Water Rendering Through Vertex Texture Displacement

Jose Antonio Frias Castillo | Advanced Rendering Techniques | 14/12/2021

Main Problem

Many research papers have appear over the years trying to replicate water behavior as realistically as possible, however many issues arise during this research, maybe computational cost is too great such as with FFT formulas which end up in a result as in Figure 1, maybe the results are not realistic enough for an ocean let's say because the surface is a plane that just simulates the rippling and reflective effect of water such as in Figure 2, so the issue this paper from GPU Gems 2 is trying to solve is how to simulate moving water over vast regions without sacrificing neither visual quality neither being computationally expensive, to achieve results similar to FFT formulas but with a lower cost, these results can be seen in Figure 3

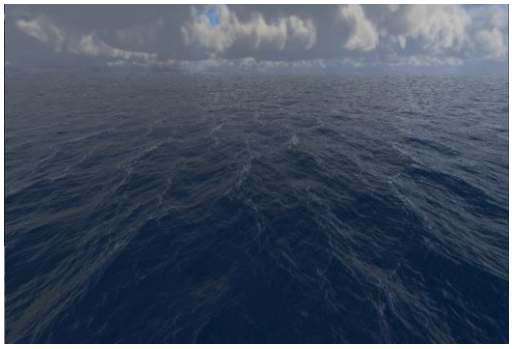

Figure 1 result with FFT

Figure 2 result of not displacing vertices

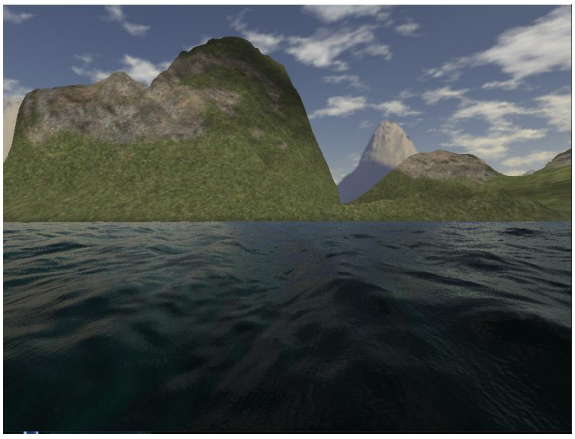

Figure 3 Result of solution by paper

How to approach the problem

The method used in the paper is simple and divided in some steps.

GRID CREATION

Firstly is the grid creation, although in the article it talks about a radial grid, for my approach since I didn't have any artist making height maps for me and the ones online didn't match specially well the shape for the radial grid, I went for a square grid which is easily tileable and its size is easier to adjust for shores of the ocean which is what this method aims for. The algorithm (Figure 4) for the grid is nothing complicated at all, just iterate through a 1 by 1 area dividing it in triangles to be easily drawn with OpenGL.

```
for (int j = 0; j <= GridSlices; ++j) {
    for (int i = 0; i <= GridSlices; ++i) {
        float x = (float)i / (float)GridSlices;
        float y = 0;
        float z = (float)j / (float)GridSlices;
        verticesGrid.push_back(glm::vec3(x, y, z));
    }
}

for (int j = 0; j < GridSlices; ++j) {
    int row1 = j * (GridSlices + 1);
    int row2 = (j + 1) * (GridSlices + 1);
    for (int i = 0; i < GridSlices; ++i) {


        indicesGrid.push_back(glm::uvec3(row1 + i, row2 + i + 1, row1 + i + 1));
        indicesGrid.push_back(glm::uvec3(row1 + i, row2 + i, row2 + i + 1));

    }
}
```

Figure 4 with code for generating the grid

VERTEX DISPLACEMENT WITH HEIGHT MAPS

To simulate a wave's motion and the bumpiness of the ocean different height maps are used. Firstly to simulate the main motion of waves two height maps are used, these height maps can't be simple Perlin noise height maps, they need to have some kind of wave shape and they are usually done by artists, the higher the resolution with the higher detail the less noticeable a repeating pattern will be. This allows artists to easily modify the behavior of the waves in the ocean with just an image. In my case I use two height maps moving on opposite directions, while one of them has more noticeable waves design, the other is used in the opposite direction to simulate the effect as when a wave goes by, there is a little impulse going the opposite direction so to speak. When sampling from them an offset is added to the UV coordinates so that it simulates a moving effect, the sampling can also be modified with parameters to modify the waves speed and height, as it's seen in Figure 5.

```
float wavesHeight = texture(waves, vec2(UV.x, UV.y + wavesOffset / WaveSpeed)).r * WaveHeight;
float wavesHeight2 = texture(waves2, vec2(UV.x, UV.y - wavesOffset / WaveSpeed2)).r * WaveHeight;

wavesHeight = mix(wavesHeight, wavesHeight2, WhichWave);
```

