Max Vendel
S0007E, Inriktningsprojekt spelutveckling, Lp3, LTU
2021-05-21

Generating Ocean Waves and Simulating Buoyant Actors

Max Vendel

Max Vendel
S0007E, Inriktningsprojekt spelutveckling, Lp3, LTU
2021-05-21

Abstract

For my specialization project I've chosen to create an ocean surface simulation and a buoyancy system acting upon said simulation. I've chosen to work on this project as I have been interested in generating a realistic ocean surface and buoyancy system for a while due to a game idea that I've been working on. The Ocean Consists of a procedural mesh grid with a shader applied to it. I chose to follow a few tutorials published by CatlikeCoding for many aspects of the water shader. For buoyancy I decided to follow a paper published by Jacques Kerner on Gamasutra as well as an interpretation of that paper made by Erik Nordeus on Habrador.com. For the steerable ship I decided to do a mix of the Habrador paper solution and a solution of my own, implementing a rudder and a force applier using a unity hinge joint.

Table of Contents

Mesh/Texture

Grid Generation

Whilst there are many different methods for generating mesh grids, I settled on making a standard procedural rectangular grid.

The requirements for the grid is that it should be scalable for different resolutions so that I can have grid chunks of different resolutions at different distances from the player. After making several iterations and attempts I ended up following the solution provided by Erik Nordeus. Having the size of each cell decided by the overall size of the grid and the cell amount. allows us to generate grids of equal sizes but varying resolution.

Texture Distortion

On top of displacing the vertices it's important to move and shift the texture of the ocean surface to give the illusion of a liquid. visually this is one of the more important aspects of the ocean simulation as even a grid absent of vertex displacement will give off a visually interesting surface

The water surface consists of two interchanging sliding textures for both main and normal. Each texture has directed sliding using a flow map, this leads to the texture sliding in different directions in different places depending on what part of the flowmap its sampling from. We apply this method to both the main texture and the normal texture.
To mask the switch between the two interchanging textures I ended up using the alpha channel of the flowmap for weight used for time offset, this results in the textures switching at different times in different places.

Ocean Shader

Vertex Displacement

One of the biggest parts of the ocean shader is the vertex displacement. There are different functions that can be used for this purpose. FFT(Fast Fourier Transform), Gerstner Waves, even Sine Waves. I chose to use Gerstner waves as I've had some previous experience with them. We end up displacing each vertex in the generated grid on both X,Y and Z by applying the Gestner Wave function to each vertex. The displacement is done on the GPU for performance.

Gerstner Wave Functions

Gerstner waves are several trochoidal waves summed up to create a complex wave pattern over a surface, not unlike the base frequencies in fourier transforms combining the final output.
As a single trochoidal wave moves the vertices of the mesh in the XY in the shape of a circle to give an illusion of a wave, the Gerstner wave does the same but in all three dimensions. moving each vertex in a sphere around its original position.

Depth Fog

The catlike coding tutorial on depth fog/refraction mentions two ways to add depth fog. Either using a global fog and applying it to everything that gets rendered before the water itself or applying the fog while rendering the water surface. As the first method is not suited for an ocean of varying height(e.g. physical waves) I chose the second approach, which coincidentally is the same approach that the paper covered.

I started out by figuring out the depth of the objects/terrain below the water. We can do this by grabbing the camera's depth texture. and comparing the depth to the position of the water surface on each fragment. This lets us visualize the distance between the waters by blending the color of objects under water with a secondary color, in our case blue.

“Fake” Refraction

the most accurate way to make refaction would be by ray-tracing each fragment of the water surface and applying refraction to each ray to figure out what the bent light will hit. As this would be very expensive I chose to follow the paper's approach and make an approximation instead.
The refraction works by jittering the UV coordinates used to grab the background.
We start by grabbing the tangent space normal vector as the offset per fragment and then we multiply the Y with our depth buffer before returning the UV output. By doing this we can get something that looks close enough to real refraction.

Buoyancy Physics

Sampling water height

The problem with Gerstner waves in contrast to Sine waves is that we only have the displacement data and not the height of each position. so if we sample the height of the displacement at some point it will actually return the height from the displaced point. The solution I found was to sample the point several times and use the delta to resample in a slightly shifted position. doing this 5 times over. This means that each vertex on the actor measured will be 5 times as demanding. After being able to sample any point in the X,Z space to return, said Y, we can sample the ship's Vertices to see which points are submerged and what points aren't.

