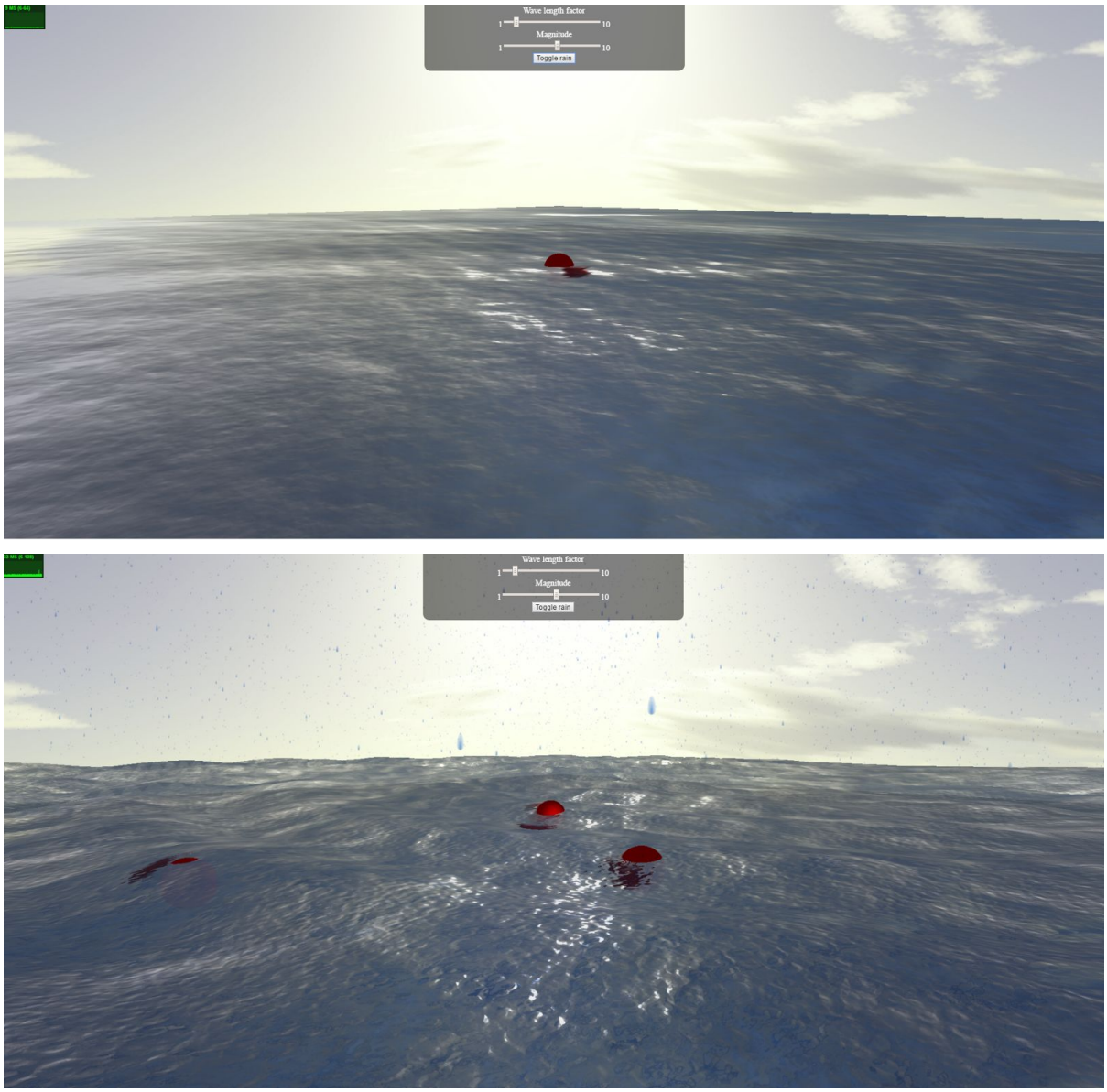
Motion in the Ocean



Group members:
Kevin Brundell Whittaker, kevinbw@kth.se
Carl Sténson, cstenson@kth.se
Emil Westin, emiwes@kth.se

Blog: https://tmito.blogspot.se/
Online link to project: http://www.csc.kth.se/~emiwes/DGIproject/

Introduction

In this project we aimed to explore the difficulties and possibilities of creating an interactive water surface. The project was developed for the web using threeJS, which is an open webGL framework for javascript. The project consists of four major parts: the water shader, simulation of waves, simulation of gravity and floating objects, and simulation of rain. All of these parts will be described further down in the report.

