r][c] = y;

        heightInfo[i++] = x;
        heightInfo[i++] = y;
        heightInfo[i++] = z;
        heightInfo[i++] = t1;
        heightInfo[i++] = t2;
    }
}
```

The detail texture cannot be added first, because each vertex can be associated with multiple detail texture coordinates, depending on which triangle it belongs to. Similar to the construction of the 3D water mesh, for the terrain, we also have to connect adjacent points of the terrain heightmap grid into patches of triangles, and add detail texture coordinates. See code fragment below.

```
//link adjacent vertices together into triangles
// also add detail texcoords
int numIslandVertices = (width - 1) * (height - 1) * 6 * 7; //patch count * 6 (2
    triangles per terrain patch) * 7 (x,y,z,t1,t2,d1,d2)
GLfloat* islandVertices = new GLfloat[numIslandVertices];
int j = 0;
for (int r = 0; r < height - 1; r++) {
    for (int c = 0; c < width - 1; c++) {
```

```
8
9           int start = r * width + c;
10
11          // triangle 1, vertex 1
12          islandVertices[j++] = heightInfo[start * 5];
13          islandVertices[j++] = heightInfo[start * 5 + 1];
14          islandVertices[j++] = heightInfo[start * 5 + 2];
15          islandVertices[j++] = heightInfo[start * 5 + 3];
16          islandVertices[j++] = heightInfo[start * 5 + 4];
17          islandVertices[j++] = 0.0f;
18          islandVertices[j++] = 0.0f;
19
20          // triangle 1, vertex 2
21          islandVertices[j++] = heightInfo[(start + width) * 5];
22          islandVertices[j++] = heightInfo[(start + width) * 5 + 1];
23          islandVertices[j++] = heightInfo[(start + width) * 5 + 2];
24          islandVertices[j++] = heightInfo[(start + width) * 5 + 3];
25          islandVertices[j++] = heightInfo[(start + width) * 5 + 4];
26          islandVertices[j++] = 0.0f;
27          islandVertices[j++] = 1.0f;
28
29          // triangle 1, vertex 3
30          islandVertices[j++] = heightInfo[(start + width + 1) * 5];
31          islandVertices[j++] = heightInfo[(start + width + 1) * 5 + 1];
32          islandVertices[j++] = heightInfo[(start + width + 1) * 5 + 2];
33          islandVertices[j++] = heightInfo[(start + width + 1) * 5 + 3];
34          islandVertices[j++] = heightInfo[(start + width + 1) * 5 + 4];
35          islandVertices[j++] = 1.0f;
36          islandVertices[j++] = 1.0f;
37
38
39          // triangle 2, vertex 1
40          islandVertices[j++] = heightInfo[(start + width + 1) * 5];
41          islandVertices[j++] = heightInfo[(start + width + 1) * 5 + 1];
42          islandVertices[j++] = heightInfo[(start + width + 1) * 5 + 2];
43          islandVertices[j++] = heightInfo[(start + width + 1) * 5 + 3];
44          islandVertices[j++] = heightInfo[(start + width + 1) * 5 + 4];
45          islandVertices[j++] = 1.0f;
46          islandVertices[j++] = 1.0f;
47
48          // triangle 2, vertex 2
49          islandVertices[j++] = heightInfo[(start + 1) * 5];
50          islandVertices[j++] = heightInfo[(start + 1) * 5 + 1];
51          islandVertices[j++] = heightInfo[(start + 1) * 5 + 2];
52          islandVertices[j++] = heightInfo[(start + 1) * 5 + 3];
53          islandVertices[j++] = heightInfo[(start + 1) * 5 + 4];
54          islandVertices[j++] = 1.0f;
55          islandVertices[j++] = 0.0f;
56
57          //triangle 2, vertex 3
58          islandVertices[j++] = heightInfo[start * 5];
59          islandVertices[j++] = heightInfo[start * 5 + 1];
60          islandVertices[j++] = heightInfo[start * 5 + 2];
61          islandVertices[j++] = heightInfo[start * 5 + 3];
62          islandVertices[j++] = heightInfo[start * 5 + 4];
63          islandVertices[j++] = 0.0f;
64          islandVertices[j++] = 0.0f;
65      }
66  }
```

4.3.2 Multi-texturing

The terrain and detail textures are loaded, and then activated and bound in turn, and passed into the `terrain.frag.glsl`,and then summed together to produce the final colour as seen in code fragements below.

