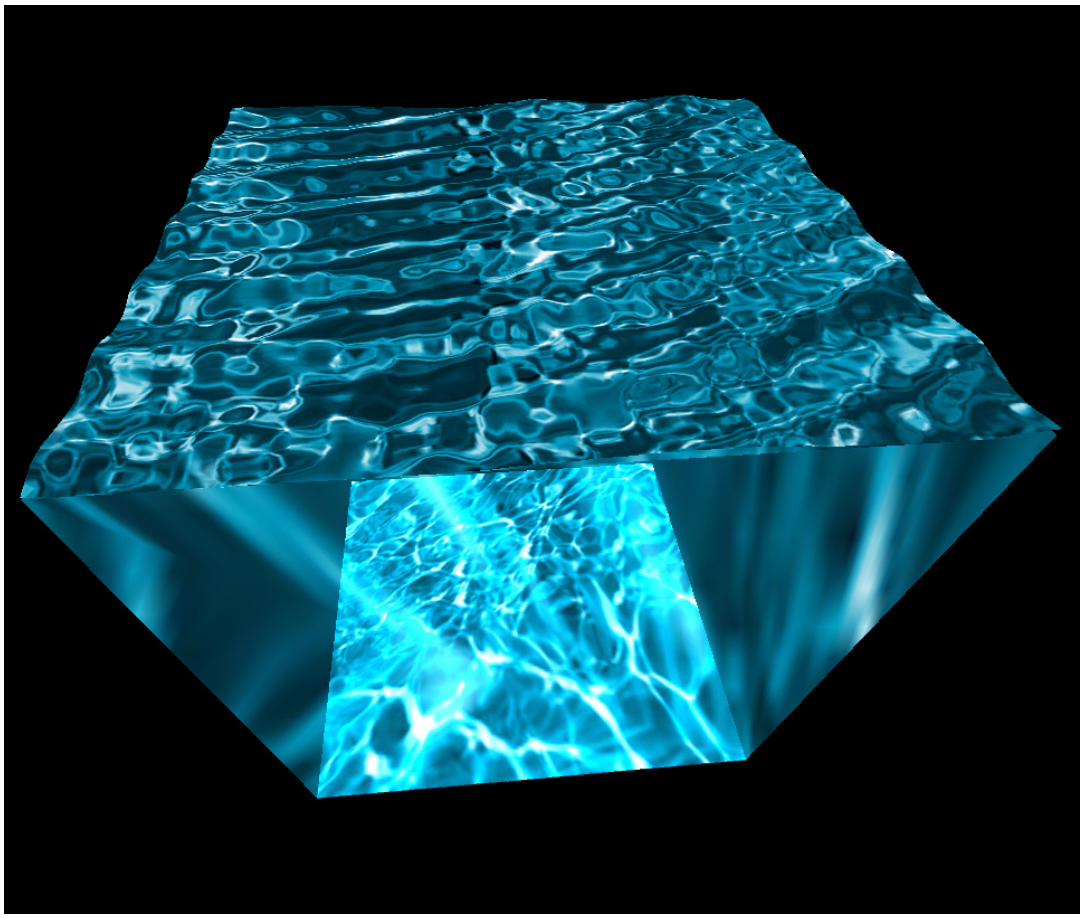
CS 550 Final Project

Water Simulation OpenGL

Margaux Masson - massonm@oregonstate.edu



Oregon State University

Computer Science

Fall 2018

Table of Contents

Chapter 1: **Introduction**

Introduction to Computer Graphics Final Project

This final project has for main purpose to apply all the techniques learned during this term. Therefore, I chose to implement some water/waves simulator using OpenGL. This project involves implementing and manipulating 3D surfaces and normals, openGL functions, mathematical equations, texturing, lighting, and more.

In this report, I will expose the different steps which led me to implement this simulation, the issues I encountered and the solutions that I chose. Plus, some code explanation and some pictures to illustrate this report.