Fluid simulation is one of the more difficult parts of graphical programming to implement in a realistic and physically correct way. This project does not aim to create the perfect water simulation. Rather, we wanted to explore different factors in creating waves and water and see how those affect the user experience in our simulated environment. We seeked to find out which are the most relevant factors for creating the illusion of a water surface within the limitations of real time rendering in a web browser.

Today it is possible to generate computer graphic models of calm water that looks realistic enough to be parts of movies such as *Waterworld* and *Titanic* (Tessendorf, 2001). However to simulate more complex water with large amounts of spray, foam, breaking waves and splashing is a bigger challenge especially if real time rendering is the goal. Our project focus on rendering a calm water surface in real time.

Our implementation

During the development of the project, four major parts emerged, which also was the main focus of our work. The parts will be described and elaborated upon in this section.

Wave simulation

At an early stage of the project we implemented a simple function that generated a sinus wave (also called an s-wave). Basically it changed the y-value of each vertex on the water surface based on time. After some discussion in the group we decided that a sinus wave looked realistic, and was sufficient enough to fulfill our criterias for the wave to look natural. At this stage there was one fixed epicenter for the waves and the waves had some parameters that affected the spread, such as wavelength, magnitude, and decay, but where wavelength and magnitude were fixed variables.

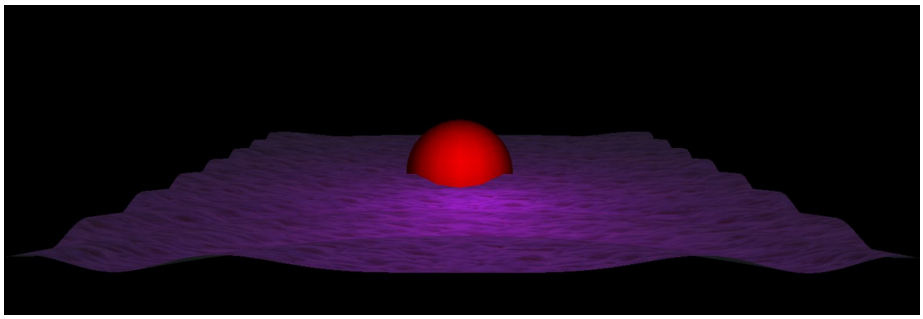


Fig. 1 - Simulated s-waves on the plane. One fixed epicenter is defined, and the whole plane is simulated at the same time.

As the project progressed, our wave function grew, and we tried to apply certain factors that would make the waves behave more natural. Up to this stage the waves depend on wavelength, magnitude and distance from each vertex and the epicenter of the wave. By setting different values for wavelength and magnitude different wave patterns occurs. A variable *decay* was also introduced, which affects the waves in such a way so that they, successively becoming smaller and fading out.

The dispersion formula for water waves, found at wikipedia looks like this:

$$n(x,t) = a * sin(f(x,t))$$

Where $\boldsymbol{f}(x,t) = (2\pi x / L) - wt$, and where L is the wavelength (in metres), w is the angular frequency (in radians per second), a is the amplitude (in metres), depending on the horizontal position (x , in metres) and time (t , in seconds) (Wikipedia, 2016a).

Our formula, at current stage, is written like shown in the figure below, and follows the dispersion formula. The *deltastep* variable is the difference in time between the creation of the epicenter and the current time in the model.

```
v.z += Math.sin(dist/wavelength - deltaStep) * vmagnitude;
```

Fig. 2 - Dispersion formula for each vertex on the plane. The local z-value corresponds to global y because the plane has been rotated 90° after its creation.

This particular line of code has not changed substantially since the first implementation and is responsible for all generation of waves in the mesh, which goes to show just how powerful math is once you get it right. While this line has remained the same, the surrounding code has been altered in order to facilitate multiple epicenters.