Figure 5 with code to simple wave displacement height maps

Then apart from those two height maps that simulate the main motion of the waves, another sampling is done in the vertex shader to calculate the height, this sampling is done from two different maps that are linearly interpolated among 13 total in my case. These maps are easier to get since they could potentially be done just with Perlin noise, these maps attempt to simulate the bumpiness and roughness of the ocean surface, as the ocean doesn't perfectly behave as a single wave, there are little disturbances along crests of the waves and the whole ocean, which is what this attempts to simulate. This value obtained from these height maps is then interpolated with the one of the wave motion but taking more into consideration the value obtained by the waves motion. This sampling can be seen clearly in Figure 6.

```
float Sampled = texture(HeightMap1, UV).r;
float Sampled2 = texture(HeightMap2, UV).r;
float height = mix(Sampled, Sampled2, interpolateFactor);
height = mix(height, wavesHeight, HeightOrWave);
height *= 0.11f;
```

Figure 6 with code to calculate final height of vertex

NORMAL AND LIGHTING CALCULATION

After having a grid dynamically moving through height maps only one thing is missing, lighting, we don't want the see to look like it has a plain color and nothing is applied onto it, however, the normal of the grid is no longer the normal of a plane, since each triangle has been moved upwards or downwards depending on the height maps, so how to calculate it, here there are a couple different options:

- Calculate it on the fragment shader using dFdx and dFdy functions (the option I chose to go for in this occasion)
- Calculate it on a geometry shader per triangle by calculating how the triangle would be moved again with the height maps and therefore getting the normal of the modified triangle
- Tilt it a little bit towards the eye of the viewer to simulate back of the waves (the approach given by the paper).

After having the normal, then standard lighting calculations are performed to simulate light reflecting on the surface of the water. Since in this case no texture is used for the color of the ocean, a color is chosen as well as values for the specular, diffuse and ambient components, from there modifications are applied to get different tones of an ocean, darker, lighter, or whichever way is preferred. This lighting stage can be clearly seen in Figure 7.

```
vec4 foamVal = texture(foam, UV);

float Interp =  Height;

vec4 Color = mix(vec4(0.04, 0.45, 0.92, 1.0), foamVal, Interp);

vec3 diffuse = vec3(0.75);
vec3 specular = vec3(1.0);
vec3 normal = normalize(cross(dFdx(PosW), dFdy(PosW)));
vec3 viewDir = normalize(viewPos - PosW);

vec3 fragToLightDir = normalize(-LightDir);

float shininess = 512;

float diff = max(dot(fragToLightDir, normal), 0.0);
vec3 halfwayDir = normalize(fragToLightDir + viewDir);
float specAngle = max(dot(halfwayDir, normal), 0.0);
float spec = pow(specAngle, shininess);

vec3 ambient = vec3(0.15) * Color.rgb;
diffuse = diffuse * Color.rgb * diff;
specular = specular * spec;


if(WireFrame)
    FragColor = vec4(1.0);
else
    FragColor = vec4((ambient + diffuse + specular) * ColorModifier, 1.0);
```

Figure 7 with how to calculate normals and lighting values

POSSIBLE ENHANCEMENTS OR IMPROVEMENTS

Foam Simulation

This is already implemented in the simulation, although not quite clear with darker tones of the ocean or if the ocean has really low heights, but it can be seen specially well with lighter colors of the ocean and high waves.

This is accomplished by having a foam texture that is sampled and blended with the ocean color material depending on the height that the crest of that wave has, that's the reason why with low height waves it's barely seen if at all.

