Sailing+, Dynamic Water Animation and Interaction through Hybrid Wave Model

Luc Jonker
Supervisor(s): Mark Winter, Elmar Eisemann
EEMCS, Delft University of Technology, The Netherlands

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

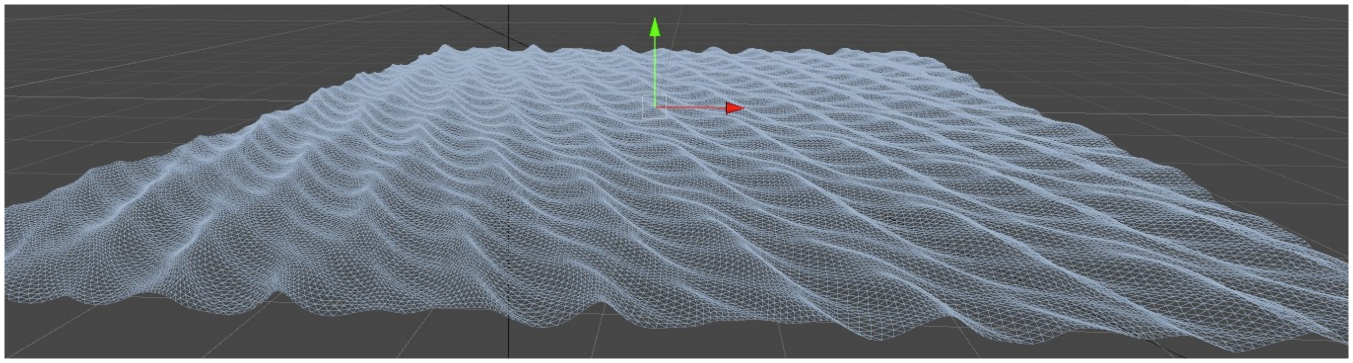


Figure 1: Geometric wave surface using Gerstner waves calculated in the GPU

Abstract

For mobile augmented or virtual reality applications with limited processing power, representing realistic water geometry is a challenge. Many existing solutions are simulations that can only run at interactive rates on desktop computers. This paper presents a lightweight approximative approach to water representation achieved through mesh LOD via clip-mapping, lightweight Gerstner wave calculations, and approximative texture-based effects for boat-geometry interaction. The foundational method is extensible to many applications modeling water geometry, or even landscapes, at a low performance cost.

1 Introduction

Water graphical representation often represents water as a mesh that gets modified using methods such as noise functions, displacement mapping, etc. While these methods can often lead to fast and relatively pleasing results, they lack a critical dynamic element of water in which it is affected by and reacts to geometry in the environment. Finding such a representation that can convincingly represent said dynamic element while also being able to run efficiently with limited processing power is considerably more difficult to do.

In general, water simulation techniques can be divided into two main subtypes: spectral methods represent ocean geography by superimposing waves of different frequencies based on oceanographic models while physically-based methods simulate water behavior with physical simulations relying on solving NSE [Navier-Stokes Equations] [1]. Spectral methods tend to produce realistic results through representing the ocean via a superposition of sines and cosines, while physically based models are somewhat more flexible and allow for object interaction. Spectral methods do not allow interactions with complex boundaries or obstacles, and physically based models can be more computationally expensive with visible detail seen in the waves being strongly correlated to the resolution of the underlying mesh [2].

For an application involving limited processing power (e.g. a mobile application), many of these models are inappropriate. They tend to emphasize simulation, where as a mobile application can get away with representing a convincing approximation of water for the sake of increased performance.

One such application is Sailing+, which is an augmented/virtual reality (AR/VR) app designed to experience sailing regattas. Users can project a virtual ocean into their living room, and use their phone to watch the sailing race from whichever perspective they like.

The task of this research is finding and/or creating a representation for the ocean water geometry for this application which is both aesthetically pleasing and more realistic. In addition, the application is designed for use with augmented / virtual reality hardware in mind, necessitating strict performance requirements.

More specifically, the research questions are:

1. What are methods for representing dynamic water geometry that support interaction with other meshes such as boat wakes, foam at shorelines, splashes, etc?
2. What techniques can be explored to optimize the representation to reach good real-time performance on mobile, AR/VR hardware?

Solving this problem could provide a basis for computer graphics developers who wish to implement aesthetically pleasing, realistic water effects at a low performance cost. Applications with speed as a priority could take and adapt this method to suit their own needs.

