Water Simulation

Asli Baltaci

Pages 1–4
2022

Keywords: Shaders, Unity, Simulation, Water, C-Sharp

ABSTRACT

Choosing one of three topics—water simulation, city generation, or squad artificial intelligence—for this project (AI), water simulatic n included parts on fluid dynamics and phenomenological viewpoints. This study focuses on the phenomenological view of the water and the accurate recreation of it using the Unity engine. Two different methods for creating visually realistic water were evaluated, employing the Gerstner Wave formula both within a shader script and within a c script, based on the study described and according to a set of predetermined parameters. The final product was eventually produced by combining these techniques.

CONTENTS

1 INTRODUCTION

A crucial component of games is immersion. The idea considers how a player and an environment interact. They feel as though they are in the virtual world because it gives them a sense of spatial presence. Games that incorporate realistic behaviour from the real world, such as physics and visual elements, help players feel more immersed. Water is one component that has historically been challenging to replicate in games. Computer games frequently feature water surfaces. They are a crucial component that significantly raises realism. Yet, due to the tremendous visual complexity of water's velocity and the way light interacts with it, accurately portraying them is a challenge. A liquid that continuously deforms in response to an external force or imposed shear stress is referred to as a fluid. Liquids follow the fundamental fluid equations. Many characteristics offered by contemporary engines might help in re-creating water surfaces that look realistic. Water surfaces produced in 3D can benefit from vertex movement and texture.

2 RESEARCH AND BACKGROUND

To create compelling water, vertex alteration is necessary. In fluid dynamics, a Gerstner Wave is an exact formula for surface gravity waves. It is known as the solution to in-compressible gravity progressive waves of permanent form on a surface (Stuhlmeier, 2015). Troughs and peaks can be altered using this approach, producing a more believable simulation using the formula consisting of time t, variables a,b, constant g, and mass m:

$$x = a + \frac{e^{mb}}{m} sinm(a + \frac{\dot{g}}{m} t) \quad (1)$$

The vertex points of the surface do not 'flow' but move in a circular motion (Figure 1).

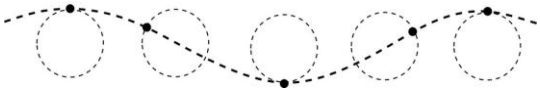

Figure 1. Gerstner Wave position movement (Constantin, 2013)

The simplification of this method due to the easily controlled parameters in the equation was examined in a thesis report utilising the Gerstner Wave implementation to reproduce wave simulation (Ming, 2010). To simulate realistic wave shapes, Ming was able to change the height, wavelength, and wave period. To blend many waves to get a realistic result, sine waves were found to be a more reasonable choice. The disadvantage of this method is that the required grid resolution prevents the GPU from processing the vertex effectively enough, making it impossible to replicate waves in great detail.

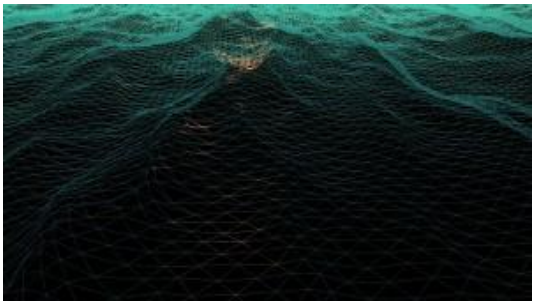

Figure 2. Lantz FFT wave outcome (Lantz, 2011)

A different strategy that makes use of the Fast Fourier Transform (FFT) was proposed by Keith Lantz (Figure 2). The FFT system makes use of many methods, including wave calculation at various degrees of resolution and texture scrolling. The discrete Fourier transformations can be quickly calculated using the FFT algorithm (DFT). Like an FFT, a discrete function transform (DFT) works. This technique is used by Lantz to produce wave height fields. This method's benefits include optimising transforms, making good use of the GPU, and including cutting-edge rendering techniques to help with light simulation. The approach produces accurate answers, but because it involves a lot of work, interactive applications cannot use it.

Through tutorials describing their approach to wave simulation, more background research was conducted. A distinct strategy was presented in a tutorial on the method of shifting vertices along a mesh to simulate flowing water (Flick, 2022). They combined moving vertices, several Gerstner Waves, and wave direction control to create their created wave. A wave that appears to be moving can be produced by animating vertex positions on a grid mesh, even while the mesh surface is still. To determine the proper levels of brightness, saturation, and colour for material rendering, shaders are straightforward computer programmes

that runs on the GPU. Several Gerstner Waves are implemented in this lesson using a surface shader inside of a function (Figure 3). The function changes the vertices' coordinates so that they are, along the surface, equal to the cosine and sine of x, respectively. Another tutorial by Flick described how they use a different shader that refers to them original shader to create underwater fog and refraction. Clearwater needs a transparent shader because it is translucent. It is possible to create the appearance of a translucent surface by altering the opacity. These two instructions can be combined to create water that looks aesthetically realistic.

```
float3 GerstnerWave (
    float4 wave, float3 p, inout float3 tangent, inout float3 binormal
) {
    float steepness = wave.z;
    float wavelength = wave.w;
    float k = 2 * UNITY_PI / wavelength;
    float c = sqrt(9.8 / k);
    float2 d = normalize(wave.xy);
    float f = k * (dot(d, p.xz) - c * _Time.y);
    float a = steepness / k;

    //p.x += d.x * (a * cos(f));
    //p.y = a * sin(f);
    //p.z += d.y * (a * cos(f));

    tangent += float3(
        -d.x * d.x * (steepness * sin(f)),
        d.x * (steepness * cos(f)),
        -d.x * d.y * (steepness * sin(f))
    );
    binormal += float3(
        -d.x * d.y * (steepness * sin(f)),
        d.y * (steepness * cos(f)),
        -d.y * d.y * (steepness * sin(f))
    );
    return float3(
        d.x * (a * cos(f)),
        a * sin(f),
        d.y * (a * cos(f))
    );
}
```