Instance Rendering

As this technique is easily tileable by putting multiple grids connected one with each other and the movement is the same in all of them, so the modifications done to the heights at UV coordinate (x,y) will be the same for all the grids, a potential optimization could be to have instance rendering and rendering them all at the same time, saving even more time and making it cheaper to compute and render it over vast open regions of ocean.

Wind simulation

This was not implemented but the direction that the wind is blowing and how it would affect the moving waves in the ocean could be easily simulated as the waves height map (the one that in our case only two are being used) is added an extra offset on the UV coordinates while sampling to add the movement effect, this implies that given a 2D vector for the wind in the x and z, with this you could modify easily which direction you are sampling of the height maps, therefore, modifying the direction that the waves are going so that they move towards the wind direction.

PROBLEMS FACED DURING DEVELOPMENT

The algorithm itself is quite simple so not many issues arose with it as a general rule, nevertheless, my main issue when implementing this method was the height maps. As I'm not an artist and I don't know how to create them I needed to rely on what I could find on the internet, which for the height maps used for bumpiness was not really an issue as they can be whichever height map really, however, the two height maps that are used for wave displacement have some special characteristics that almost no height map online could achieve, as these height maps need to be perfectly tileable for it not to produce artifacts when displacing the waves (which is currently in the demo), also these height maps would benefit from having a decent to high quality where you can put multiple waves and making less noticeable the repeatable patterns when tiling the grids.

SIMULATION AND ITS PARAMETERS

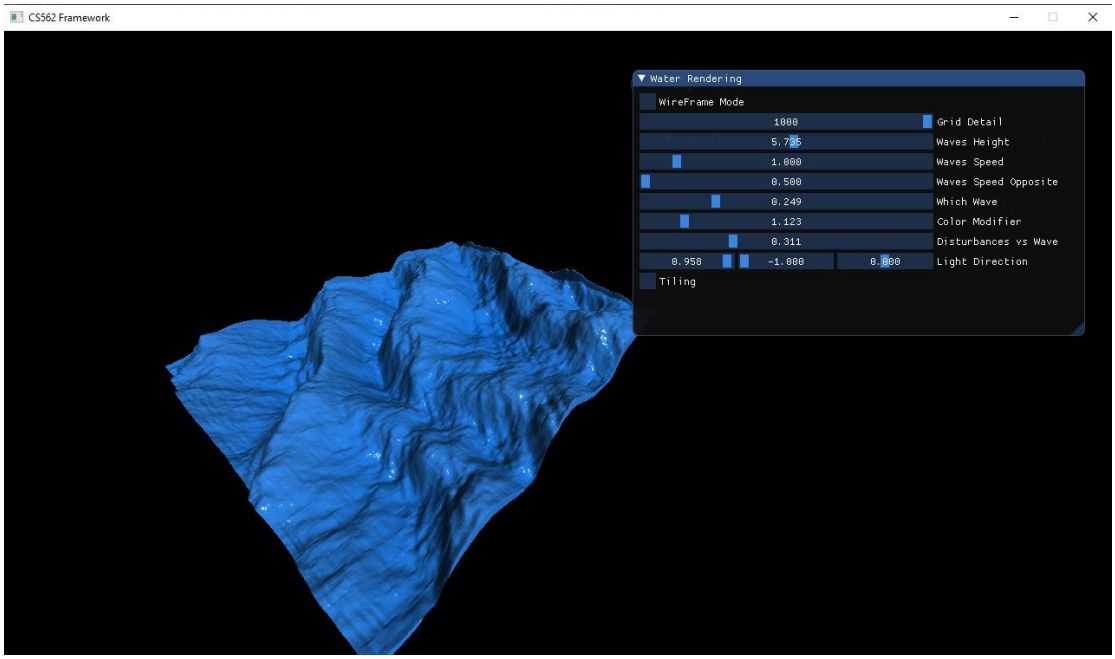

Figure 8 Demo

What can be seen in the Figure 8 is an example of the demo how it is right now with one grid and its parameters. Now what does each parameter affect:

- WireFrame Mode: Allows you to see the grid in wireframe mode, without lighting or colors in the triangles
- Grid detail: The amount of subdivisions inside the grid, so how low/high poly you want it to be
- Waves Height: Allows you to increase the altitude of the crests of the waves produced by the wave displacement height maps.
- Waves Speed/Waves Speed Opposite: How fast you want the two height maps that simulate the main wave motion to go towards each direction
- Which Wave: Allows you to choose which one of the two height maps used for wave motion should be favored
- Color Modifier: A modifier of the color produced by the lighting stage, to produce darker or lighter colors.
- Disturbances vs Wave: Allows you to choose which height should be favored, either the one produced by the wave motion height maps, or the one produced by the disturbances height maps, allowing you to go for a calmed ocean.
- Light Direction: The direction that the sunlight is going towards.
- Tiling: Allows you to simulate as if 9 grids were set one next to each other simulating a higher region of ocean.