This paper presents a hybrid wave model inspired by ‘wavelet’ approaches [2]. The model combines spectral approaches for macro wave generation with water surface effects calculated via approximation techniques to give a convincing result at the micro scale.

The structure of this paper is as follows. Section 2 describes the methodology chosen to represent the water. Section 3 provides results both with performance metrics and commentary on the visual appeal. Section 4 discusses the results and methods chosen, giving some motivation for the choices made. In addition it describes some limitations of the chosen methods. Lastly, Section 5 provides a summary of the research, along with paths forward for extension of the work described in this paper.

2 Methodology

The goals of the geometric water representation place priority on two major factors, performance and visual appeal.

Methods were chosen with these priorities in mind, striking a balance between low performance impact with somewhat realistic and visually appealing results.

The water representation can be split into three distinct subsections which come together to form the overall water geometry technique. First is mesh generation, then static macro wave geometry representation, and lastly dynamic 'micro' water effect representation.

2.1 Mesh Generation

Mesh generation forms the basis of the geometric water representation. The underlying mesh must have a high enough resolution to represent high frequency wave components [2]. However with higher mesh resolution comes higher performance impact. One could simply make a large high resolution mesh, but the resulting performance overhead makes this solution undesirable given current phone hardware. Therefore, a Level of detail (LOD) approach was chosen to balance visual fidelity with performance.

The chosen mesh generation technique is a clip-map approach [3]. This method represents geometry as a series of expanding meshes at lowering levels of detail as the mesh expands away from the camera, as can be seen in Figure 2.

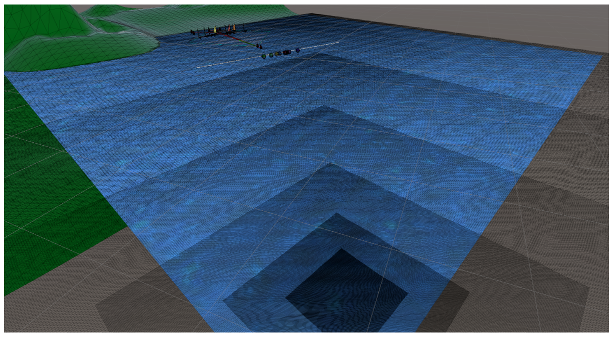


Figure 2: Clip-map mesh generation with vertex density lowering away from the center (darker area) where the camera is located. At the edge of the play area, the pixels are culled using the UV coordinates to maintain a clear boundary

The mesh follows the camera as it moves, maintaining a higher level of detail close to the viewer.

As the mesh is moved around, the UV coordinates of the vertices are dynamically offset to give the appearance of movement, as the water effects are calculated using the UV coordinates. This is done by using the clip-map's relative position to the play area as an offset to the UV coordinates. In that sense, the clip-map acts as a blanket that moves with the camera over the play area to maintain high mesh resolution near the user while maintaining the illusion of a 'static' surface.

Furthermore, the UV offset allows for automatic detection of when a section of the clip-map no longer overlaps with the play area, in which case it can be hidden. Figure 2 shows how the clip-map mesh looks, and demonstrates how the parts of the mesh not overlapping with the play area are culled.

If the clip-map moves too far away, such that it doesn't overlap the play area, a low resolution underlying mesh is revealed. This mesh won't be able to support the high resolution effects, but if the user is at such a distance then it isn't noticeable.

2.2 Static Macro Wave Geometry

The ocean in the real world features large swells that move through it over long distances. Representing this macro wave movement adds much more authenticity to the final result. For this application, emphasis on customizability and performant calculation was a priority.

To that end, macro wave geometry is represented via superimposed trochoidial Gerster waves [4]. Unlike a simple sine wave, Gerstner waves approximate water motion by having a given point move in a circular motion as opposed to just up and down, as shown in Figure 3.

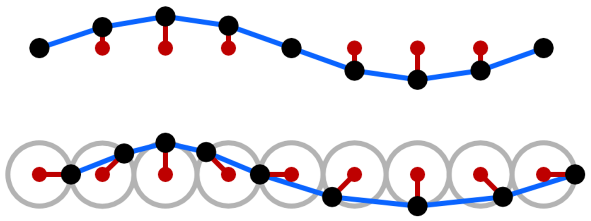


Figure 3: Motion of the vertex in a sine wave versus in a trochoidial wave, vertices bunch up in crests and spread out in troughs. Source: adapted from J. Flick. [5]

The equations to model such behavior should describe a sinusoidal movement for the point not only on the y axis, but on the x and z axes as well.

