Augmented Reality Water Effects Visualization

by

EGE DUMANLI

Supervisor(s): MARK WINTER, BEREND BAAS

A Dissertation
Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

June 20, 2022

Abstract

Simulating visually compelling water is difficult especially in Augmented Reality environments where the water has to interact with the user's surroundings. In this paper, implementations of reflections, refractions and transparency effects that are physically inaccurate but result in visually compelling water simulation in Augmented Reality environments are presented and tested. There are many works on simulating water efficiently; however, they focus on doing this in a scene that is fully virtual. The contribution of this paper is to extend some of these effects so that they also work in Augmented Reality environments which are non-virtual.

The effects are tested by measuring the frames per second. In the end, the methods described in the paper make the water look aesthetically pleasing at around 26 frames per second. It is important to mention that the application is locked at 30 frames per second. This means the water shader is improved with the visual effects for the cost of 4 frames. The frames were measured on an OPPO mobile phone with the model CPH2195.

1 Introduction

Simulating fluids is a fascinating and well-studied subject that started at around 1996 [12] and is still being studied and improved upon today. There are several algorithms for simulating the motion of fluids and the effect of light interacting with such fluids especially used in media such as video-games and movies. However, generally in these cases, the environment is fully virtual. Meaning that the scene with the fluid simulation and the objects inside that scene can be modified such that it works realistically with effects such as refraction and reflection caused by the fluid. This research focuses on Augmented Reality(AR) technology where the environment will be in the control of the user navigating the camera. The research aims to find a performant yet visually compelling way to simulate water and its interaction with light on a mobile platform using AR where the environment is non-virtual.

Water rendering and simulation in interactive applications is a difficult balance between physical fidelity and efficiency. This can be even more challenging in the constrained hardware of AR mobile applications and the non-virtual environments. Consequently, more research in this area is needed. The effects described in this paper can be applied to any AR/VR application that wants to simulate visually compelling water.

The team behind "Sailing+" focuses on implementing an AR application where sailing races can be projected onto a surface by using AR technology. This application is being developed using the Unity Game Engine and its Universal Render Pipeline.

The basic approach taken in this paper is to extend the water effect implementations such as reflection and refraction such that they also work in AR environments. This is important because AR technology is becoming more widely available every day [10] and visually pleasing water simulation should be accessible to all platforms.

The questions that were explored and answered to arrive at optimal water visualizations are as follows:

- What types of water effect implementations are efficient and effective for AR environments? As water simulations involve intensive calculations, this will have to be explored in regards to mobile hardware.
- What water effects can be used to enhance the AR experience? The effects could be divided into two categories: the immediately noticeable and the not so immediately noticeable. This could be used to prioritize implementations.
- How will the refraction and reflection effects on fluids be implemented to work with AR environments? The reflection and refraction will have to be based on what the mobile device's camera sees. This means that there will be objects in the scene that are not in our control.

The next section mentions the works that are related to this research. Afterwards, Section 3 explains how the water effects were chosen and details the methods behind their implementations. Then, in Section 4, the results of the research are shown. Section 5 discusses these results and how they relate to the research questions. Finally Section 6 provides a summary of the paper, and mention several points for further research. The appendix A mentions some aspects of responsible research concerned with this paper.

2 Related Work

As briefly mentioned in Section 1, fluid simulations is a well explored topic within the domain of computer graphics. This research paper gathers ideas from different works, combines and extends upon them. This section will describe these works for the effects that they were used in.

Water flow and still movement
The idea for using texture distortion as the basic movement for the water simulation was described by Alex Vlachos in the SIGGRAPH2010 presentation Water Flow in Portal 2 [4]. This technique is implemented in this paper as described in this presentation.

Reflections
The main work used for ideas on reflection was the Boat Attack Demo [14] made by Unity Technologies themselves. This demo shows reflection probes and planar reflections as two methods of implementing reflections.

The planar reflections implementation was extended in this paper by combining it with the texture received from the user's mobile camera to reflect the user's environment. This is further explained in Section 3.

Refractions
The idea for using the screen texture and jittering it was described by Jasper Flick in his Flow Tutorial [8] series. In this paper, Flick's idea was built upon by using the screen space UV coordinates of the water plane. This made it so that only the parts of the texture received from the user's mobile camera that were occluded by the water were subjected to jittering.

Caustics
A way to fake caustics is described in the book GPU GEMS [2]. It involves tracing a minuscule amount of rays assuming that the sun is shining from directly above. The extension to this method is the use of incorporating the alpha cutoff. Since there is no ground material to put the caustics shader on, a plane is created on the ground level. Then alpha cutoff is used to only display parts that are bright.