CONCLUSIONS

While this method is simple to implement and highly efficient (as it only requires 4 texture fetches per triangle vertex and it can simulate better wave movement over vast regions), it has a disadvantage if you are using it for a project that

you are doing alone, and that is that you won't have any artist to provide you with height maps for waves displacement. This method highly relies on them for a good behavior of the waves and to camouflage repeatable patterns, so if you don't have an artist you might want to go for another approaches such as the one of the plane shown in Figure 2 which doesn't require tileable height maps to simulate the bumpiness, or for a more physics related method such as FFT (Fast Fourier Transforms) that no tileable height maps are needed and it indeed produces an even better result than this method, although at a higher computational cost.

So as a summary, if you have an artist to provide you height maps, these method can yield excellent results, if you don't have an artist for height maps then an approach such as the FFT is highly recommended. I leave in the end (after the bibliography) some images of outputs that I got with this method, showcasing the variety of behaviors for the ocean that can be achieved by tweaking parameters

BIBLIOGRAPHY

- https://faculty.digipen.edu/~gherron/references/References/Modeling/GPU_Gems2_ch18.pdf
- https://www.youtube.com/watch?v=HusvGeEDU_U&t=150s
- https://www.youtube.com/watch?v=CeJCNmI-B7s
- Helping websites such as stackoverflow and others for doubts that I found during the project but I didn't keep them all saved.