The Gerstner wave equations for this application are adapted from J. Tessendorf [6] and J. Flick [5]. To define a given wave to superimpose, one need only provide a four dimensional 'wave vector' $\mathbf{W}$ containing the direction, 'steepness' $(0 \leq \Omega \leq 1)$, and wavelength.

$$\mathbf{W} = (D_x, D_y, \Omega, \lambda)$$

Then the $\mathbf{x}_0 = (x, z)$ and y offsets are calculated by[1]:

$$\mathbf{x}'_0 = \mathbf{x}_0 + \mathbf{D} \cdot (A \cos (k \cdot (\mathbf{D} \bullet (l \cdot (\mathbf{uv})) - tc))) \quad (1)$$

$$y = A \sin (k \cdot (\mathbf{D} \bullet (l \cdot (\mathbf{uv})) - tc)) \quad (2)$$

Where $A = \Omega/k$ represents the amplitude, $k = 2\pi/\lambda$ is the wave number, $\mathbf{D} = (D_x, D_y)$ is the (normalized) 'direction vector' indicating wave direction, $\mathbf{uv}$ is the UV coordinate of the vertex, l is the total length of the play area, and $c = \sqrt{9.8/k}$ is the speed.

These waves are computed on the GPU per vertex, using the UV coordinates of the vertex as opposed to position for the wave calculation (hence scaling the UV value by the length of the field). The final wave surface can be customized by superimposing multiple Gerstner waves with different wave vectors $\mathbf{W}$. Combining these values in different ways leads to a considerable variety in the final visual result, with a simple example seen in Figure 1.

[1] $\bullet$ denotes dot product and $\cdot$ denotes scalar multiplication

An important feature of the water in Sailing+ is that the state of the ocean should be able to be found analytically with time. That is, one should be able to input the time, and have as output the state of the ocean at any given time-point in the simulation. This is because the user of the application has the option to move through the timeline of a race. For the macro waves, this is as easy as using the simulation time of a given race as the input time t used in the Gerstner wave equations.

2.3 Dynamic Micro Water Effects

Apart from the large scale motion of the ocean, there is also naturally water interaction occurring at a small scale as objects such as boats move across the surface. Small wakes will be produced as boats move across the surface. Attempting a full simulation of the water surface is difficult to achieve in a performant manner on a mobile device. Thus, a technique to emulate the effects of a full simulation was chosen.

Real boat wakes form a pattern called the "kelvin wake", in which the superimposed waves form a distinct V shape behind the boat, as seen in Figure 4.

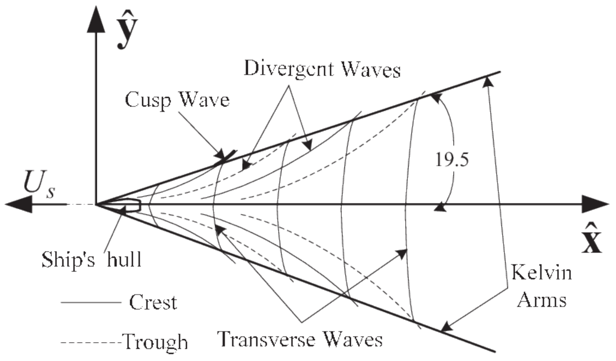


Figure 4: Distinct chevron formed from a boats 'kelvin wake'. Source: adapted from J. Wang et al. [7]

This wake effect can be approximated by taking advantage of information about the boats in the race, such as the direction and UV coordinates of the boat relative to the field. This information about a given competitors past n positions is used to superimpose rings which fade out and widen the older they become.

The calculations take place on a compute shader on the GPU, which allows for parallel computation. For each boat point P_n containing the direction and UV coordinates, the radius r of the circle is calculated for that point. Using the radius, a square of pixels of size $2r \times 2r$ centered on the boat position is determined. The strength of the j^{th} circle for the i^{th} pixel is given by:

$$\frac{lerp(1, 0, |\mathbf{P}_j^d \bullet \mathbf{l}_i|)a}{b|r^2 - (\mathbf{l}_i \bullet \mathbf{l}_i)|^2 + jc} \tag{3}$$

In which $\mathbf{l}_i = p_i^{uv} - \mathbf{P}_j^{uv}$ is the direction vector from the boat position to the i^{th} pixel calculated by the difference between the pixel position in UV coordinates and the boat UV coordinates. $\mathbf{P}_j^d$ is the direction of the j^{th} point, and $lerp(x, y, z)$ represents a linear interpolation between x and y based on z.