Triangle Intersect Algorithm

The main task of the triangle intersect algorithm is to determine the intersection between the surface of the water and the surface of the hull. it then checks each triangle in our hull mesh. Any triangle with all 3 vertices submerged is submerged, and one with all 3 vertices above the surface is not submerged. any submerged triangles get logged for later usage.
The triangles that have either 1 or 2 of their vertices submerged are surface breaking triangles. These triangles are split up into smaller triangles using the Triangle intersect algorithm. It's important to note that this intersection algorithm approximates some things to remain performance friendly. and will therefore not approximate some triangle cases as precisely as it could have.
We start by ranking the points of the triangle by height, designating them as High, Middle or Low. We then calculate the height to the surface for each point. At this point the cases will be different depending on if the triangle has 1 or 2 points submerged, but the principle is the same. Between the center vertex and the side vertices we create 1 new point respectively specifically using MI and LI or MI and HI depending on the orientation of the triangle.
With some rudimentary points along the sides of the triangle marking the waterline we can now split the triangle up into smaller triangles and log the submerged halfs into our submerged triangle index. Eventually I ended up saving the triangles above the water in their own list as well so I could calculate air resistance.

Force Application

When we have sampled the hull we can apply forces to the ship. There are 4 forces acting on the ship in total. the hydrostatic force, viscous water resistance, pressure drag forces and slamming force.
The hydrostatic force, also known as the buoyancy force, applies an upward force on the triangles underwater depending on their respective depth. The size of the force depends on the tangent, area and distance to the surface. As well as the water density and gravity multiplier.

Calculating drag is pretty straight forward as its just the **size of the surface** * **resistance coefficient** * **velocity squared** * **fluid density / 2**. This formula is applied to each triangle underwater.
Viscous water Resistance is the term for the force that occurs when the water flowing across a surface “sticks” to the surface itself. The further away from the surface the water is it sticks less and less.
The final force added is the water slamming force. it's there to simulate a sudden area colliding with the surface at a flat angle. this is to prevent a ship from shooting through the water surface if it would fall a substantial length before making contact.

Ship Controls

I opted to make a physical rudder using a unity Hinge Joint. Using the rudders transform for an “Engine” I was able to generate force in the direction I wished. As a final feature I made the engine produce thrust only when it was under water. For the Controller I opted to phone a friend and grab his 3rd person character Camera. He also helped me understand unitys new Input System so I could set up the controls for the ship.

Result & Thoughts

While the final project lacks some polish and has missing features, I'm still very happy with what I was able to achieve. Regarding the water physics and point sampling I succeeded in what I set out to do, and I will be able to use what I learnt to set up ship/water physics in my own game projects from now on.

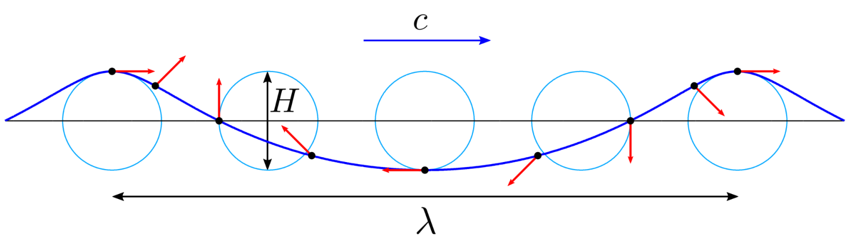
If I could do it all again I would focus more on the water graphics. including but not limited to fresnel, reflections and water caustics. I would also have been more serious about the planning phase. I would have started by reading as much of the HLSL documentation as I could as a wider knowledge base in shader programming would have helped me tremendously at the start of my project. I would also have implemented some form of structure in the beginning as I had to remake my project several times due to lack of structure in my programming/unity project.

So if i've learnt anything it's about the importance of planning. From how to proceed with implementation of research papers to a deeper understanding of water surfaces in games. Every step of the way I've learnt a lot.