A difficulty we faced was to make the waves spread circular with center at the epicenter, and not have the whole water surface generating waves at an instant. We solved this by calculating the position of the wavefront and comparing the distance of each vertex on the plane to the wavefront. If the vertex distance is between the wavefront and the epicenter, its

y-value is changed and a sinus is simulated. Each vertex on the wave decays in height depending on time, creating the effect of declining wave height, see figure 3 & 4.

```
if(dist < wavefront){
    var vmagnitude = (magnitude*(dist/wavefront*decay));
    if(vmagnitude < 0.001 && deltaStep > 100){
        EPICENTERS.splice(j,1);
        break;
    }
    v.z += Math.sin(dist/wavelength - deltaStep) * vmagnitude;
}
```

Fig 3. - Magnitude of each vertex is declining over time.

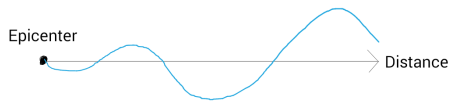


Fig. 4 - Wave decay

Of course, water waves does not behave as perfect s-waves. There are several kinds of waves that could be simulated, such as p-waves (Wikipedia, 2016b), gerstner waves (Wikipedia, 2016c), and choppy waves (Tessendorf, 2001). Gerstner wave slopes are steeper the higher the wave is, which make them sharp at the upper edges and choppy waves are peaked at the top and flattened at the bottom due to weather conditions. Both of these wave types would with most certainty have contributed in creating more realistic waves, especially in more wild water but in our project our focus was to simulate calm water, which behave more like an s-wave.but due to the scope of the project we did not try and implement them.

Surface shader

At the beginning of our project we aimed at creating our own shader that handled reflections and refractions in the water. Quite a lot of work was put in to understand the functionality of shaders in threeJS and to make some easy shaders by our own, but we have little to show for it. After assessing the amount of work required to create our own mirror shader (with some bugs), we came to the understanding that making a realistic water shader would require a lot of time and could very well be a project in itself would. This realisation lead to the decision of implementing an existing shader rather than creating our own. Writing a water shader from scratch simply was not feasible within this rather short project. Ultimately we decided to go with the shader *Ocean* by Jbouncy (GitHub, 2016).