The coefficients a, b, and c change the behavior of the circles. Changing a impacts the strength of the circle overall, changing b impacts the strength of the blur around the circle, and changing c affects how much circles blur as they age.

The linear interpolation has the effect of dimming pixels on parts of the circle pointing in the same direction as the boat, while in the denominator the strength of the circle at the pixel is calculated and offset by the age of the point to achieve a fade over time effect.

By calculating these values for many rings and superimposing them, a wake-like effect is produced, an example can be seen in Figure 5. The resulting values are written to a texture to be sampled in the vertex and fragment shaders. In the vertex shader a displacement is applied to the geometry, and in the fragment shader the value is used to create a white foam effect.

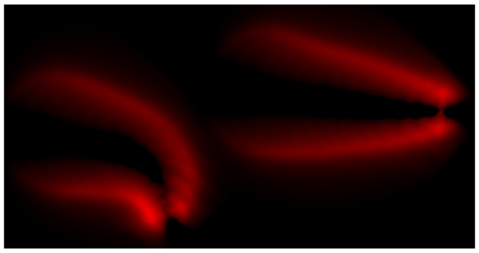


Figure 5: Approximated wake shape for a turning boat and a boat moving straight, formed by superimposing expanding circles at given time points, stored only in the red channel

Because the calculation only relies on the past n-positions of a given boat (to superimpose n rings), the state of the boat wakes at any given time can be found, and thus also conforms to users ability to scrub through the race timeline.

3 Results

The combined result creates a geometrically appealing ocean surface, with a reasonably small impact on performance. The performance tests were done on a Pixel 2 XL smartphone, and as such is representative of current or somewhat older phone hardware at the time of writing. An important note is that the application currently supports a maximum frame-rate of 60 frames per second (FPS), and the reported values are the mean between the maximum and minimum frame-rates.

3.1 Clip Map Analysis

The clip-mapping approach attempts to bridge the gap between mesh resolution and visual fidelity in the underlying ocean mesh. To that end, the implementation supports a large variety of resolutions at different sizes to support the needs of the various races, and updating hardware capabilities.

At maximum resolution, the clip-map mesh resolution is able to support small scale effects around dynamic elements such as boats, seen in Figure 6. In contrast, the maximum resolution of the naive solution (a single mesh at the maximum resolution supported by Unity) still left boats sitting on individual triangles, as shown in Figure 7.

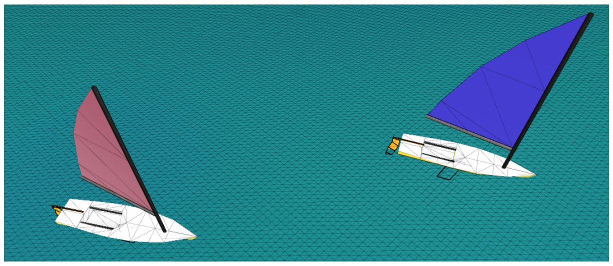


Figure 6: Maximum resolution of mesh using the clip-map, boats sit on a large amount of triangles which allows for small scale deformations

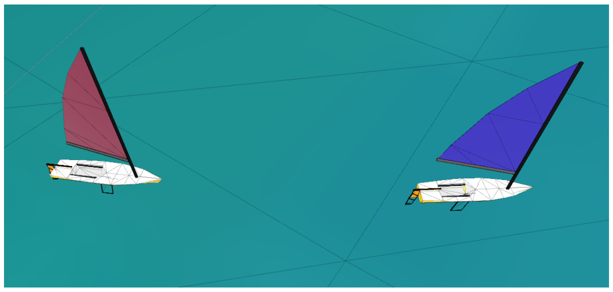


Figure 7: Mesh resolution of single mesh at highest resolution supported by Unity (65536 vertices) in which whole boats still sit on individual triangles

The performance tests were done with 2, 6, and 10 clip-map levels at a resolution of 50, 100, and 150 per clip-map level, without using any wave effects. The FPS values can be found in Table 1. Having just the basic Unity plane, as a control, resulted in a mean of 34 FPS.