Max Vendel
S0007E, Inriktningsprojekt spelutveckling, Lp3, LTU
2021-05-21

Attachments

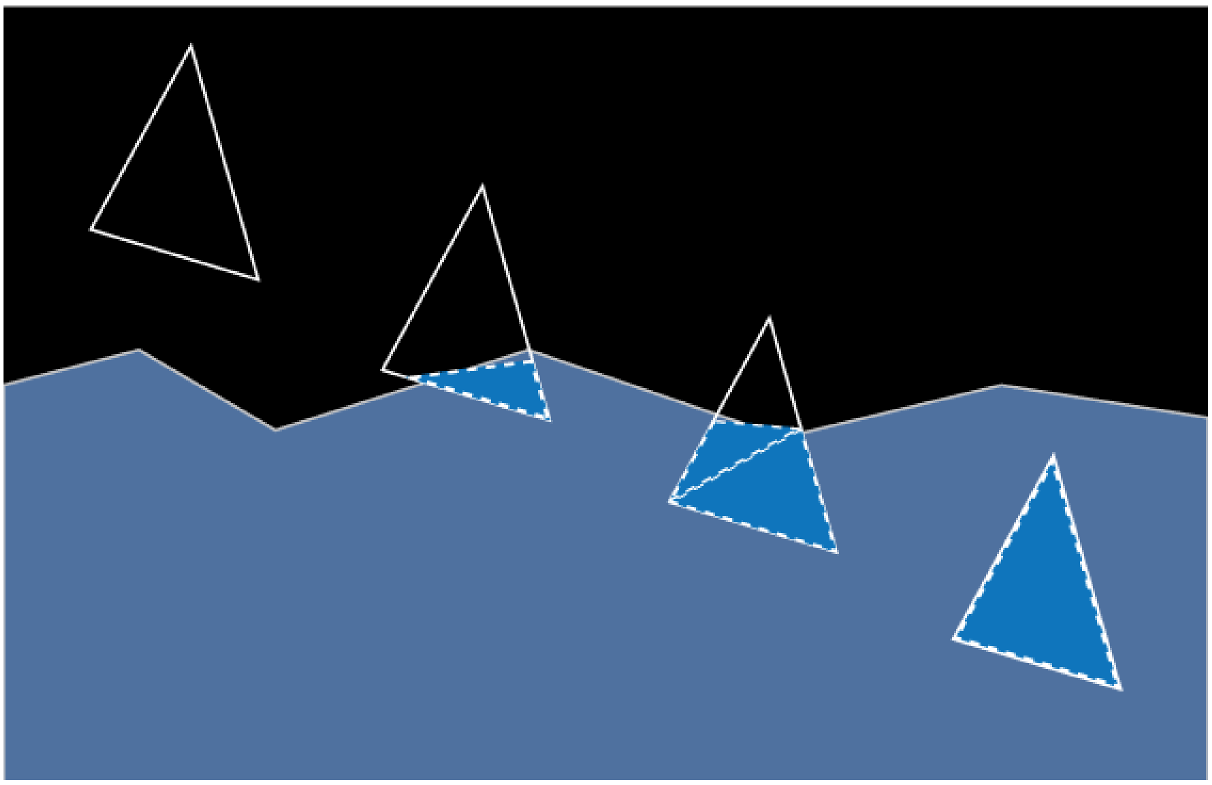
Trochoidal Wave visualized



Gerstner Wave Formula

$$\mathbf{P}(x,y,t) = \begin{pmatrix} x + \sum\left(Q_i A_i \times \mathbf{D}_i.x \times \cos\left(w_i \mathbf{D}_i \cdot (x,y) + \varphi_i t\right)\right), \\ y + \sum\left(Q_i A_i \times \mathbf{D}_i.y \times \cos\left(w_i \mathbf{D}_i \cdot (x,y) + \varphi_i t\right)\right), \\ \sum\left(A_i \sin\left(w_i \mathbf{D}_i \cdot (x,y) + \varphi_i t\right)\right) \end{pmatrix}.$$