Chapter 2: **Project**

2.1 Reminder of the initial proposal

Water – Waves simulation using OpenGL

For the final project I would like to implement a simulation of water with waves using the vertex and fragment shaders. And, if possible add some reflection as well. And if I can get further enough, I would like to add some landscape using the water implement, as a lake or the sea. So in this project I will use:

1. *vertex fragment shaders*
2. *equation to create the waves*
3. *texturing*

2.2 Project Timeline

2.2.1 Looking for a tutorial

The first step of this project was to find a tutorial which could help me designing water. I wanted to find one in which I knew I could implement both water appearance and waves simulation. I found one using surfaces and normals' implementation that seemed really handily and interesting. So, after making it work on my computer, I started integrating a part of this code into the sample.cpp file that I use all around the term as a base for my CS550 projects.

2.2.2 Implementing Surfaces

So, first I had to implement the surface that will be the water. This surface is a table defined by:

```
static float surface[6 * resolution * (resolution + 1)]
```

and implemented as following:

```
/* Vertices */
for (j = 0; j < resolution; j++)
{
    y = (j + 1) * delta - 1;
    for (i = 0; i <= resolution; i++)
    {
        indice = 6 * (i + j * (resolution + 1));
        x = i * delta - 1;
        surface[indice + 3] = x;
        surface[indice + 4] = 0;
        surface[indice + 5] = y;
        if (j != 0)
        {
            /* Values were computed during the previous loop */
            preindice = 6 * (i + (j - 1) * (resolution + 1));
            surface[indice] = surface[preindice + 3];
            surface[indice + 1] = surface[preindice + 4];
            surface[indice + 2] = surface[preindice + 5];
        }
        else
        {
            surface[indice] = x;
            surface[indice + 1] = 0;
            surface[indice + 2] = -1;
        }
    }
}
```

Listing 2.1: creation of the flat surface

After this, the surface is drawn using this simple code:

```
    /// Water ///

    glEnableClientState(GL_VERTEX_ARRAY); // allow vertices
    glVertexPointer(3, GL_FLOAT, 0, surface); // use the table surface
                                              // to draw the arrays

    for (int i = 0; i < resolution; i++)
        glDrawArrays(GL_TRIANGLE_STRIP, i * length, length);
        // length = 2 * (resolution + 1)

    glEnd();
```

Listing 2.2: Drawing water surface

Here is the result:

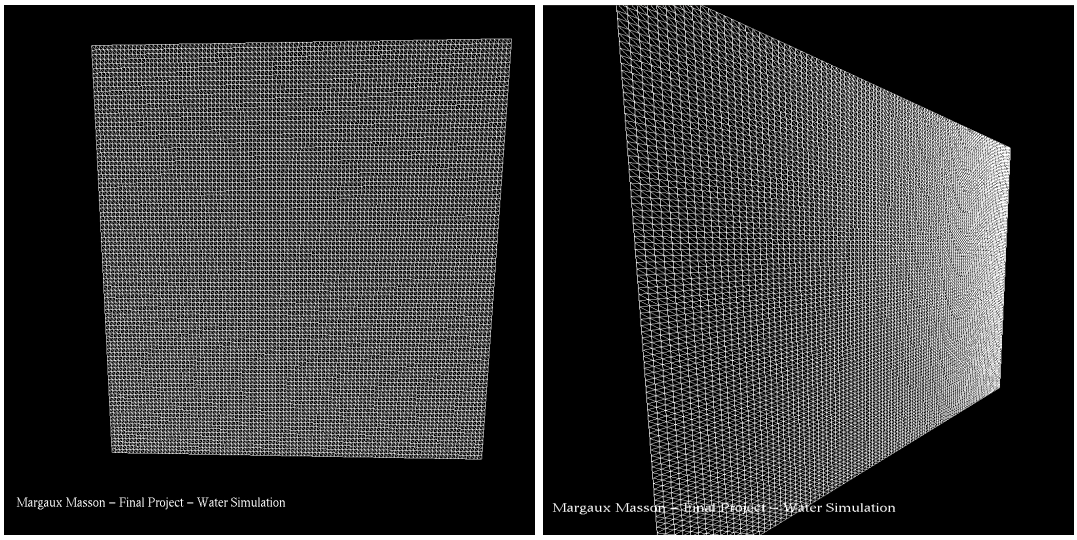


Figure 2.1: Basic surface - without waves' implement yet

2.2.3 Waves - Noise

Now in order to simulate the waves, it only requires to change these lines in the previous code:

```
/* Vertices */
for (j = 0; j < resolution; j++)
{
    ...
        surface[indice + 4] = waves_creation(x, y, t);

        ...
            surface[indice + 1] = waves_creation(x, -1, t);
        ...
}
```

Listing 2.3: creation of the waves on the surface - modifying the Z coordinates

with:

```
static float waves_creation(const float x, const float y, const float t)
{
    const float x2 = x - 3;
    const float y2 = y + 1;
    const float xx = x2 * x2;
    const float yy = y2 * y2;

    return ((2 * cosf(20 * sqrtf(xx + yy) - 4 * t) + Noise(waves_intensity * x,
     waves_intensity * y, t, 0)) / 200);
    }
}
```

Listing 2.4: Function which creates the waves

and the function Noise returns the Perlin Noise which can be found in the file noise.cpp. Moreover, the parameter `waves_intensity` is used to increase or decrease the intensity of the waves.

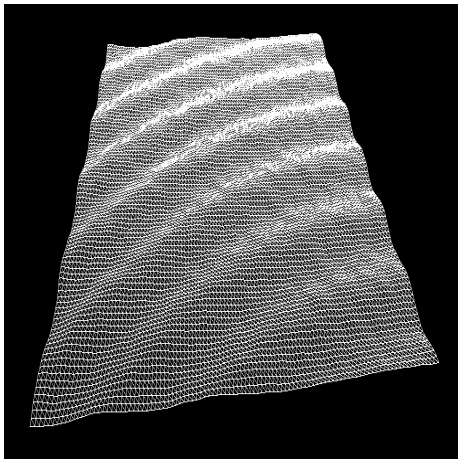


Figure 2.2: Surface with waves

Therefore, as we can observe here, the function waves creation creates waves using the sinf function and a Noise function which take the time as parameter, so the waves are animated according to the time. So, by adding a condition:

```
if (!Frozen) {
  ...
}
```

We can use a freeze/un-freeze option which is really useful for debugging.

2.2.4 Normals

Once the surface is implemented, the next step is to implement normals corresponding to the surface in order to be able to assign a texture to the later.

The code used to implement the normals is long, and I did not modify the one from the tutorial because it was good enough for the use I am doing in this project. I chose to take more time for the texturing part that comes after this step.

However, it is essential to add these lines to the code that draws the surface (cf Listing 2.2) so the normals can be enabled:

```
    /// Water ///
    ...
    glEnableClientState(GL_NORMAL_ARRAY);
    glNormalPointer(GL_FLOAT, 0, normal);
    ...
```

Listing 2.5: Enabling normals when drawing water surface

We can now observe the normals' behavior with the waves on the surface:

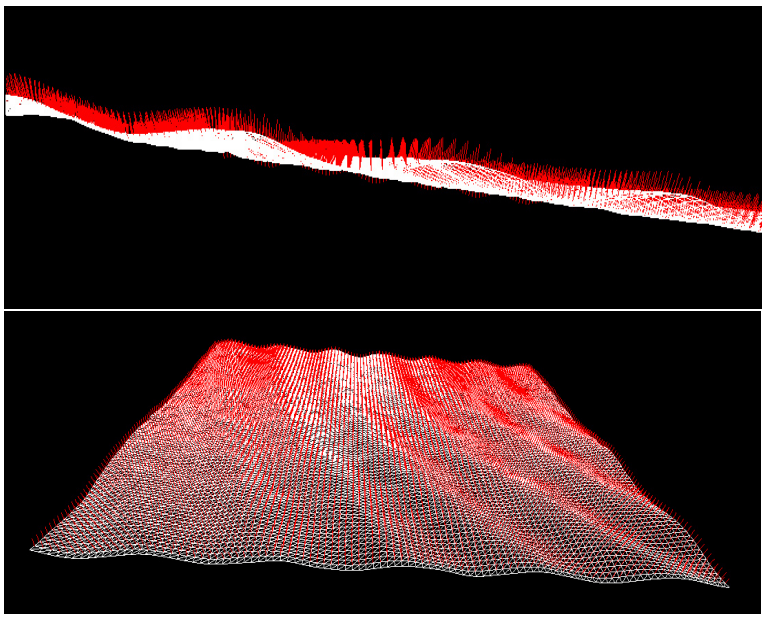


Figure 2.3: Surface with waves and normals

2.2.5 Texturing

I wanted to use different textures, so the user can switch between them. I use these three pictures as textures for the water:

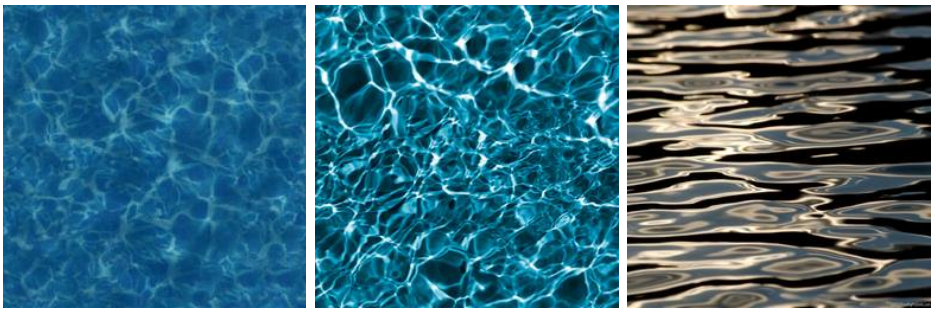


Figure 2.4: jpeg files used as textures for the water

which are converted into textures thanks to the function load texture which takes jpeg files as input.

The textures are using the normals In order to obtain these moving textures, I used the function gluBuild2DMipmaps as following:

```
gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGBA, 256, 256, GL_RGBA,GL_UNSIGNED_BYTE,
    total_texture);
```

Results:

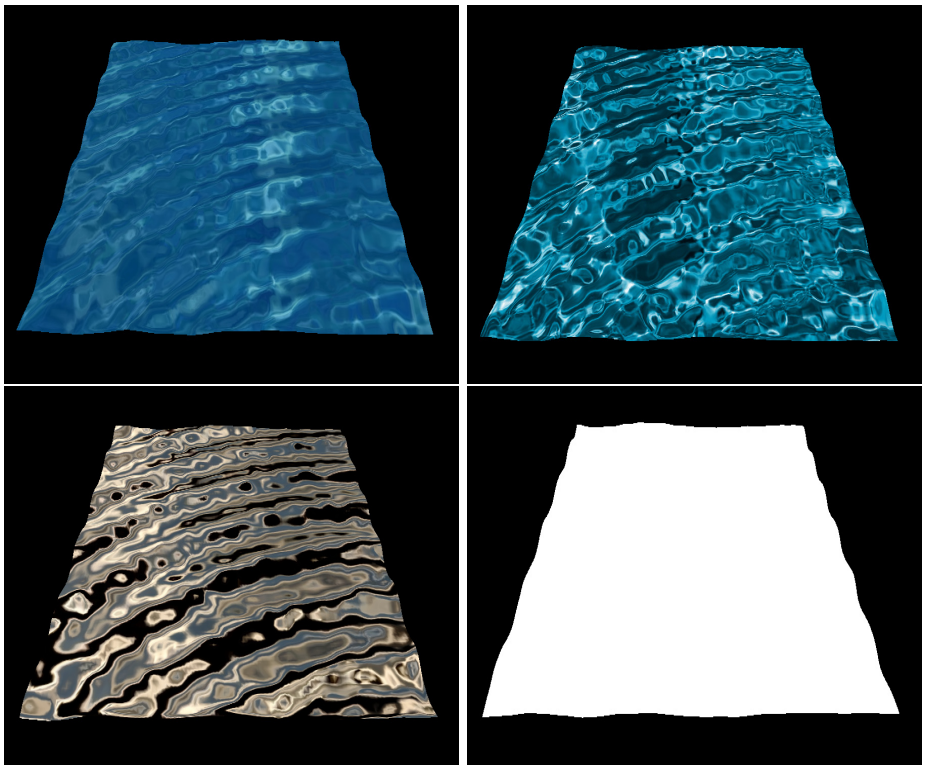


Figure 2.5: Texturing water: simple, deep blue and dark water and blobish-without texture surface

2.2.6 Cube

In order to create a more realistic scene, I wanted to implement the water surface into a "cube" which would appear as a swimming pool. I could not use the GLUT cube because the top face of the cube was interfering with the water surface. I decided then to create five "walls" which are simple translated and rotated `GL_TRIANGLE_FAN`.

Results:

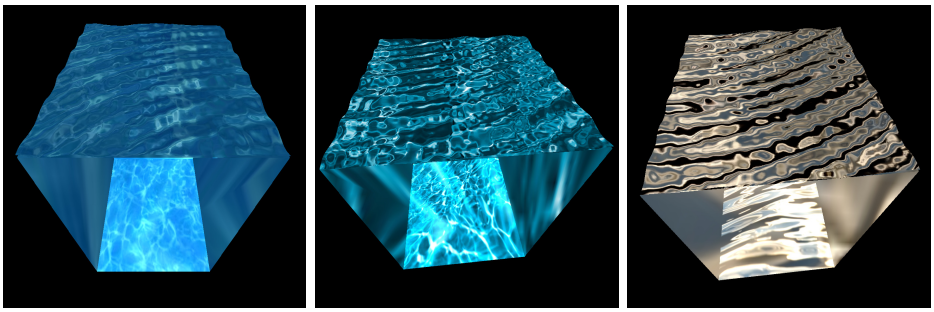


Figure 2.6: jpeg files used as textures for the water

2.2.7 Additional study: Grass field using surfaces

When implementing this surface for the water, I realized that it might be possible to re use this kind of surface to create landscape such as hills or even grass. So I tried to add some grass all around the water surface by adding 8 surfaces similar to the water one and modifying the `waves_creation` function as following:

```
static float field_creation(const float x, const float y)
{
    return (Noise(10 * x, 10 * y, 20, 0) / 8);
}
```