Levels	Res: 50	Res: 100	Res: 150
2	34 FPS	27.5 FPS	20.5 FPS
6	26 FPS	21.5 FPS	16 FPS
10	19 FPS	15.5 FPS	14.5 FPS

Table 1: Mean FPS values of clip-map at different number of levels and different resolution per level

The triangle counts for the different resolutions can be found in Table 2. An important note is that the mesh was being fully rendered, no culling occurred.

Levels	Res: 50	Res: 100	Res: 150
2	142820	565620	1268420
6	390868	1541668	3452468
10	638916	2517716	5636516

Table 2: Number of triangles that make up the clip-map for a given resolution and number of levels

3.2 Macro Wave Analysis

The macro wave simulation makes the surface dynamic, and the customizability allows for a large variety in how the waves end up looking. Figure 8 shows multiple waves in the same direction, which could represent a hard wind. While Figure 9 displays a calmer, more uniform ocean surface being displaced.

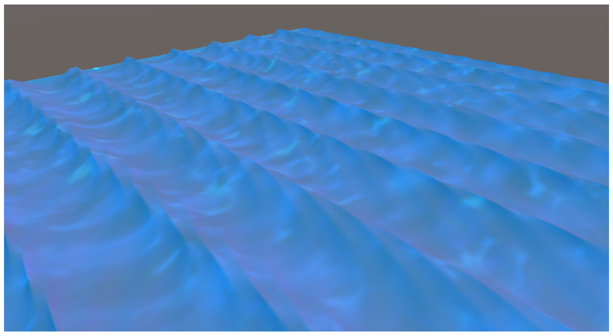


Figure 8: Waves aligned with higher amplitudes, representing a strong wind in a given direction

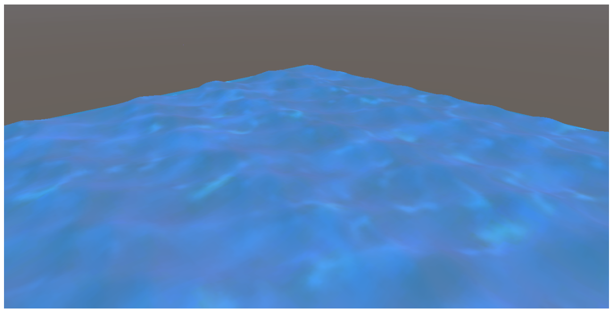


Figure 9: Calmer ocean surface, with swells and some smaller disturbances, representing a less intense wind

The performance impact of the macro wave generation is also remarkably small. Table 3 shows different numbers of superimposed waves. These tests were performed on a clipmap mesh with 6 levels and a resolution of 80. Having no waves, as a control, resulted in a mean FPS of 36.

Num Waves: 1	Num Waves: 2	Num Waves: 3
36 FPS	36 FPS	36 FPS

Table 3: Mean FPS values of different number of superimposed Gerstner waves

The superposition of more waves results in no noticeable performance impact given their parallel calculation in a vertex shader.

3.3 Micro Water Effect Analysis

The micro water effects result in a more dynamic ocean appearance shown in Figure 10.

The performance depends on two factors, the number n of historical boat points to consider, and the resolution of the output texture. Naturally it also depends on the number of competitors in a given race, but this is somewhat analogous

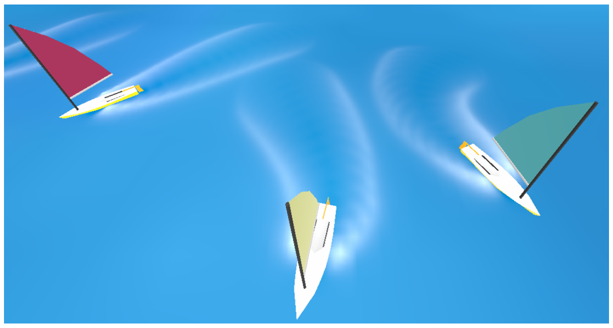


Figure 10: Wake left behind by competitors, in which geometry is displaced and a white foam color is added based on the texture values

to changing n. For these tests a race was used with 35 competitors. A performance comparison between n and texture resolution can be found in Table 4.

Num Points n	Res: 2048	Res: 4096	Res: 6144
2	28 FPS	13.5 FPS	7.5 FPS
6	28 FPS	12.5 FPS	7 FPS
10	25.5 FPS	12 FPS	7 FPS

Table 4: Mean FPS values of different number of historical points n output to different resolution textures. For a visual comparison see Appendix B

The performance, especially on mobile, depends much more heavily on texture resolution than the number of points, and thus optimizations that can lower the required texture resolution would have a higher impact.