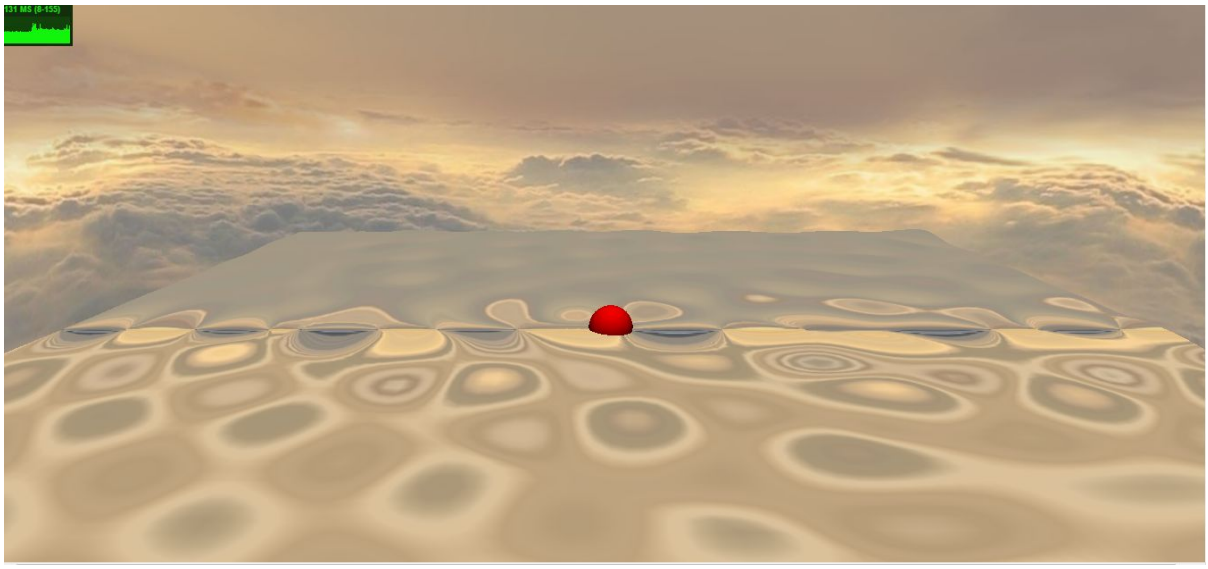


Fig. 5 - basic mirror shader with bugs

A shader in threeJS is written in web GL rather than JS, however, threeJS serves us with an interface. The shader consist of two parts: a vertex shader and a fragment shader, which both runs entirely on the GPU thus leaving the CPU free to do other computations, such as calculating the waves. (Lewis, 2016) The vertex shader handles processing per vertex. It returns a gl_Position which is the absolute position of the vertex on the screen. It can also do some setup work for later stages in the pipeline. In our case the vertex shader calculates the surface XYZ-position of the vertex which is later used in the fragment shader. The fragment shader is responsible for returning a gl_FragColor of which color the pixel should have. In this particular shader, most of the calculations are done in the fragment shader. The fragment shader uses a lot of input variables such as eyeDirection, waterColor and light direction. These variables are used to create a mirror effect. However, water is almost never a perfect mirror and this is accounted for by the use of a scrolling normal texture map. The normal map simulates ripples on the water surface according to the colors in the normal map **(figure 6)**. The fragment shader then adds some noise to translate the texture a little each frame. This results in the illusion of movement on the surface.

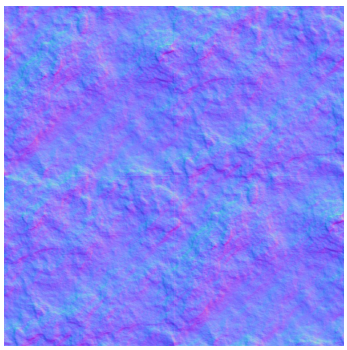


Figure 6 - scrolling normal texture

The final result of the shader implemented can be seen in figure 7.

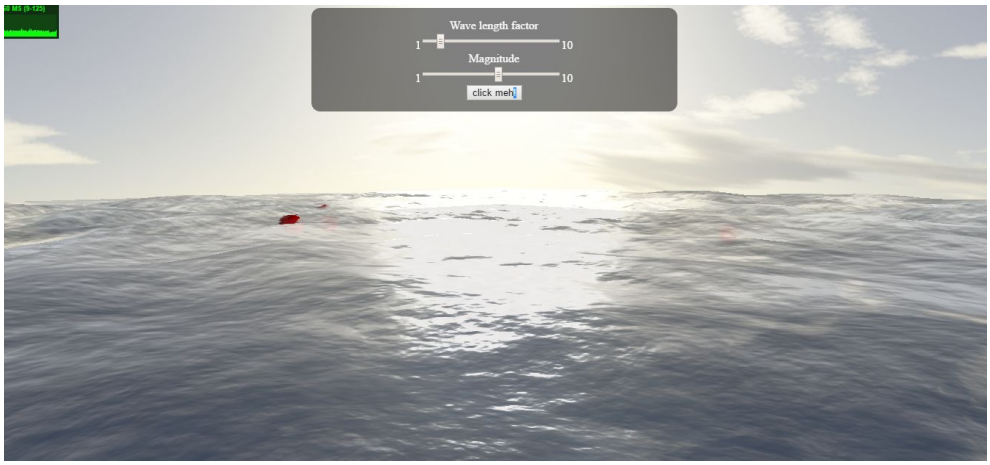


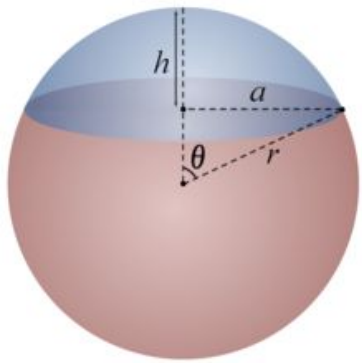
Fig. 7 - Result of water shader

Physics

In our scene we also included the ability to drop balls into the water. We simulate how a ball drops into the water and then floats on the surface, moving up and down. The very first implementation was to move the ball up and down like any other vertex in the wave, but slightly out of phase. This looked alright but was of course rather hard-coded. We decided to include some kind of physics to simulate it more realistically. We started out with implementing physics in the vertical direction and then moved on to horizontal direction as well, caused by the waves.