3 Methodology

In this section, the methods of finding the right effects and their details will be discussed.

The first step in approaching the questions was to decide on what makes a water visually compelling. In order to arrive at answers, several images of rivers, seas and oceans were observed.

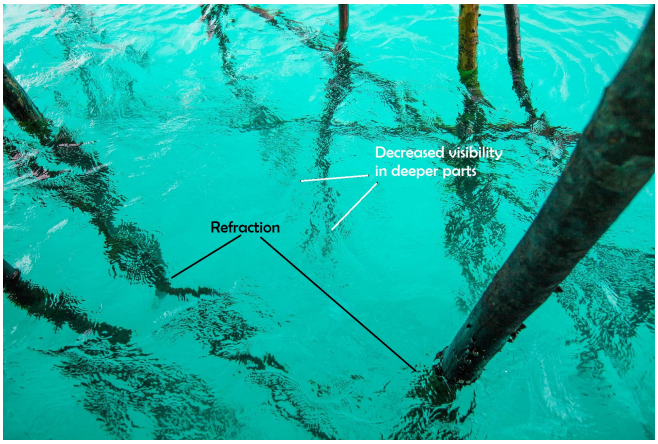

Figure 1: The refraction of the light entering the water distorts the view of the poles underwater. Also, the deeper parts of the poles are less visible because of the absorbed light. Courtesy of Kanenori [6].

Figure 2: The reflection of the boat is visible on the water. The reflected boat appears jittery. Courtesy of Marvin R. Molin [9].

Figure 1 does a good job at showing the effect of refraction and transparency while Figure 2 does a good job at showing the effect of reflection.

Reflection, refraction and transparency were decided to be the most crucial ones as they were the most present and apparent in pictures. Caustics and foam effects were decided to be treated as additional details which would be looked at if time permitted. As many mobile phones, including the one used in this paper do not support the Depth API, the caustic effect was given more importance compared to the depth based transparency.

For the above-mentioned visual effects to work, there needs to be a water shader that they could be added on top of. This shader should consist of the two basic motions of fluids: directional flow and still movement. In order to write a shader that could do these motions, a method called texture distortion [4] was followed.

3.1 Movement Water Shader

In this subsection, the details behind the texture distortion implementation are discussed.

The distortion of the normals used in this approach is essential for the water effects to look visually compelling. This is because the normals of the water surface are used in all of the visual effects to position them correctly along the water plane and to give them the jittery effect caused by the movement of the water.

Texture distortion method can be cleanly separated into two parts, the directional flow and the still movement.

3.1.1 Directional Flow

The directional flow part makes use of a flow map which is a 2D texture that describes how the UV coordinates should be manipulated. An example of a flow map can be seen in Figure 3

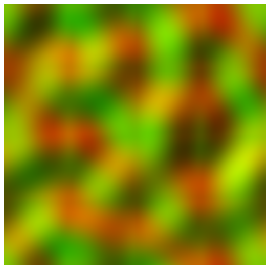

Figure 3: A flow map with the R(red) channel representing the U component and the G(green) channel representing the V component of the UV. Courtesy of Jasper Flick [8].

When a flow map is used in combination with a dynamic value such as time, it can be used to distort the UV coordinates of the water plane, creating the effect of flow. The resulting texture is shown in Figure 4

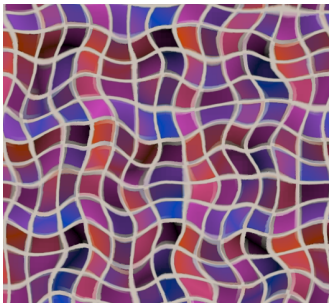

Figure 4: A UV texture being distorted in both directions via the flow map.

3.1.2 Still Movement

The still movement part makes use of a UV jump. In order to simulate still movement, an identical second texture can be sampled. Then, its UV's are offset from the first textures UV. Finally they are blended together. A UV jump is when the offset between two UVs jump to a different value in order to avoid the offset looking directional. Figure 5 shows this effect nicely where two different square tiles can be seen being blended.

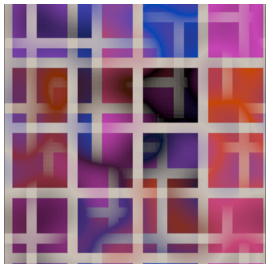

Figure 5: The blending between two sampled textures. Two different sets of square tiles can be seen changing colors.