3.4 Desktop Performance

For applications that target a desktop user, this method also performs remarkably well. For this test, a Desktop with a Ryzen 5 5600X CPU at 3.7 GHz, 16GB of RAM, and an RTX 3600 was used to test performance. The same race was used as the one on mobile. The clip-map was at a resolution of 150 with 8 levels. Three superimposed Gerstner waves were used. And lastly the dynamic effect texture had a resolution of 32768x32768, with 15 historic points.

With these settings, the application ran at an average of 215 FPS. Dropping the texture resolution to 8192x8192 with 6 clip-map levels results in an average of 288 FPS.

4 Discussion

The final result manages to be both visually appealing as well as largely performant. The heavy emphasis on mobile performance proved to be difficult, as many ’real time’ techniques only performed well on desktop computers.

The clip-map and Gerstner waves make the water feel considerably more dynamic and interesting, and have minimal performance costs on both desktop and mobile. The dynamic water effects are what produce the most overhead.

A ‘fully dynamic’ solution was initially implemented to represent the micro dynamic water effects, modeled as a wave height field in which subsequent values were computed iteratively based on previous values [8]. This method performed remarkably poorly on mobile hardware despite being a two decade old method, though this could be due to poor implementation.

The gap to fill was finding how a similarly convincing effect could be approximated while staying within the processing budget of a mobile device. An early implementation of the texture based water effects ran reasonably on desktop, with an average of 50-60 FPS on a 6k texture. On mobile however it could only manage 3FPS, with a 256 by 256 texture. The new ‘pixel square’ method resulted in an exponential increase on both desktop and mobile, but mobile still remains somewhat slower than hoped.

An important note about the FPS values reported. It may appear that it is relatively low overall (even 34 FPS for the default mesh). This is due to ‘events’ taking place during races. When an event occurs, effects such as a highlight and sphere are generated which cause a large FPS drop resulting in a lowering of the mean. For comparisons however, it is still an appropriate metric.

Furthermore, an interesting question may be, how was performance better during the Gerstner wave test at a resolution of 80 and 6 levels, than even the resolution 50 clip-map test. The likely answer is that the shader that performs the Gerstner wave calculations, also discards pixels at the edge of the play area, as seen in figure 2. During the clip-map tests, this wasn’t the case, and the entire mesh was rendered, which could result in a more noticeable performance drop.

4.1 Limitations

An important limitation to note with this technique is that it is assuming a bounded, preset “ocean area”. While clipmapping could support a larger mesh along with macro Gerstner waves, the small scale wave surface works on a precise “play area” and would need work to be extended to boundless, or procedurally generated surfaces. However this is certainly possible and could prove to be a remarkable application of the technique.

Clipmapping also assumes a flat ocean mesh. For most applications such a flat mesh is desirable, however in the even that one would want a water *volume* of sorts in which the ocean mesh bent (i.e. to form other 3D objects besides a plane such as a cube or sphere) work would need to be done to assure that the mesh resolution would be high enough at points where the mesh strayed away from the basic plane shape.

While Gerstner waves can support a large variety of convincing and appealing water surfaces, it as a periodic function does inevitably repeat in a periodic manner. This behavior can be mitigated and may even be desirable depending on the application.

Lastly, the dynamic water effects make some assumptions about boat behavior. Firstly it takes the previous boat position always a set time earlier for each point. This results in circle spacing which works well on some races, but when the boat moves more quickly, the effect is diminished. Solving this would involve using the boat velocity as a factor in determining the previous time point to use. Next, if the boat is sitting somewhat still or moving slowly, the circles stack in an unrealistic manner. If the number of historic points also dynamically changed based on boat velocity this could lead to more realistic results. Lastly the wakes are designed with the boats being at their smallest scale, however if the user moves away the boats scale up and occlude the wake. Solving this would involve scaling the wake texture simultaneously with the boat models.

5 Conclusions and Future Work

Representing a dynamic, visually pleasing geometric water surface with high performance on mobile hardware is a challenging task. Phone hardware is quite limited in comparison to their desktop cousins. In addition, most techniques that exist for dynamic water representation involve simulations or complicated solutions to water dynamics equations. The technique described in this paper largely succeeds in striking a balance between these two opposing priorities.