To simulate the movement up and down we implemented gravity in every frame by adding a velocity downwards. The water surface then needs to "catch" the ball and this was implemented from calculating the ball's buoyancy. This force is calculated through calculating the weight of the volume of the sphere that is under water. This is done by the following formula:

$$\frac{1}{6} * \pi * h * (3 * a^2 * h^2)$$



Where ***a*** can be calculated through:

$$a = \sqrt{h(2r - h)}$$

(Mathworld.wolfram.com, 2016a)

This calculated volume we used as the buoyant lift force. The
force can now be applied to the vertical velocity of the ball which then will decrease downwards to then start accelerating upwards. With this in place the ball will now be "catched" by the surface and bounce up and down with subsidence magnitude. These concepts are close to Archimedes' principle and Newton's gravity law, but with some

simplifications. The ball's only affecting property is its volume, it has no specified mass (rather the mass 1). Strictly speaking there are no forces acting on the ball, only the resulting acceleration. This has some negative implications, the ball will actually bob up and down forever even if the surface is entirely flat and by resetting the surface with the ball just above it will make it float very high, because the fall is not enough to build up sufficient downward speed. The model does not account for surface tension either.

When it comes to the movement in the other directions we discussed a lot of how a ball would actually react to the waves or if it is mainly the wind that affects it. Our conclusion is that wind is a big factor in this that we haven't accounted for, but that a ball will move a little with the waves mainly because of the wave slopes. Our implementation utilizes the normals of the faces in the water surfaces and then use the x and z component (according to world space coordinates) to change the velocity in x and z direction. To make the velocities decline we divide it by a constant that made it look realistic. In this way the xz-velocity will decelerate over time.

Rain

Towards the end of the project, we decided to implement rain into the scene. As a first attempt we decided to create a lot of small spheres at random x- and z-positions above the water surface. This seemed like a good idea since we already had some physics implemented for the ball. The result of this attempt was a low frame rate and severe lag due to a large amount of objects – around 200 – being rendered. After this failed attempt we decided to look into and implement a particle system which utilizes 2D sprites instead of 3D objects. By changing our approach we could with ease create 100 000 raindrops without having the frame rate decrease to unacceptable levels. Each sprite accelerates towards the water surface and when its y-value is equal to the the y-value of the plane, we reset the raindrop to a random position above the water surface. In order to accommodate waves and floating balls while raining we lowered the amount to 50 000 sprites.

We then had to consider how we should create ripple effects from the impacting raindrops on the water surface. It seemed unlikely that we would be able to handle so many epicenters. Firstly the wave equation is rather costly, but this could have been circumvented by limiting the affecting area of each ripple. However because the ripples are very small we would need a very high resolution mesh plane to be able produce said ripples, as per the Nyquist frequency (Mathworld.wolfram.com, 2016b). Instead we decided to find a scrolling normals texture that would give the impression of ripples.

To further optimize the process of falling raindrops, an attempt was made to define an area around the camera where the raindrops would be generated. Instead of randomly position the raindrops over the whole surface, an area of 1000x1000 pixels around the camera was

defined where the raindrops are generated. This allowed us to reduce from 50000 sprites to 30000 sprites while achieving a higher frame rate and a more intense rainfall.

When the defined area around the camera was fully functioning, we discovered a bug that arose when the camera was positioned in certain angles. At some times the particle system was not visualized at all, and we have, up to this point in time, still not figured out why.

Skybox & lighting

A skybox was created in order to establish a nicer surrounding of our environment. A shader was applied to the material, creating the illusion of the cube having round corners. A directional light was also inserted into the scene to make everything lighter and help our water shader get a nice looking reflektion on the water surface. A directional light is a light source that does not spread light from a certain position, rather it is sending light in a direction, creating the illusion of a sun giving light from a far distance. We also added a small ambient lighting to avoid having completely black areas in the scene.

Controls

This section describes the supported interactions in the project.