3.2 Water Effects Shader

With the water movement shader handled, this section discusses the implementation of the reflection, refraction and caustics effects. Every effect will be discussed in two parts: the base implementation, and the AR specific extensions. All implementations in this section assume that there exists no depth information on the AR environment.

Before the effects are explored further, it is important to mention that an accurate implementation of all of the effects in this section would require the usage of ray-tracing. Since ray-casting is not performant on mobile hardware [5] and since ray-tracing is not commonly supported on mobile hardware, a method that results in a convincing effect should be utilized instead of a physically correct one.

3.2.1 Reflection

To understand this subsection, the word reflection, as it is used in this paper, should first be defined. Reflection is when the light ray hits a surface and changes direction while not changing medium. This is shown in the Figure 6.

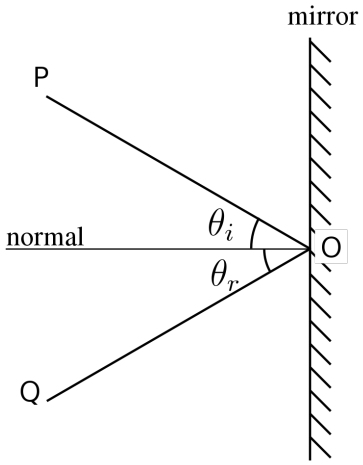

Figure 6: A light ray is sent from point P to point O. The reflected ray is the line from point O to point Q. The angles represent the angle between the ray and the normal of the surface. Courtesy of Johan Arvelius [13].

To fake real-time reflections, there are few methods. One such method is planar reflections. Since the surface of the water can be approximated as a plane, and since this method works with dynamic environments, it was chosen over the others. This approximation is acceptable in this paper because physical accuracy of the water visualisation is not the main concern. Some other methods result in more realistic reflections however, they are either too expensive to run or not applicable to a dynamic environment. The AR specific extensions section of the reflection effect will mention some of these and

explain why they were not chosen.

Base Implementation

In order to understand how planar reflections work, it is useful to think in terms of mirrors. When an observer looks in the mirror, they see their image as if they were looking through the other side of the mirror. This is visually shown in Figure 7.

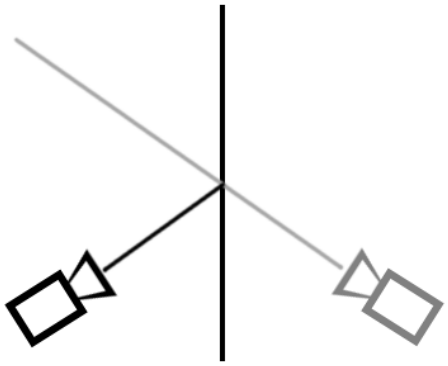

Figure 7: The image on the mirror can be seen as if the camera(black) was reflected to the other side of it(gray).

In the same way displayed in Figure 7, the camera could be reflected around the water plane in order to simulate convincing reflections. To reflect a point about a plane that does not go through the origin, an affine transformation matrix M [15] seen in equation 1 is used.

$$Plane = ax + by + cz + d = 0,$$

$$\mathrm{M} = \begin{bmatrix} 1-2a^2 & -2ab & -2ac & -2ad \\ -2ab & 1-2b^2 & -2bc & -2bd \\ -2ac & -2bc & 1-2c^2 & -2cd \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

But this introduces a problem. There may be an object on the opposite side which is blocking the view of the reflected camera. This problem is visualised in Figure 8.

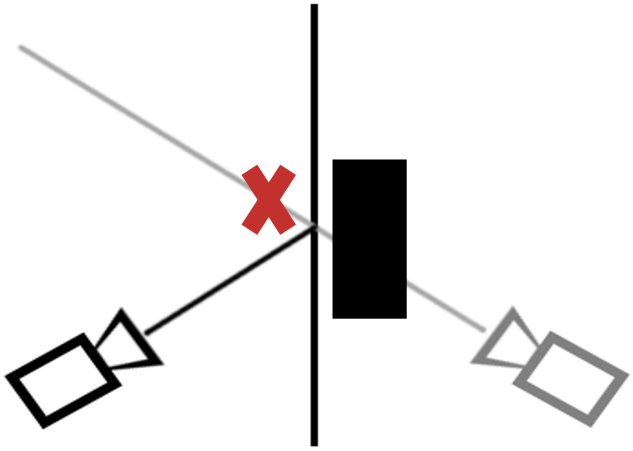

Figure 8: The reflected camera(grey) is being blocked by the black rectangle.