Listing 2.6: Function used to create the grass field

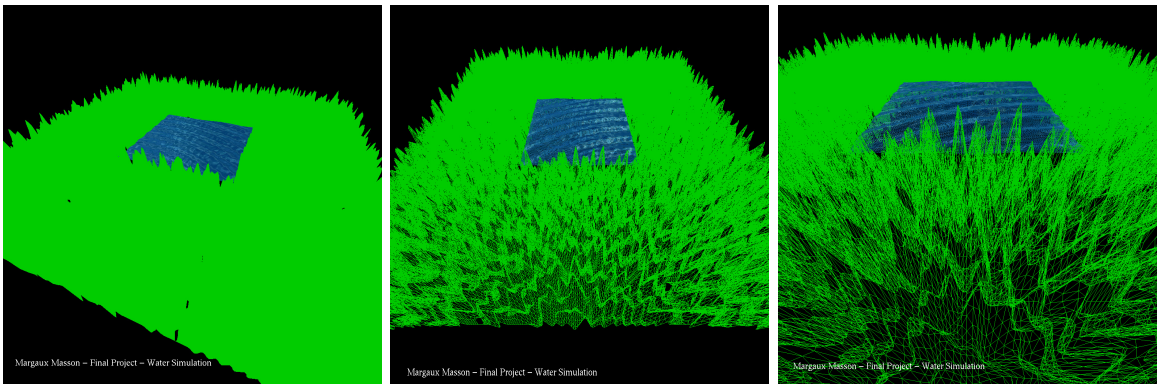


Figure 2.7: Creation of a grass field using the same surface as the water but using a slightly modified field creation functions

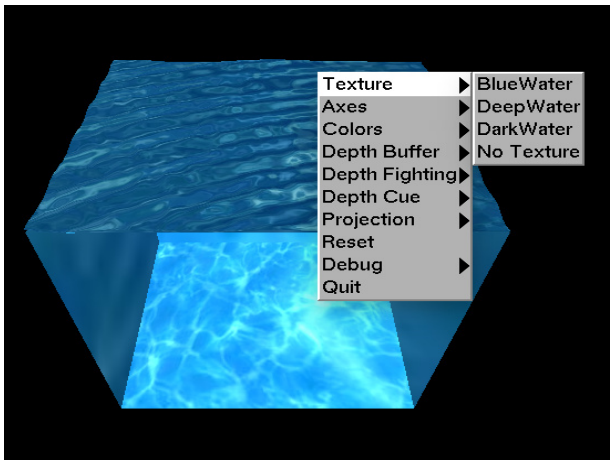
2.2.8 Options: Keyboard commands

Several keys can be use to control the scene:

- *p or P:* Orthographic projection
- *o or O:* Perspective projection
- *f or F:* Freeze or un-freeze the waves' animation
- *l or L:* Display the wires
- *n or N:* Display the normals
- *t or T:* Switch between the three textures (blue, deep blue, dark blue) and no texture
- *+ and -:* Increase/Decrease intensity of the waves
- *x or X:* Rotate the scene around the X-axis
- *y or Y:* Rotate the scene around the Y-axis
- *z or Z:* Rotate the scene around the Z-axis
- *a or A:* Switch between cube scene and field (grass) landscape
- *w or W:* Simulate the water without unidirectional waves

- *0:* Display only the water
- *q or Q:* Quit the program

Some of these actions can also be found in the right-click menu:



2.3 Differences between proposal and actual project

I wanted to use the vertex and fragment shaders to create the waves and simulate the water's texturing but I found the tutorial using this kind of surface and I wanted to know more about it since I also had some ideas about how to re use this implementation, such as for the grass field.

Chapter 3: **Conclusion**

3.1 What I learned

Through this project, I learned how to combined together some projects done during the term, for instance I re used the texture implementation that I have done for the project 3. Moreover, I learn how to draw and scuplte a surface in openGL in order to create a 3D shape such as water or grass. I also learn how to create some more complex texturing using the normals and even to add some transparency to some surfaces using `GL_BLEND`.

3.2 Useful Links

3.2.1 Github repository

You can find the code for this project here.