Create epicenters

Left click anywhere on the water surface to create a new wave epicenter. Wavelength and magnitude is set in the interface on screen before placing the epicenter. Instead of waiting for the waves to die out, the space key can be used to reset the water surface.

Dropping balls into the water.

Right click anywhere on the water surface in order to drop a ball on that position. The ball drops into the water and slowly floats up to the surface. Upon impact the ball creates a new epicenter which is relative to its size.

Camera controls

WASD pans the camera forwards, backwards and sideways. RF pans the camera up and down. The camera is rotated by the mouse, either by dragging on the screen or by first hitting the L-key and then move the mouse around. Hit the L key again to release camera control from the mouse movement. The camera can also be panned back and forth by the use of the mouse scroll.

Toggle rain

An on screen button toggles the rain effect.

Improvements

When looking at our initial aims and goals of the project we have fulfilled some, and some was not implemented. To start, we had the aim of creating a splash effect when objects were colliding with the water surface. We believe that our approach to creating the water surface made it hard to implement any kind of splash effect because it would need to be generated by a particle system. Therefore, we did not explore this further.

In order for us to get better feedback to our project we should have carried out a user study to help us further improve the user experience. In the study we would have given the participants the opportunity to play around in the scene. During their test we would have measured a little about how big waves they generated etc. Because we would like to know if the scene might be more realistic with some amplitude of waves. However, we still want the participants to interact with the scene because otherwise they will not get the feel that they can affect the surface which is one of our goals. A survey would collect quantitative data and interviews could be conducted if qualitative data is desired.

In the study concerning Tobii they only gathered quantitative data which we think not always is sufficient, especially when you want to recreate an experience that is user friendly. With qualitative data we can also get inputs on what to improve and not only measuring how the current system performs.

The algorithms for making the objects float and move in the water surface works, but could be improved. We attempted to create our own type of gravity which works in our case, but a possible improvement could be to look at physics plugins to threeJS in order to create a more realistic behaviour for each object. But then again as a learning project, we would prioritize low level coding over the result. The physics could however be more true to physical laws. The gravity and buoyancy should have been implemented as methods valid for all objects in the scene and dependant on parameters for said objects, such as geometry and density. Going further, surface tension should also be accounted. Creating floating effects for other, more irregular, shapes than spheres would be a good next step. Another improvement could be to include wind in the scene which certainly will make the movements of the balls more realistic. Given more time writing a shader from scratch would be good challenge. Even if the result would be worse, it would be a great learning experience.

Another thing that would be interesting to investigate is if other wave forms would enhance the user experience. This would probably extend the scope of the project to also include higher waves. Maybe a combination of different wave shapes would be most realistic. An perceptual study on this would be really interesting.

Lessons learnt

During the project we have learnt a lot of how to build up an interactive scene. And for us with previous knowledge and experience from working with web technologies the first steps was easy.

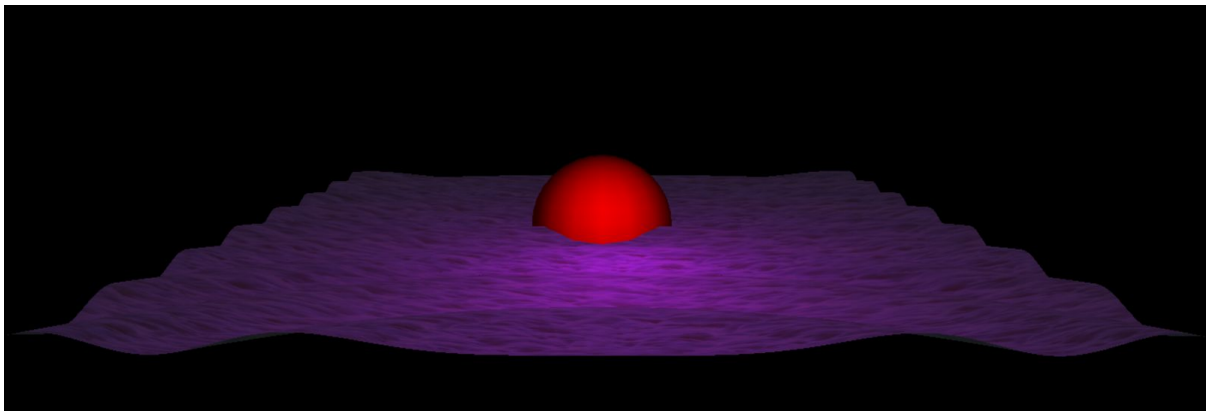
- Shaders can be complicated, at least when dealing with so many characteristics.
- Physics “in game” does not have to correspond to real physic equations in order to create a fully natural “look and feel” of the scene.
- Frame rate, optimization.
- The characteristics of water is very complex which have made us appreciate water in our surroundings a lot more than before.