This problem can be tackled in two ways. Firstly, a custom shader could be written which disposes of the pixels that lay under the water plane. This is an efficient way of doing it. However, there is even a more efficient method called the oblique view frustum clipping which makes use of the fact that the clipping stage is always going to take place [1]. Oblique view frustum clipping describes the necessary transformations in order to make the near clipping plane any arbitrary plane. This means that by using the water plane as the near clipping plane, anything under can be clipped for free, without the use of any shaders. This method is visualized in Figure 9. The transformation matrix required is given in equation 2. In this equation, M is the initial projection matrix. M_i represents the i-th row of the matrix M. C and Q can be seen in Figure 9.

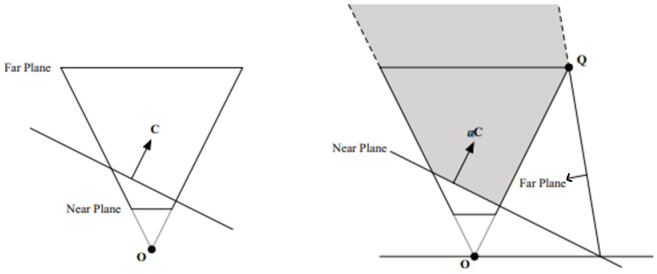

Figure 9: Before the transformation of the near plane(Left) and after the transformation of the near plane(Right). C is an arbitrary plane and O is the origin. The gray part is the part which is not clipped. Courtesy of Eric Lengyel [1].

$$\begin{bmatrix} M_1 \\ M_2 \\ \frac{2M_4 \cdot Q}{C \cdot Q} C - M_4 \\ M_4 \end{bmatrix} \quad (2)$$

Implementing the planar reflections result in a convincing reflection effect that can be seen in Figure 10.

Now that the virtual world can be reflected on the water plane, reflections in the real world need to be handled.

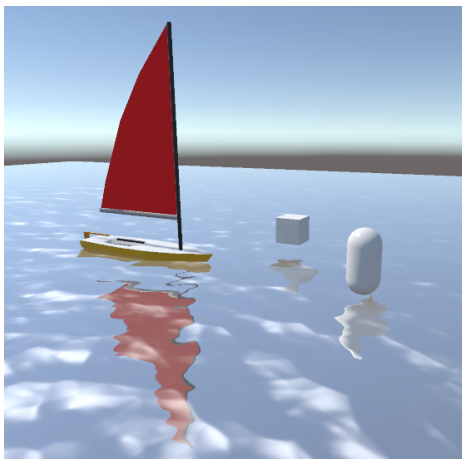

Figure 10: The water shader with planar reflections. The boat, cube and the capsule are all being reflected.

AR Specific Extensions

The base implementation described above works for objects that were defined in the scene. But for AR environments, the user's environment needs to be reflected. Since the assumption is that there is no depth information, simulating an accurate reflection of the real world environment is not feasible. Considering this constraint, two different solutions that would result in inaccurate yet visually compelling reflections were arrived at.

The first solution takes place in the fragment shader. The output of the user's mobile camera is retrieved, lets call this the background texture. The background texture is treated as if it lies in the scene an arbitrary distance away from the water plane. Let's call this imaginary plane, the background plane. Then, for every visible pixel of the water plane, a line is drawn from the scene camera to that pixel and the reflected line is calculated. After that, if reflected the line intersects with the background plane, the color value of the intersection point it used. If it does not intersect, black color is used.

The second solution makes use of the previously explained planar reflections implementation. Instead of the first solution where the background texture is treated as an imaginary plane in the world space, it can be made into an actual plane mesh and the background texture can be drawn on it. This plane is placed an arbitrary distance away from the water plane. Now, since the plane is in world space, the general planar reflection code will also reflect it. But this plane is currently sitting in one position instead of moving with the camera. In essence, the plane should always face the camera as it is used to reflect what the user can see. A basic implementation would involve applying any transformation on the camera also to the plane. This is the parenting approach. There are several problems with this approach. One of them is that the background plane can now coincide with the water plane if the user moves away from the water plane because each position change of the camera is applied to the background plane. Another problem is that if the position of the camera along the y axis changes, then so does the background plane's. This results in the reflections constantly moving up and down with the camera which is an undesired effect.