OTHER RESULTS

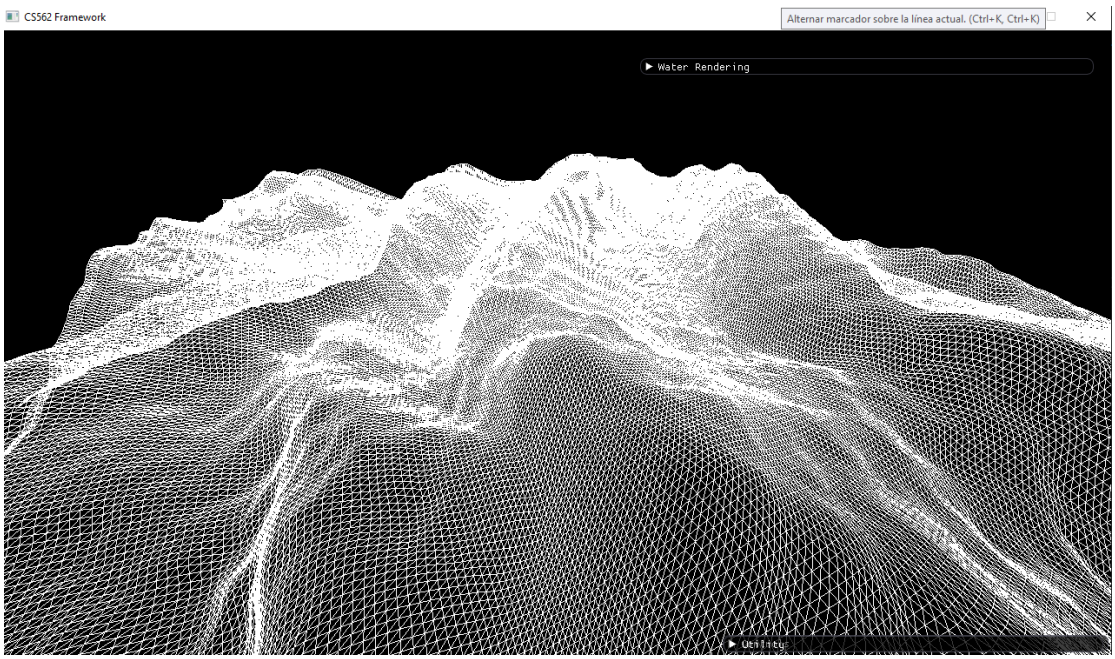

Figure 9 in wireframe mode to see grid movement

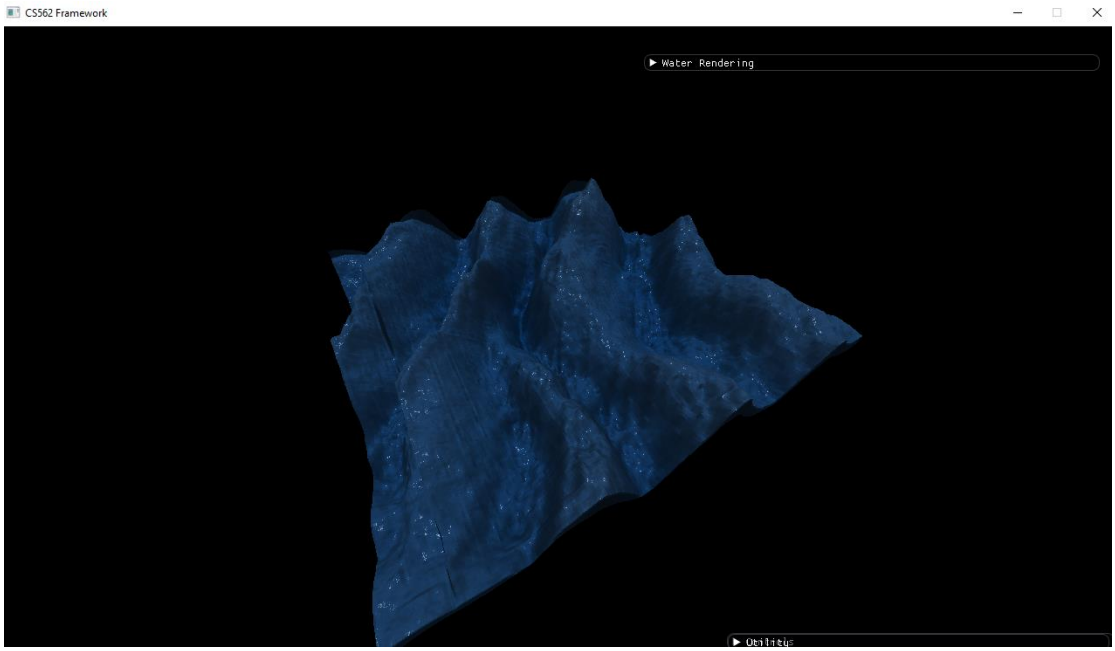

Figure 10 Simulation of a dark stormy night with high waves