Using a clip-map as an LOD technique serves to effectively provide detail close to the user while not putting too much strain on performance. Superimposed Gerstner waves as a macro scale ocean behavior representation serves to provide an aesthetically pleasing, customizable, and efficient approach. And the texture based dynamic water for the micro scale adds a level of authenticity and visual appeal, while being broad enough to be applied to and extended upon in many applications.

There are a handful of exciting and interesting extensions to this research as well. Extending this approach to a boundless surface by making the clip-map the entire surface and modifying textures to provide the underlying displacement and effects extends the applications for this method dramatically to many kinds of games emphasising performance. The clip-mapping technique could also potentially be expanded to support other basic mesh shapes besides a plane.

The texture based effects have large versatility in what information is stored and used for what kind of effects. For example at the moment, only the red channel of the texture is used. One could store directional information in two other channels to create some other effect. And it is possible that with optimizations and future mobile hardware that a more physically accurate wave simulation could be integrated for water-geometry interactions.

The dynamic water wake effect works well and is a nice approximation of boat wakes, but on mobile still performs somewhat poorly at resolutions needed for the effect to be as pleasing as possible. Since the performance of the boat wakes relied more heavily on texture resolution than number of history points, techniques like signed-distance fields could potentially provide massive performance increases [9]. In addition the dynamic effect is only implemented for boat wakes, but in theory could be extended to support wakes around buoys by expanding full circles around a buoy's position, and possibly shoreline wave behavior using something like geodesic distance mapping to inform wave shapes [10].

This technique serves as a strong base for developers, and its versatility and customizability ensures it could be useful in a broad variety of applications wishing for performant water geometry generation. Furthermore, nothing is limiting the foundational method from being used for other applications, such as modeling landscapes with dynamic effects (e.g. moving grass around the player). This is especially true if the application needs to run well on weaker hardware, or is trying to optimize heavily, such as a virtual reality game. Overall with its relative simplicity and versatility, this technique can find a place in all manner of applications.

References

[1] E. Darles, B. Crespin, D. Ghazanfarpour, and J. C. Gonzato, "A survey of ocean simulation and rendering techniques in computer graphics," *Computer Graphics Forum*, vol. 30, 2011.

[2] S. Jeschke, T. Skřivan, M. Müller-Fischer, N. Chentanez, M. Macklin, and C. Wojtan, "Water surface wavelets," *ACM Transactions on Graphics*, vol. 37, 2018.

[3] A. Asirvatham and H. Hoppe, "Terrain rendering using gpu-based geometry clipmaps," *GPU Gems 2*, 2005. [Online]. Available: https://developer.nvidia.com/gpugems/gpugems2/part-i-geometric-complexity/chapter-2-terrain-rendering-using-gpu-based-geometry

[4] L. M. Lachman, "An open programming architecture for modeling ocean waves," 2007.

[5] J. Flick, "Catlikecoding unity wave implementation." [Online]. Available: https://catlikecoding.com/unity/tutorials/flow/waves/

[6] J. Tessendorf, "Simulating ocean water," *SIGGRAPH'99 Course Note*, p. 7, 01 2001.

[7] J. Wang, Z. Min, J.-L. Chen, and Z. Cai, "Application of facet scattering model in sar imaging of sea surface waves with kelvin wake," *Progress In Electromagnetics Research B*, vol. 67, pp. 107–120, 01 2016.

[8] M. DeLoura, "Game programming gems," *Interactive simulation of water surfaces*, pp. 187–194, 2000.

[9] C. Green, "Improved alpha-tested magnification for vector textures and special effects," 2007.

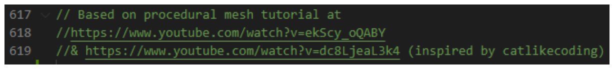
[10] S. Liu, X. Jin, C. C. L. Wang, and J. X. Chen, "Water wave animation on mesh surfaces," *Computing in Science Engineering*, vol. 8, pp. 81–87, 2006.

A Responsible Research

For research to be robust, regardless of the results, it should be conducted such that the research and methods are ethical, reproducible, and presented without bias. This research was conducted with these principles in mind.