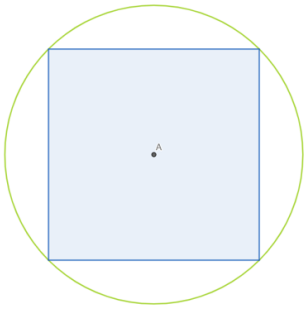

Figure 11: From above, the water plane (blue square) and the circle representing the possible positions the background plane can move to. The plane moves tangent to the circle. The radius of the circle should be bigger or equal to half the length of the water plane's diagonal.

A better approach would be to allow the background plane to move in a circle around the water plane with the radius set as the arbitrary distance mentioned before. This circle is visualized in Figure 11. The background plane is then moved on this circle by only applying the camera's rotations on the y axis to the circle. This way, the background plane is always the same distance away from the water plane and at the same height but still faces the camera. This plane is shown together with the camera and the ground plane in Figure 12. The only thing left to do is to configure the main camera so that it does not render this background plane that was created.

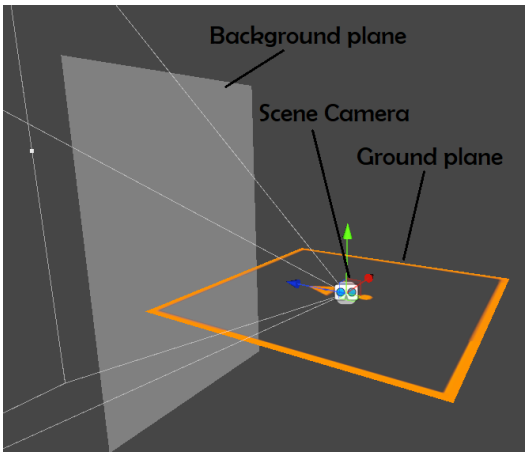

Figure 12: The background plane always faces the scene camera. It can only move along the circle that is shown in Figure 11.

The second solution was chosen over the first because it works out of the box with the planar reflection implementation. A screenshot showing this effect on a mobile phone can be seen in Figure 13.

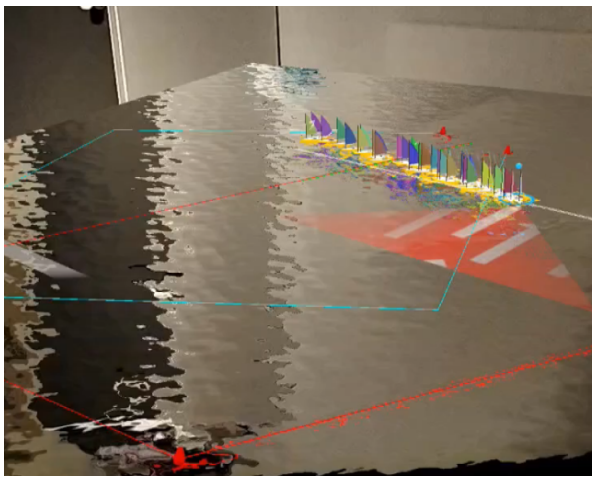

Figure 13: The reflections working in an AR environment without any water texture applied. The boats reflections are seen right below them. The cabinet on the left of the screen is being reflected as well.

As mentioned before, there are other reflection implementations such as reflection probes and screen space reflections but both are not suitable for AR. Screen space reflections is an expensive dynamic post processing effect which Unity itself states "it is not recommended to be used on mobile." [7] Reflection probes rely on information from the environment and store it in a cubemap; however, this also is not feasible when the output of the user's camera is the environment because it is not a static scene.

3.2.2 Refraction

Before explaining this effect further, the word refraction, as it is used in this paper, should be defined. Refraction is the light changing direction as it enters from one medium to another. This is visualized in Figure 14.

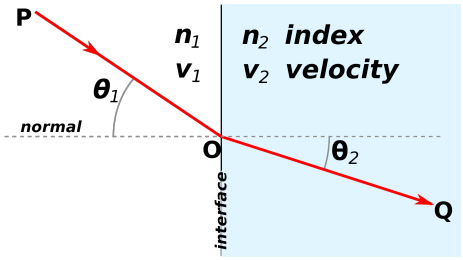

Figure 14: The light ray is shot from point P towards point O. The refracted ray is the line from point O to point Q. The angles represent the angles of the rays to the normal of the surface. Courtesy of Cristan and Sawims [11].

Base Implementation

The constant movement and shifting of the water plane's normals make faking the refraction effect easier. This is because the water surface will be jittery and distorted.

Each frame, the rendered scene is sampled by using the screen-space UV coordinates of the water plane. This effectively renders the rendered scene onto the water plane. For AR, this rendered scene will instead be the texture retrieved from the user's mobile camera.