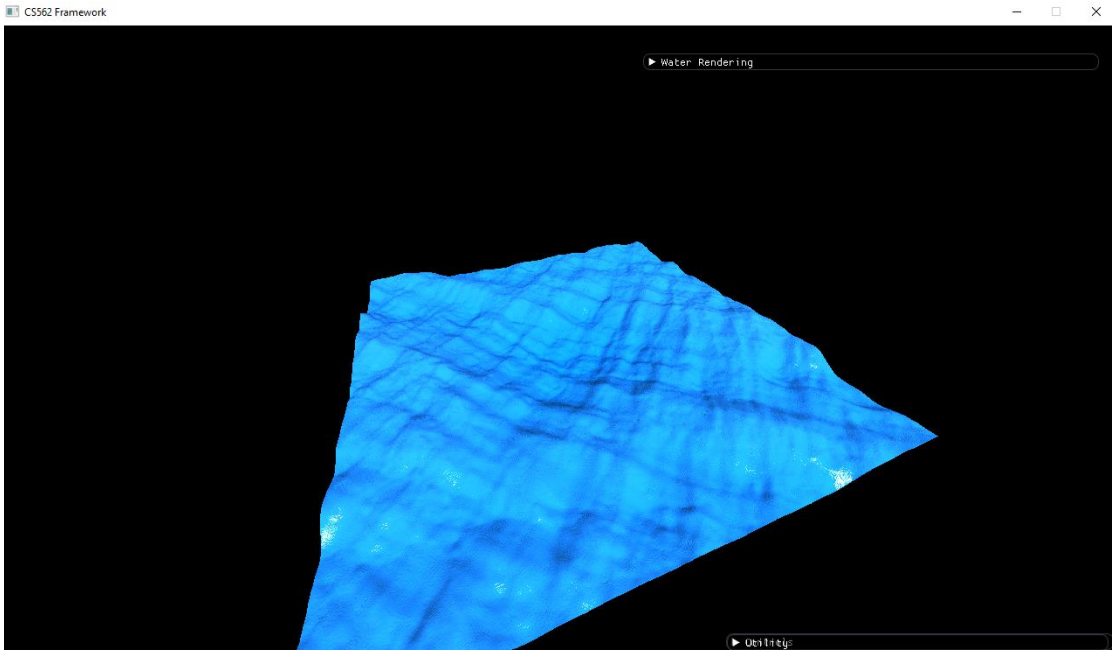

Figure 11 Simulation of a calmed ocean with only natural bumpiness and no waves

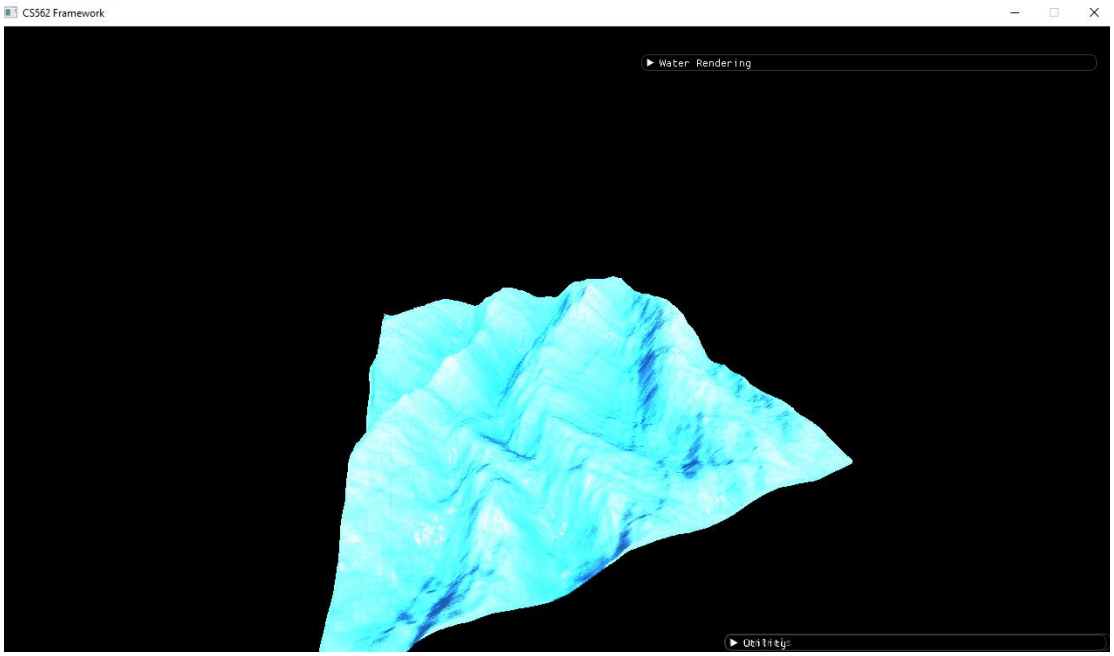

Figure 12 Simulation of hawaian clear waters with waves and the foam on top of them