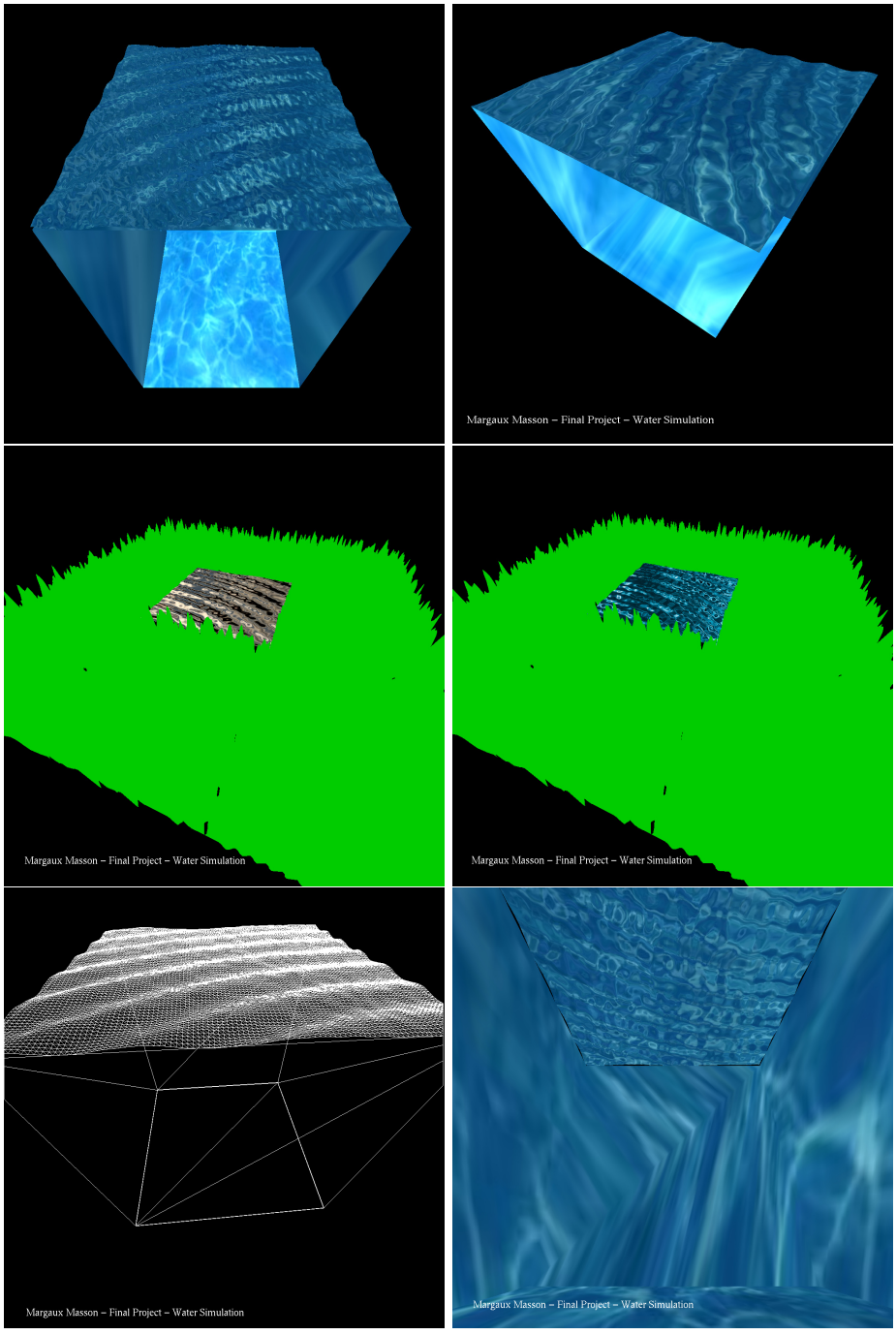
Clone the repository and run make to execute the code.