AR Specific Extensions

Lets call the texture retrieved from the user's mobile camera the background texture. The refraction should only affect the part of the background texture occluded by the water plane. In order to do this, the screen space UV coordinates of the water plane need to be calculated. The screen space coordinate is a four dimensional vector with variables x, y, z and w. Since the UV coordinates are needed which is two dimensional the x and y variables will be used. One last thing to do before moving on is to divide the x and y by the w. This is called homogeneous divide and it is used to get from homogeneous coordinates to 3 dimensional space [3]. If the world space normals are then added to the screen space UV's, the refractions become jittered with the water plane's normals. This means that the incident ray, the line PO in Figure 14, is completely disregarded and the refracted ray, OQ in Figure 14, is approximated by the normal of the water surface at point O. The world space normals can be multiplied by a value before the addition if the intensity of the refraction needs to be controlled. By using these new UV coordinates to sample the background texture, only the parts that are occluded by the water plane will be rendered and jittered. A screenshot of this effect can be seen in Figure 15.

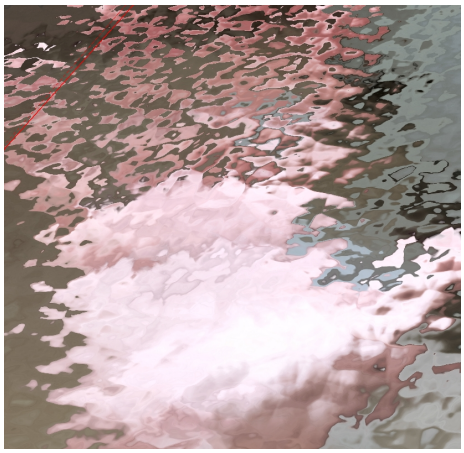

Figure 15: My hand under the water plane distorted because of the refraction effect.

3.2.3 Caustics

As the light gets refracted, some points on the ground underwater get hit by more rays resulting in brighter sections.

Base Implementation
To fake this effect, a section from the book GPU GEMS describes an ingenious approach. The approach assumes that the sunlight is coming from directly above the water plane. Therefore, the map shown in Figure 16 can be used as a representation of how the sun would look like when it is directly above.

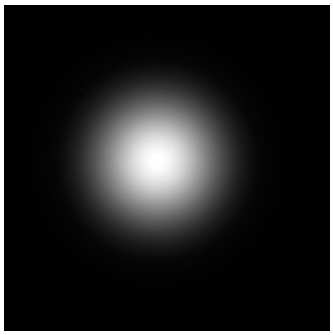

Figure 16: The texture used to simulate sun that is directly above.

Then, to determine where a ray, shot from the ground plane towards to water surface, lands on the sun map, a calculation is computed. This calculation includes calculating the line plane intersection to find the point where the ray, shot from the ground plane, lays on the water plane, and then uses the normal at that point to find the final ray.

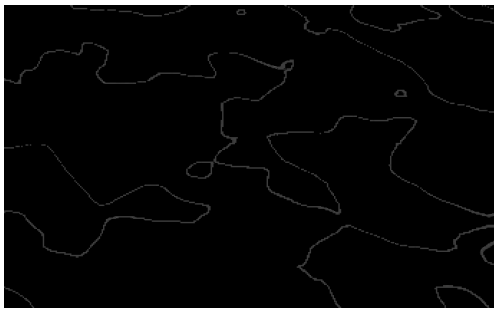

Figure 17: The caustic texture. Fully opaque.

AR Specific Extensions
The method described in the base implementation results in a texture where the caustic lines are white and the rest are black. This is shown in Figure 17. If the ground plane had a texture, this could be added onto the final color. However, since the ground in the AR environment is not an object in the scene, a plane on the same level as the ground needs to be created. Then, the texture is applied to this planes material. However, the resulting plane is not transparent like caustics are supposed to be. To fix this, the red, green and blue values of the pixel can be added in the fragment shader. If the sum is equal to 0, this means that the pixel is a black pixel. Then, the alpha value of the black pixel is set to 0 in order to make it fully transparent. A trick that can be used here to avoid conditional statements in the shader code is that the red, green and blue values can be summed up and clamped between 0 and 1. Consequently, the clamped value can be used directly as the alpha value. The resulting texture can be seen in Figure 18.

Figure 18: The caustic texture with correct alpha values. The grey background is the sky box, which means that the black parts seen in Figure 17 are now transparent.