```
glBindVertexArray(islandVAO);

glUniform1i(glGetUniformLocation(terrainShader.Program, "texture0"), 0);
glUniform1i(glGetUniformLocation(terrainShader.Program, "texture1"), 1);

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

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

glDrawArrays(GL_TRIANGLES, 0, numIslandVertices);

glDisable(GL_TEXTURE_2D);
glActiveTexture(GL_TEXTURE1);
glDisable(GL_TEXTURE_2D);
glActiveTexture(GL_TEXTURE0);
glDisable(GL_TEXTURE_2D);
```

```
//terrain.frag.glsl
#version 330 core

in vec2 TexCoords;
in vec2 TexCoords2;
in float clip;

out vec4 color;

uniform sampler2D texture0;
uniform sampler2D texture1;

void main()
{
    vec4 texcolor = texture(texture0, TexCoords);

    if (clip < 0.5)
        discard;
    else
        color = clamp(texture(texture1, TexCoords2) + texcolor - vec4(0.5, 0.5,
    0.5, 1.0), 0.0, 1.0);
}
```

4.3.3 Putting terrain into the sea

The terrain is submerged slightly under the water, with the uneven submerged part clipped for better visual effect. Clipping is implemented in the terrain vertex shader `terrain.vert.glsl`, such that the parts of the terrain that are too far below the water surface are discarded in the fragment shader `terrain.frag.glsl`.

4.4 Implementing reflection of sky and terrain in water

To implement reflection, I set the water to be semi-transparent, and render the inverted skybox and terrain under water. This is done by multiplying -1 to the y-coordinate of the skybox and terrain vertices in the skybox vertex shader `sky.vert.glsl` and the terrain vertex shader `terrain.vert.glsl`, if the `rev` uniform is set to 1 before being passed into the vertex shaders.

4.5 Wandering around the scene

The user can wander around the scene with keyboard and mouse movements. I also add a function to prevent users from going underwater or into the terrain for better user experience. Also, I added a function for users to gradually slow down instead of stopping abruptly when releasing a keyboard key.

4.5.1 Forbidden regions*

The `setForbiddenRegion()` function bans the user form entering forbidden regions as seen in code fragment below. The idea is to keep the camera above sea level by setting the camera's y-position to above 1.0 at all times, so that the user cannot go under water. When the x and z coordinates of the camera overlap with the island terrain's x and z coordinates, I also ensure that the camera's y-coordinate will be set above the terrain height, to prevent the user from seeing the interior of the terrain.

```
void setForbiddenRegion()
{
    GLfloat cam_y = camera.Position.y;
    if (cam_y < 1.0) {
        //keeps camera above sea level
        camera.Position.y = 1.0;
    }
    else {
        //prevents user from getting inside the terrain
        if (camera.Position.x > 0.0 - 0.1 && camera.Position.z > 0.0 - 0.1 && (
    camera.Position.x < scale + 3) && (camera.Position.z < scale + 3))
        {
            int x_ind = std::round(camera.Position.x / scale * 256);
            int z_ind = std::round(camera.Position.z / scale * 256);
            GLfloat ter_height = heightmap[x_ind][z_ind] * scale;
            if (cam_y - ter_height - 0.2 < 0.0) {
                camera.Position.y = ter_height + 0.2;
            }
        }
    }
}
```

4.5.2 Gradual slowing down*

In the function `Do_Movement()`, I implemented gradual slowing down when a key is released, such that the camera still moves in the same direction with decreasing speed when the key is released.

```
void Do_Movement()
{
    // water 3d waves
    if (keys[GLFW_KEY_C]) {
        wave3d = 1 - wave3d;
```

```
    }

    // Camera controls
    if (keys[GLFW_KEY_W]) {
        camera.ProcessKeyboard(FORWARD, deltaTime);
        lastPressed[0] = lastFrame;
    }
    if (keys[GLFW_KEY_S]) {
        camera.ProcessKeyboard(BACKWARD, deltaTime);
        lastPressed[1] = lastFrame;
    }
    if (keys[GLFW_KEY_A]) {
        camera.ProcessKeyboard(LEFT, deltaTime);
        lastPressed[2] = lastFrame;
    }
    if (keys[GLFW_KEY_D]) {
        camera.ProcessKeyboard(RIGHT, deltaTime);
        lastPressed[3] = lastFrame;
    }
    if (keys[GLFW_KEY_Z]) {
        camera.ProcessKeyboard(UP, deltaTime);
        lastPressed[4] = lastFrame;
    }
    if (keys[GLFW_KEY_X]) {
        camera.ProcessKeyboard(DOWN, deltaTime);
        lastPressed[5] = lastFrame;
    }

    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)));
}
```

5 References

1. Past year reference CG project on terrain engine: https://github.com/FedorIvachev/GraphicsCourse/tree/master/Terrain%20engine

2. 3D water waves rendering: https://www.cc.gatech.edu/~ybai30/cs_7490_final_website/cg_water.html