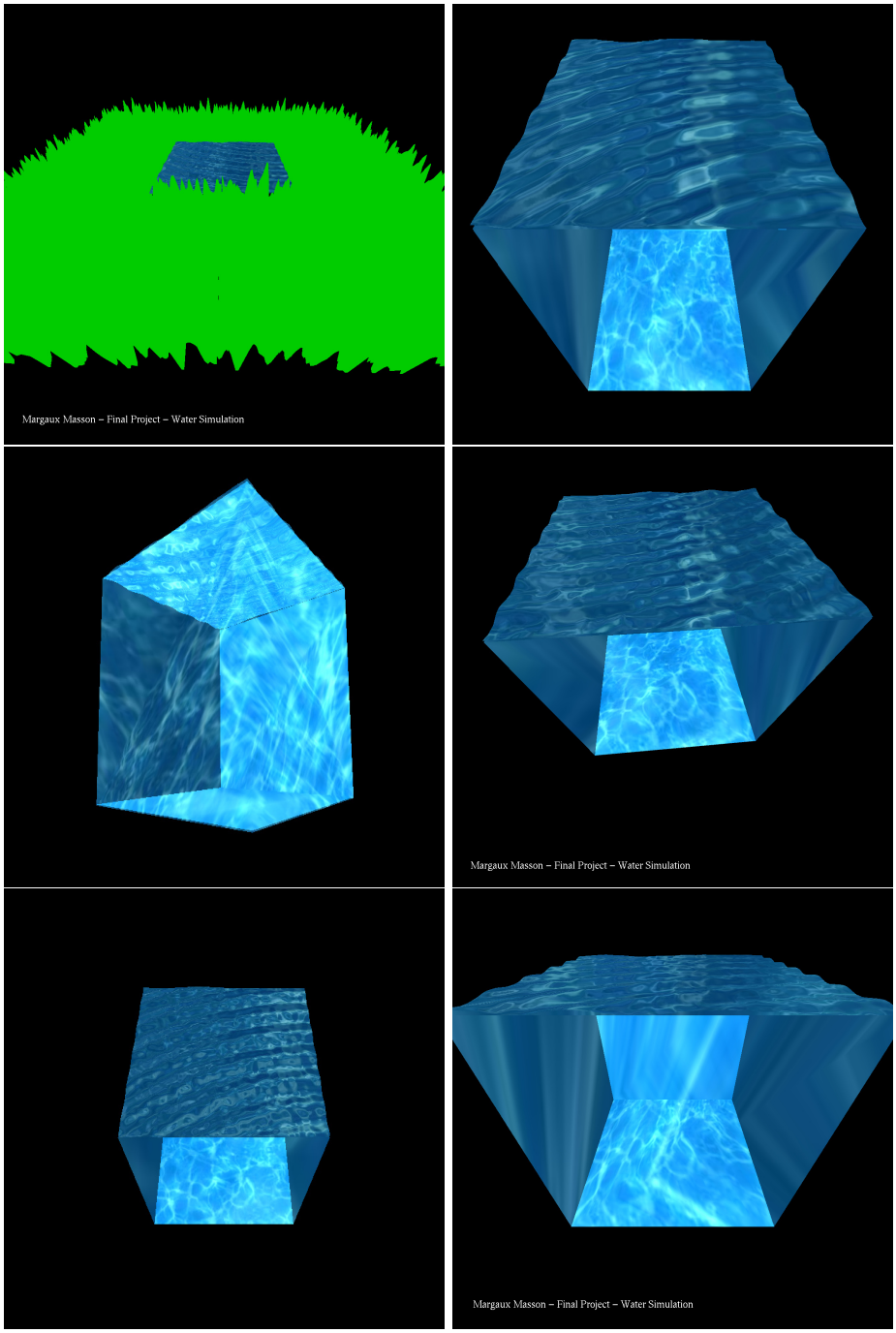
3.2.2 Video

Link to the video:

Click here

3.3 Images


Margaux Masson – Final Project – Water Simulation
Margaux Masson – Final Project – Water Simulation
Margaux Masson – Final Project – Water Simulation
Margaux Masson – Final Project – Water Simulation


Margaux Masson – Final Project – Water Simulation
Margaux Masson – Final Project – Water Simulation
Margaux Masson – Final Project – Water Simulation

Bibliography

[1] Simple water rendering *Simple water rendering*. Tutorial.