4 Results

This section will show how the final water shader looks like and compare the frames per second(fps) before and after the implementations detailed in this paper. Since fps will vary from system to system, the exact machine used in the testing will also be mentioned.

First of all, lets see the final result which combines reflections, refractions, caustics and the movement shader by looking at Figure 19.

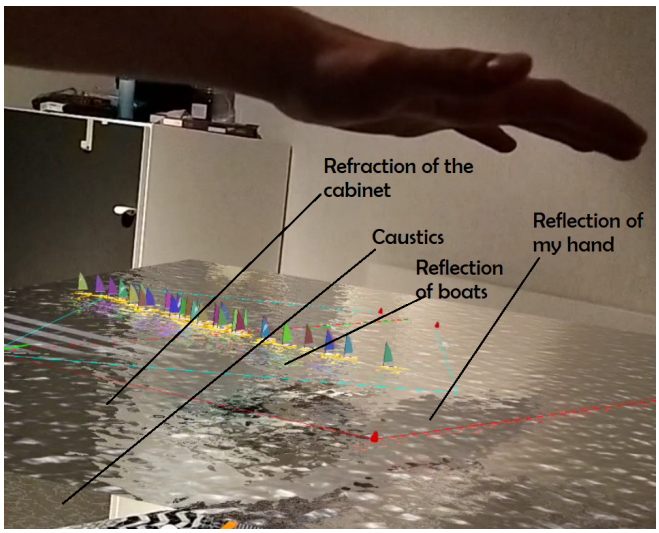

Figure 19: Reflection, refraction and caustics shown all together.

These images and the frame tests were handled on an OPPO A54 5G mobile phone with the model CPH2195 running ColorOS version V11.1 with Android version 11. It should be mentioned that AR applications are capped to 30 fps because of the camera's capture rate. The initial water shader that existed before my work is visualized in Figure 20. The application with this water shader ran at 30 fps without any frame drops. After the implementation of the shader detailed in this paper, the application dropped to an average of 26 fps. With this shader, the lowest fps value was 24 and the highest was 30. The lowest value was observed when moving away from the water plane which causes the camera to render everything. The highest was observed when the camera was close to the water plane. Table 1 displays the minimum, maximum and the average frames per second for each effect individually.

Effects	Min fps	Max fps	Average fps
Reflection	24	30	27
Refraction	26	30	29
Caustics	28	30	30
Texture Distortion	27	30	29

Table 1: Fps for individual effects.

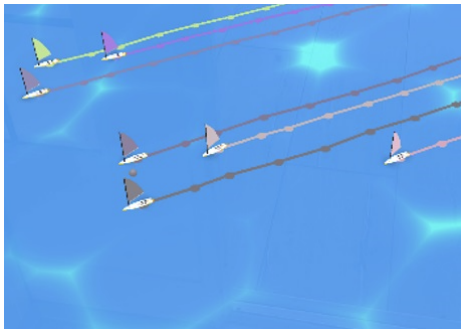

Figure 20: The water shader before this paper.

5 Discussion

The water shader detailed in this paper is specific in terms of the environment it can be used in. A lot of previous work on water shaders focus on different environments other than AR/VR. This paper can be seen as small steps into the territory of water shaders in these environments.

The most apparent limitation is the lack of use of depth information. Many of the effects can be improved to look better if depth textures were used. For example, the angle of refraction could be estimated by the depth texture which would result in different amount of distortion for objects laying in different distances. Or the reflections could be made so that only the parts with depth value higher than the water plane are reflected. These implementations would not work as simple extensions to the current implementation and would most likely require a rewrite of them.

One limitation concerns the reflection effect. As the user starts looking down at the water plane, the reflected background displays parts that are below the water and should not be reflected. This is because the effect is faked without the use of depth information.

Another limitation with the reflection effect is that the reflection of objects appear right below them. For example, in Figure 19, the reflection of the hand should have been further away from the camera.

A limitation with the refraction effect is that every object is distorted the same amount regardless of the angle they are looked at. Since in this paper aesthetic beauty was prioritized over physical fidelity, this was expected. Moreover, the constant movement of the water surface hides this well.

Finally another limitation with the reflection effect is that as two cameras are being used, one main and one for reflections, the scene is rendered twice. This introduces some overhead. However, it can be reduced by rendering the reflections in a lower resolution. I chose to render the reflections in quarter of the main cameras render resolution. It is acceptable to do this since the reflection is distorted by the water plane's normals and the quality change won't be as noticeable as it would be on a smooth, clear plane.