Triangle intersect Algorithm



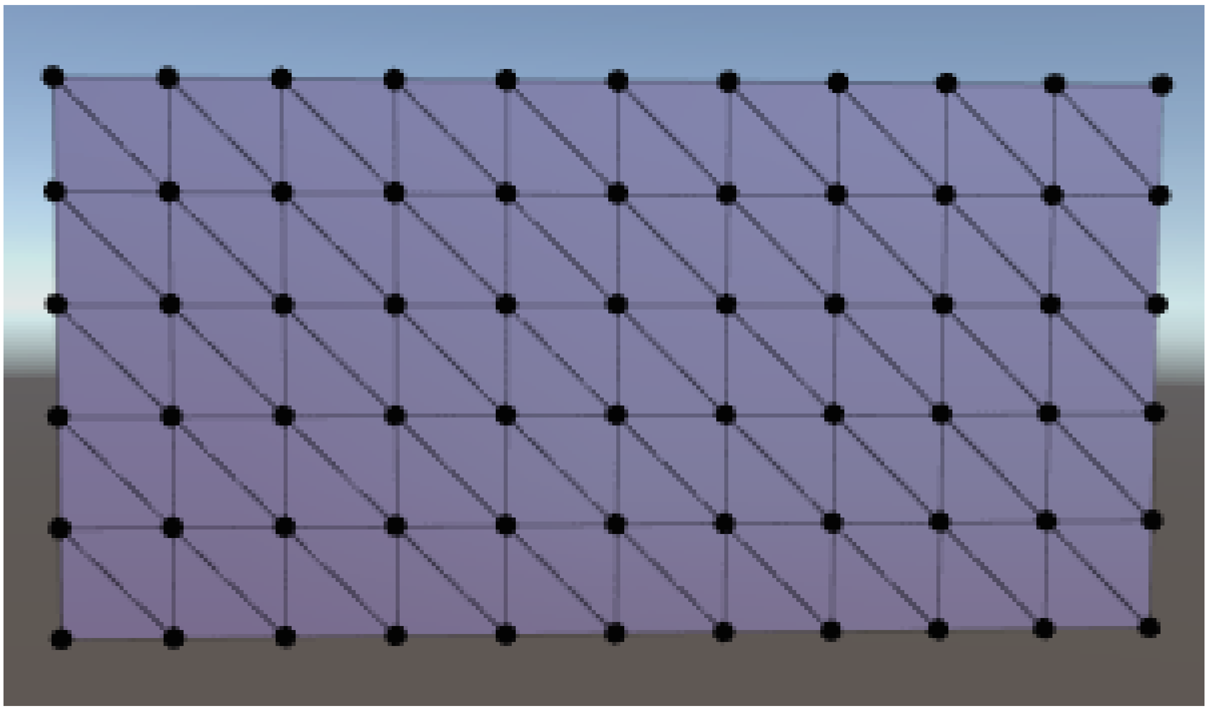
Drag Equation

$$R = \frac{1}{2}\rho C S V^2$$

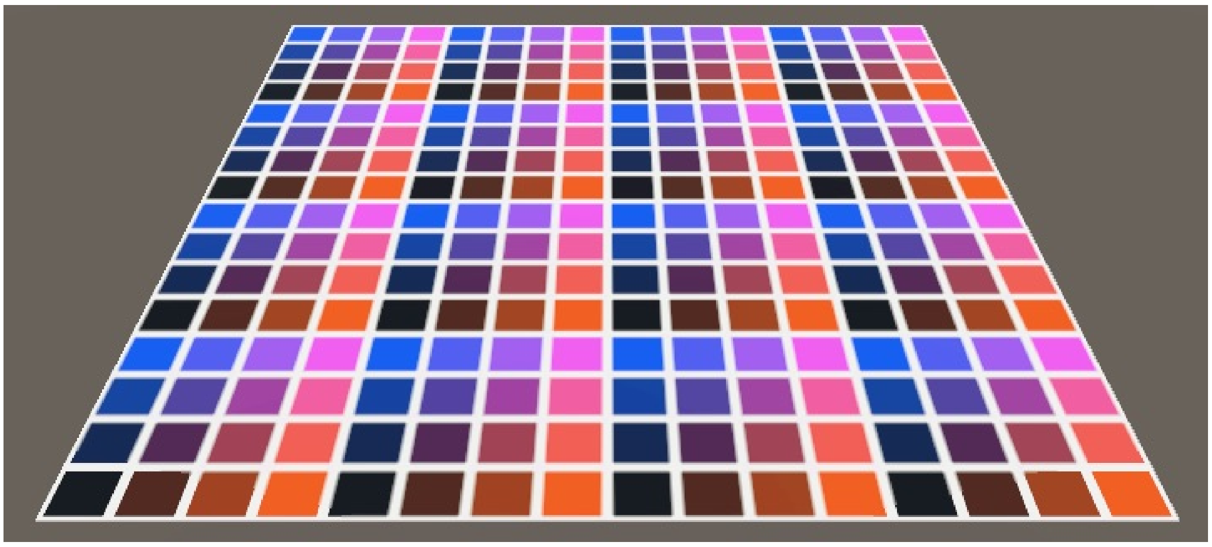
Viscous Water Resistance Equation

$$C_F(R_n) = \frac{0.075}{(\log_{10} R_n - 2)^2} \qquad (1)$$

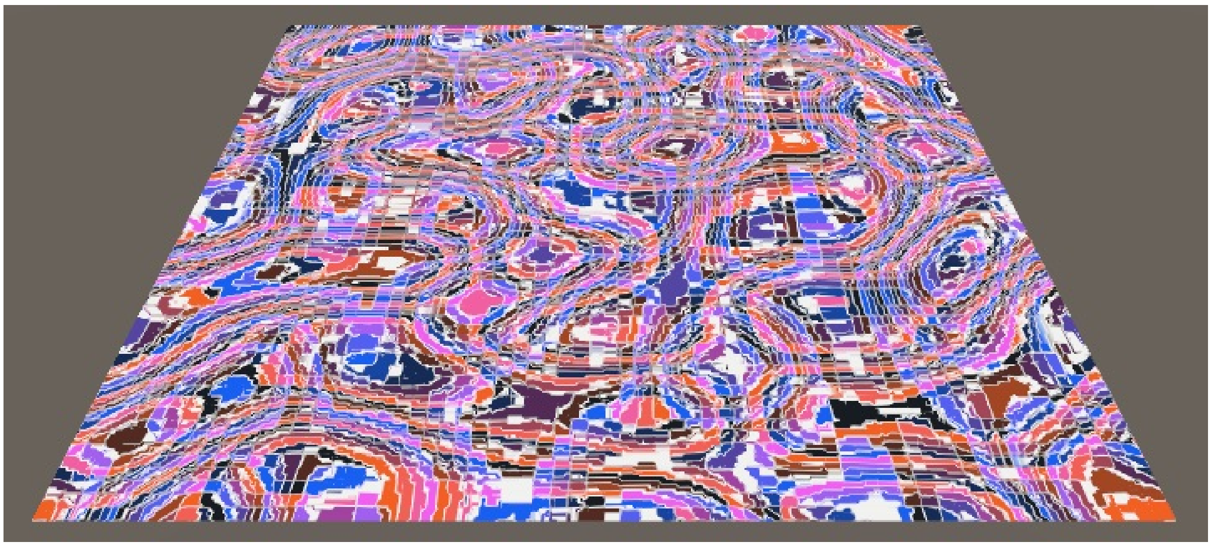
Procedural Rectangular Grid



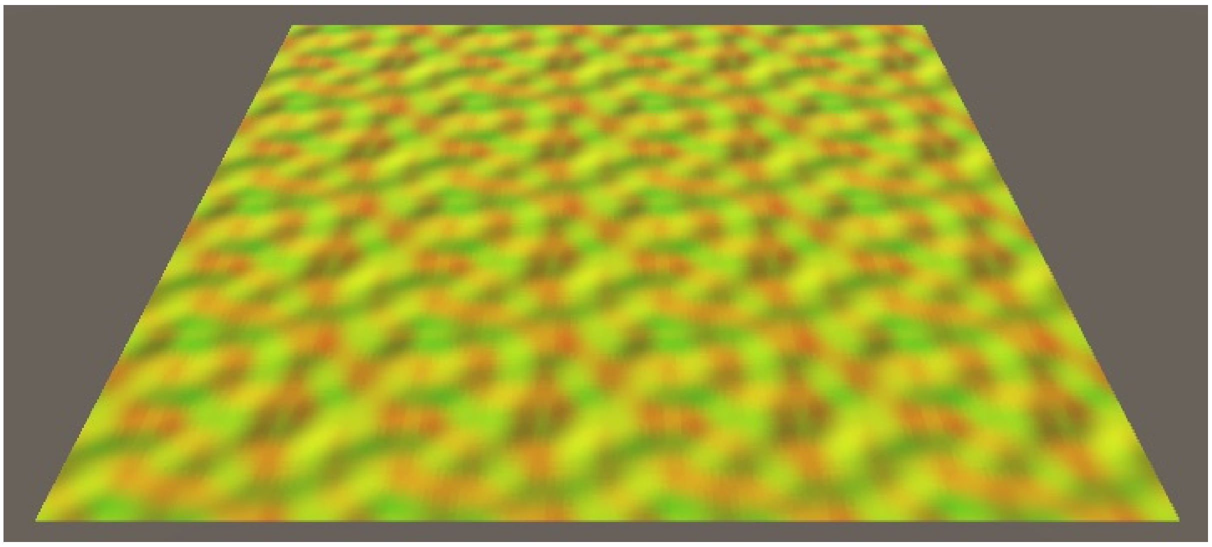
Standard Texture



Standard Texture with Directional flow



Flowmap



References

Catlikecoding: Texture Distortion:
https://catlikecoding.com/unity/tutorials/flow/texture-distortion/

Catlikecoding: Waves:
https://catlikecoding.com/unity/tutorials/flow/waves/

Catlikecoding: Transparency:
https://catlikecoding.com/unity/tutorials/flow/looking-through-water/

Gamasutra Buoyancy: part 1:
https://www.gamasutra.com/view/news/237528/Water_interaction_model_for_boats_in_video_games.php

Gamasutra Buoyancy: part 2:
https://www.gamasutra.com/view/news/263237/Water_interaction_model_for_boats_in_video_games_Part_2.php

Gamasutra Buoyancy: Unity:
http://www.habrador.com/tutorials/unity-boat-tutorial/5-resistance-forces/

Sampling GerstnerWaves:
https://youtu.be/kGEqaX4Y4bQ?t=803