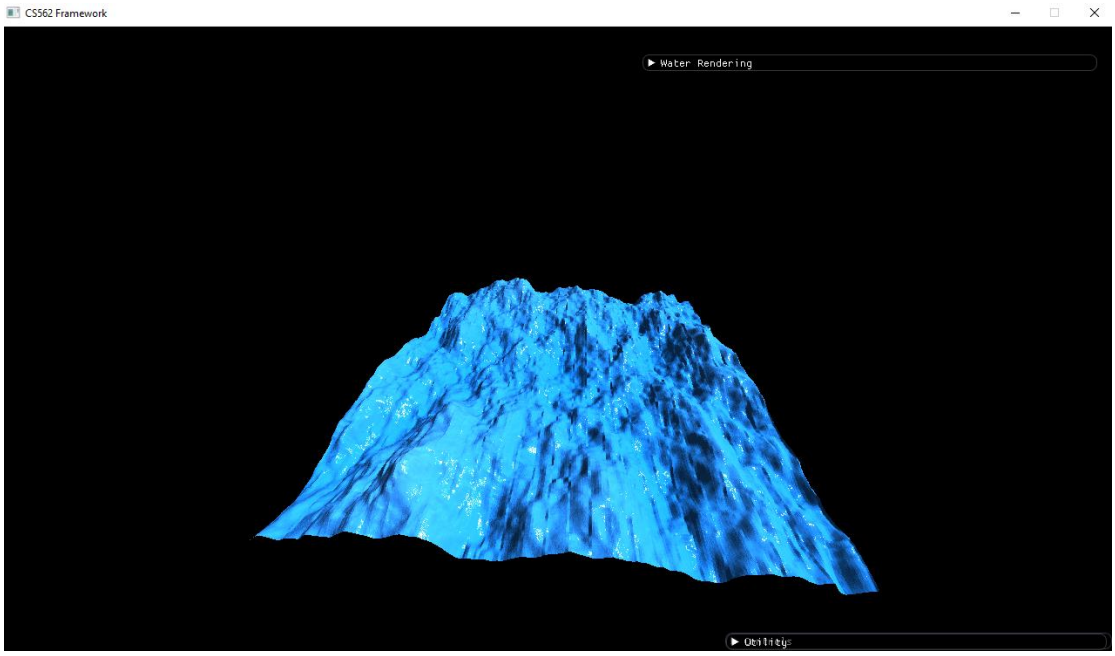

Figure 13 Simulation of what could be also a river (speed was going forward)

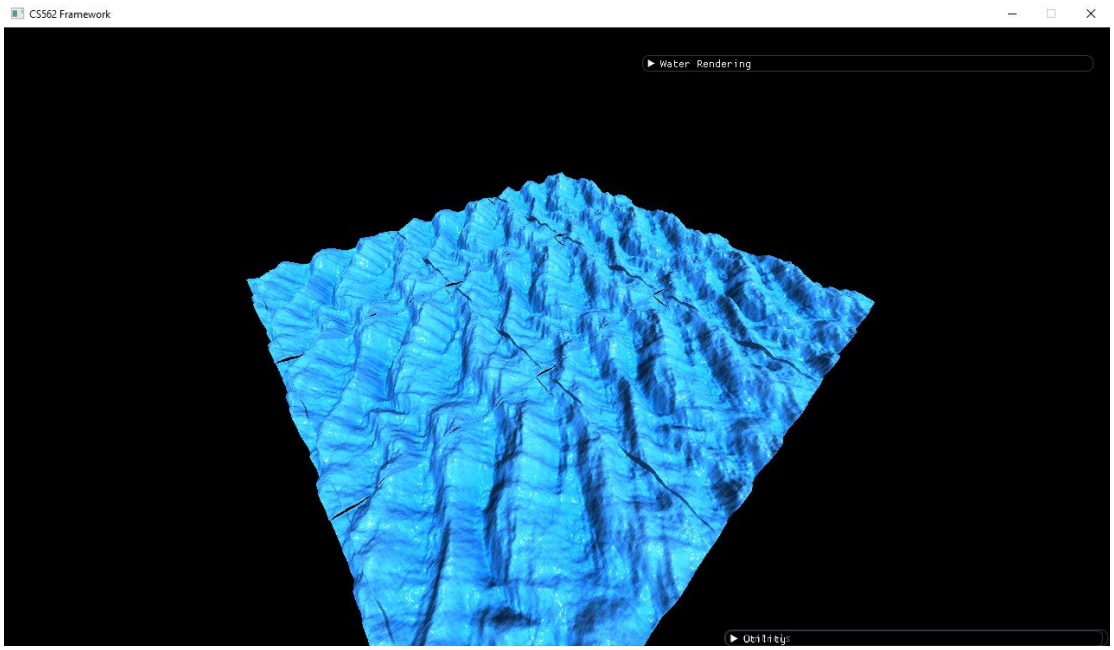

Figure 14 Simulation of a bigger ocean region with nine grids tiled together (making more visible the visual artifact of height maps not being tileable)