6 Conclusion and Future Research

The effects caused by reflection, refraction and caustics were implemented as they were deemed to be the most noticeable. As seen in Section 4, the new water shader looks visually compelling and interactive while the old shader looks basic and bland.

Every effect in this paper is implemented without depth information. This is because the water should look visually compelling even on mobile phones that do not support this feature. The reflection, refraction and caustics could be improved if depth information was used for them.

There are several ideas that are worth exploring if a further research was to be held. These will now be mentioned with no order of importance.

To improve the reflections, the user could be asked to take pictures of all 6 directions of their environment or to scan their room with photogrammetry. This could be mapped to a cubemap texture and used in combination with reflection probes to calculate reflections which would look more accurate than the current implementation. However, this approach would not reflect anything dynamic such as the user's hand. This was not implemented because it includes a new step in the setup part of the application for a single visual effect and it does not allow for dynamic interaction.

To improve the caustics, rays could be traced backwards from the ground towards the light source. This would reduce the amount of rays to trace because it is known that these rays are the ones that hit the ground.

Maybe as mobile GPUs continue to evolve, an approach using ray-tracing would become feasible, allowing us to enter the domain of hyper realistic mobile simulations.

A Responsible Research

Every method shown in this paper is reproducible by anybody with an access to a computer and a mobile phone supporting AR. Each effect mentioned in Section 3 was detailed with figures. Parts which were short can be understood better by following the resources given either in Section 2 or in the References. These parts are generally the base implementations that this paper has extended upon therefore they can be learned in their respective sources if the reader desires to get a deeper understanding.

References

[1] E. Lengyel, "Oblique view frustum depth projection and clipping", *J. Game Dev.*, vol. 1, pp. 1–16, 2005.

[2] J. Guardado and D. S. Crespo, "Rendering water caustics", in *GPU gems: Programming Techniques, Tips, and tricks for real-time graphics*, R. Fernando, Ed. Addison-Wesley, 2007, pp. 31–44.

[3] E. Lengyel, "Projection matrix tricks", GDC07, 2007. [Online]. Available: http://www.terathon.com/gdc07_lengyel.pdf.

[4] A. Vlachos, *Water flow in portal 2*, URL: https://cdn.cloudflare.steamstatic.com/apps/valve/2010/siggraph2010_vlachos_waterflow.pdf, Jul. 2010.

[5] W.-J. Lee, Y. Shin, J. Lee, *et al.*, "A novel mobile gpu architecture based on ray tracing", in *2013 IEEE International Conference on Consumer Electronics (ICCE)*, 2013, pp. 21–22. DOI: 10.1109/ICCE.2013.6486777.

[6] Kanenori, *Transparent turquoise water*, 2017. [Online]. Available: https://pixabay.com/photos/water-transparent-turquoise-2328294/.

[7] U. Technologies, *Screen space reflection - unity manual*, May 2017. [Online]. Available: https://docs.unity3d.com/es/2017.4/Manual/PostProcessing-ScreenSpaceReflection.html.

[8] J. Flick, *Unity flow tutorials*, Jun. 2018. [Online]. Available: https://catlikecoding.com/unity/tutorials/flow/.

[9] M. R. Molin, *Boat on the sea of sabang, calabanga, camarines sur*, 2018. [Online]. Available: https://commons.wikimedia.org/wiki/File:Boat_on_the_Sea.jpg.

[10] M. Boland, *Mobile ar users approach 600 million*, Nov. 2020. [Online]. Available: https://arinsider.co/2020/09/10/mobile-ar-users-approach-600-million/.

[11] W. Commons, *File:snells law.svg — wikimedia commons, the free media repository*, [Online; accessed 14-June-2022], 2021. [Online]. Available: %5Curl%7Bhttps://commons.wikimedia.org/w/index.php?title=File:Snells_law.svg&oldid=560690761%7D.

[12] Wikipedia contributors, *Fluid animation — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Fluid_animation&oldid=1058057758, [Online; accessed 27-May-2022], 2021.

[13] W. Commons, *File:reflection angles.svg — wikimedia commons, the free media repository*, [Online; accessed 14-June-2022], 2022. [Online]. Available: %5Curl%7Bhttps://commons.wikimedia.org/w/index.php?title=File:Reflection_angles.svg&oldid=636381280%7D.

[14] U. Technologies, *Boat attack: Demo project using the universal rp from unity3d*, https://github.com/Unity-Technologies/BoatAttack, 2022.

[15] Wikipedia contributors, *Transformation matrix — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Transformation_matrix&oldid=1089725131, [Online; accessed 27-May-2022], 2022.