A.1 Ethics

Ethical research should keep in mind the impacts this research may have on others. In the case of developing wave representation for an AR/VR application, there is not much that must be considered ethically speaking. The exception to this would be making certain that it is clear what code I have written is my own, and what takes inspiration from others. To that end I have made sure to leave comments in the code, such as in Figure 11, which highlight the contributions of others on the implementation side of this research, as to avoid plagiarism.



```
// Based on procedural mesh tutorial at
//https://www.youtube.com/watch?v=ekScy_oQABY
//& https://www.youtube.com/watch?v=dc8LjeaL3k4 (inspired by catlikecoding)
```

Figure 11: In-code references to resources that assisted with Unity implementation of concepts

A.2 Reproducibility

Reproducible research must be transparent, the steps taken should be traceable and and the methods used and where they came from should be clear. To that end, I have maintained a daily logbook[2] of my activities during the research project that is open to the public. In said logbook I keep track of what project related tasks were worked on each day. Furthermore, it provides links to web resources such as YouTube videos that were watched either for inspiration or to learn relevant tools such as Unity.

In addition, I keep a full reference list of papers, books, and conference proceedings along with some web resources using the reference management software Mendeley[3]. This list contains all references used for this paper along with any others I have investigated. If the reader wishes to view this list for research or peer review purposes, they may send an email.

Lastly, the code produced via this research project by me has been published to a public GitHub repository [4] for those who wish to view the concrete implementation.

A.3 Integrity

To fulfill integrity the presentation of research must not allow bias from the researcher color the results. This is important if the results are to be of use to those who wish to build upon said research, or to inform research in the future. Thus, when presenting results, a conscious effort has been made not to exclude results that reflect poorly on either the research or I as the researcher. Poor results such as the failure of the initial dynamic water representation system and the still sub-par performance of the texture based approach have been explained such that future researchers are informed on how and why certain methods were unsuccessful. The hope being that they may make the optimal choices for their future potential extensions to this research.

[2]https://seasoned-wind-8dd.notion.site/Logbook-50fbda9ad4b043969f7b2b5a1f994936

[3]https://www.mendeley.com

[4]https://github.com/lucjonker/water-animation-interaction-CSEBEP

B Wake Resolution Results

B.1 2048x2048 Texture

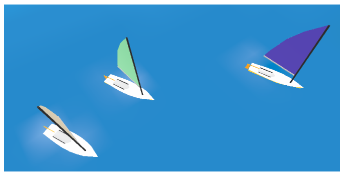


Figure 12: Two historic points

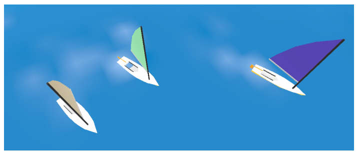


Figure 13: Six historic points

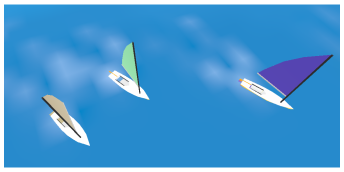


Figure 14: Ten historic points

B.2 4096x4096 Texture

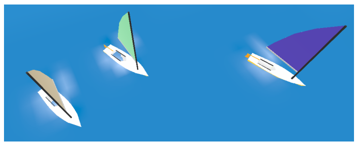


Figure 15: Two historic points

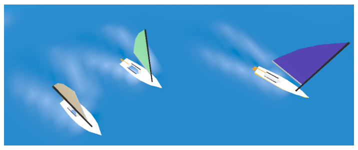


Figure 16: Six historic points

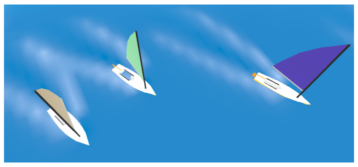


Figure 17: Ten historic points

B.3 6144x6144 Texture

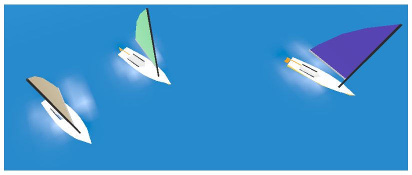


Figure 18: Two historic points

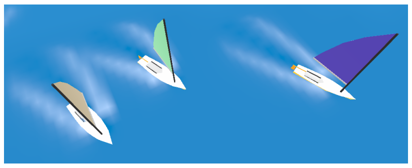


Figure 19: Six historic points

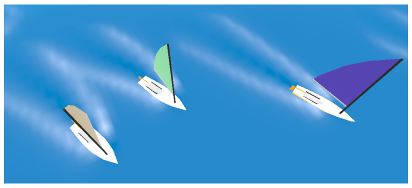


Figure 20: Ten historic points