Figure 3. Gerstner Wave Function (Flick, 2022)

3 METHOD

For this project, Unity 2019.4.17 was utilised. To get started, a plane was added to the scene with two scripts attached, one of which calculated the wave mesh and controlled wave height, and the other of which generated two waves by computing their cosine and sine, as well as their speed and amplitude. The Gerstner Wave was incorporated into the C script using the study mentioned above. Due to the wave's edges being triangular rather than smoothly rounded, it gave the impression that the water was pushing and tugging in an artificial way (Figure 4). It was decided to create a temporary shader graph to represent the intended aesthetic.

Figure 4. Gerstner Implementation Result

This outcome led to the scripts being completely deleted and the creation of a surface shader. A vert function was added to the base script to allow for the manipulation of vertex data. This gives us the ability to alter the vertices' input and output. Both the x and y axes are moved by waves. The plane generates an output similar to Figure 4 when using the sine wave and other elements originally programmed in the C script, but when using Flick's formula:

$$k = \frac{2\pi}{\lambda} \tag{2}$$

We have simple control over wavelength, and as a result, it makes a good shader property. In this instance, the wavelength and speed are both set to 10. Inside the amplitude calculation, the speed must be multiplied by the time to change how the plane moves. We must change how the vertex positions are calculated so that the wave seems to move horizontally rather than vertically. It causes this effect by converting the initial x amplitude to a cosine. Nevertheless, altering it from a float to a range between 0 and 1 simplifies things and makes modifications easier. This was renamed Steepness. The lighting does not change as the plane advances, making the water appear flat and resembling the wave in Figure 4. Vertex normals are characteristics of the mesh that can include UVs, tangents, etc. and dictate how the mesh will appear when rendered. To give the appearance of light interaction, the tangents of the wave must be normalised because the mesh moves along the x and y-axes but stays stationary along the z-axis (Figure 5).

Figure 5. Wave After Vertex Normals and a direction vector are Implemented

Because waves do not typically travel in a straight direction, we can add a direction vector to the script. Using a rearrangement of a position calculation to a vector3, the implementation of d.x and d.y can replace variables within the x and z-positions. For clear and concise coding, all normals were placed into a calculateNormal function, and all parameters that create the wave were put in a makeWave function. At this stage, the wave has attributes of movement, but no visual accuracy about transparency and reflection. Transparency of the mesh can simply be done by adding 'transparent' the renderType and Queue within the tags of the script.

A cginc file uses helper functions that allow light and shadow functionality within a surface shader. to create depth and access this, we have to add a sampler2D variable, as well as a float4 that gets the screen coordinates of the fragment on the z-position. However, to get a depth difference, creating a background and surface depth using a LinearEyeDepth and converting it to a linear depth is needed. This calculates the distance between the water and the background and as a result, makes a gradient (Figure 6).

Figure 6. Wave Using Cginc File

4 EVALUATION

Due to production-related problems, the plane appeared triangular and moved artificially, making it appear unrealistic. These problems were caused by the methodological approach of adjusting the mesh's vertices. The main issue that necessitated restarting the project was a lack of vertices on the mesh, which made it appear unnatural. An increase in vertices would make the mesh appear more natural but would result in significant GPU utilisation.

A thorough comprehension of the Gerstner wave was a minor concern as well. Due to a lack of knowledge, the surface shader methods and the code had to be refactored rather than starting from scratch. Despite this, a strong grasp of both developed over time and led to a high-quality final product, representing an accurate visualisation of water. Before beginning the project, more time must be spent on research and a thorough understanding of the process to avoid this in subsequent initiatives.

5 CONCLUSION

Realistic water in video games is important to create immersion for players. A game's sensory experience can be enhanced by water. The movement and reflection of light on water, for instance, can enhance the visual experience. There are several ways to simulate water, including the Gerstner Wave, which helps produce realistic waves, and the Fast-Fourier Transform, which makes good use of the GPU.
Despite the project's understanding and vertices problems at the beginning, the technique employed for this project's simulation helped produce a realistic wave using a plane and surface shaders.

REFERENCES

1. Raphael Stuhlmeier (2015) Gerstner's Water Wave and Mass Transport. Available at: https://www.researchgate.net/publication/282547102_Gerstner's_Water_Wave_and_Mass_Transport Accessed 27 October 2022
   AdrianConstantin(2013)*SomeThree* − *Dimensional NonlinearEquatorialFlows.Availableat* : *https* : *//journals. ametsoc.org/view/journals/phoc/*43/1/*jpo* − *d* − 12 − 062.1.*xml*[*Accessed*28*October*2022]
2. Keith Lantz (2011) Ocean simulation part two: using the fast Fourier transform. Available at: https://www.keithlantz.net/2011/11/ocean-simulation-part-two-using-the-fast-fourier-transform/ [Accessed 5 November 2022]
3. Jasper Flick (2022) Waves Moving Vertices. Available at: https://catlikecoding .com/unity/tutorials/flow/waves/ [Accessed 17 October 2022]
4. Jasper Flick (2022) Looking Through Water Underwater Fog and Refraction. Available at: https://catlikecoding.com/unity/tutorials/flow/looking-through-water/ [Accessed 17 October 2022]
5. Chen, Ming (2010) Fast, interactive water wave simulation in gpu for games. Available at: https://dr.ntu.edu.sg SCE_THESES_3PDF
6. Accessed 5 November 2022