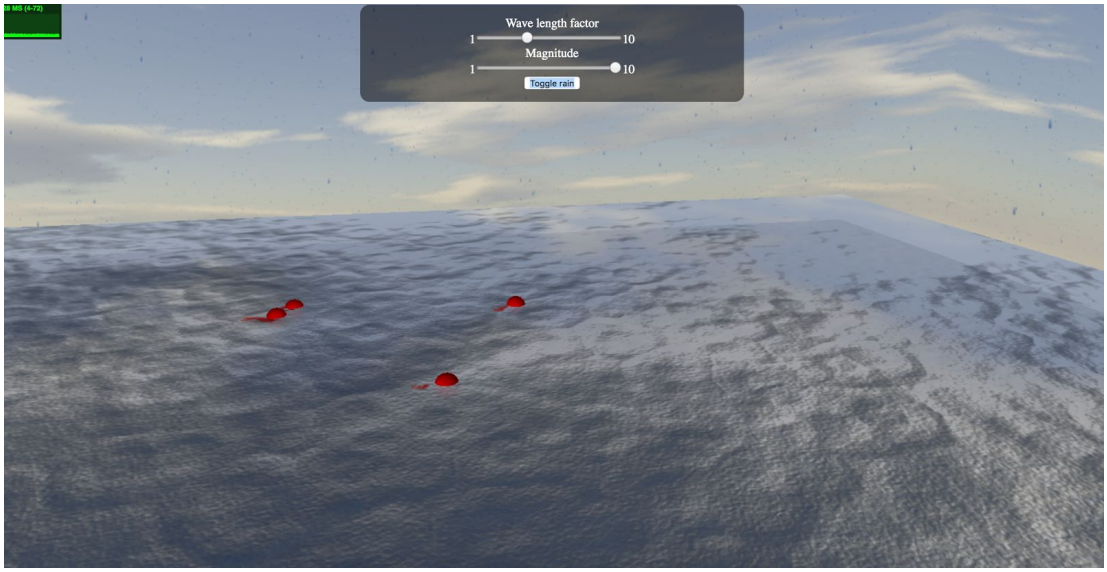
Contributions of different team members

The work load has been evenly distributed between the group members and everyone has had insights into every part of the project. However we have been responsible for different areas.

Water shader - Calle, (Emil)
Ball physics - Kevin, Calle
Rain particle system - Emil, Kevin
Wave generation - Kevin, Emil

Project in images




Wave length factor
1
10
Magnitude
1
10
Toggle rain

References

GitHub. (2016). jbouny/ocean. [online] Available at: https://github.com/jbouny/ocean [Accessed 3 Jun. 2016].

Lewis, P. (2016). Aerotwist - An Introduction to Shaders - Part 1. [online] Aerotwist.com. Available at: https://aerotwist.com/tutorials/an-introduction-to-shaders-part-1/ [Accessed 3 Jun. 2016].

Mathworld.wolfram.com. (2016a). Spherical Cap -- from Wolfram MathWorld. [online] Available at: http://mathworld.wolfram.com/SphericalCap.html [Accessed 3 Jun. 2016].

Mathworld.wolfram.com. (2016b). Nyquist Frequency -- from Wolfram MathWorld. [online] Available at: http://mathworld.wolfram.com/NyquistFrequency.html [Accessed 3 Jun. 2016].

Tessendorf, J. (2001). Simulating Ocean Water. Environment, 2, 1–19. http://graphics.ucsd.edu/courses/rendering/2005/jdewall/tessendorf.pdf

Wikipedia. (2016a). Dispersion (water waves). [online] Available at: https://en.wikipedia.org/wiki/Dispersion_(water_waves) [Accessed 3 Jun. 2016].

Wikipedia. (2016b). P-wave. [online] Available at: https://en.wikipedia.org/wiki/P-wave [Accessed 3 Jun. 2016].

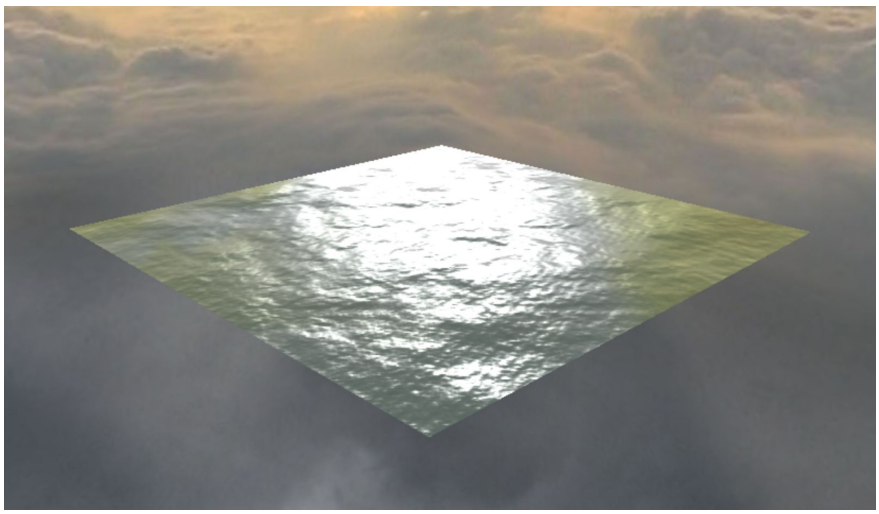
Wikipedia. (2016c). Trochoidal wave. [online] Available at: https://en.wikipedia.org/wiki/Trochoidal_wave [Accessed 3 Jun. 2016].

Project Specification DH232

Kevin Brundell Whittaker, kevinbw@kth.se
Carl Sténson, cstenson@kth.se
Emil Westin, emiwes@kth.se

Idea

We want to create an interactive water surface where a user can play around with some object(s) floating in it. We also want to explore the possibilities to add additional features such as rain and/or fog. The focus of the project is, at first, to implement physics that seems realistic toward the user. This would include features such as wave generation, as well as lighting and shading effects. If we have time we want to try and implement a splash effect which would occur when objects are dropped into the surface, and also look at implement rain and/or fog together with wind. We have the most developing experience within web technologies and we believe that there is a strong trend of computer applications moving online. Because of this we have decided on using webGL via the library threeJS.



A plane of water created with threeJS. (source)

Scope

Here the scope of the project is presented as well as potential extensions which might be implemented if there is time to do so.

Basis

- Water surface created with three.js.
- Interaction with objects floating in the water, creating waves and ripples.
- Implement light reflection and refraction, from a single light source.

Addon 1

- Implement splash effect.

Addon 2

- Rain and/or Fog effects and possibly wind.
- Possibly thunderstorm.

Kevin Brundell Whittaker, kevinbw@kth.se
Carl Sténson, cstenson@kth.se
Emil Westin, emiwes@kth.se

Addon 3

- Implement stereoscopy with a VR-headset.

Project Execution

To start, we need to read up on the threeJS documentation and explore the webGL library. Since none of us have any previous experience with graphic programming, other than the labs, it is hard for us to estimate a time frame for the project, as well as the difficulty level of the features we want to implement. We will use Github to ease the coding process during the project.

Our goal is to be done by the 31th of May, with work starting the 17th of May

Since we are three people in the group, the work on the code needs to be divided properly between the group members. At this point we have not located suitable areas to distribute to each group member.

Grade aspiration

We are aiming for E-D, but it would of course be interesting to know how much more it would take to get a higher grade. What would our proposed add-ons do to raise the grade? Do you have any other ideas for add-ons that you think is more suitable for the course?

Desired feedback

- Is the project scope suitable for 3 persons?
- How low level should the coding be? Ultimately we would like get a nice, immersive result, thus we would perhaps prioritize result over low level coding (if that makes any sense).