Real-Time Procedural Ocean Waves Simulation

Josselin Somerville Roberts, Xiyao Li

***Abstract*—We present a procedural method for sea surface simulation. Our model is based on a superposition of many different waves generated according to some physical rules to reproduce natural phenomena. More precisely, we choose a wave spectrum approach to create the wave's main structure and Perlin noise for small scale details. Then, we use a layered BRDF with a diffuse term, a specular term using a microfacet model based on the GGX distribution and a global illumination term, for the rendering procedure to get a satisfying visual result.**

I. Introduction

Natural phenomena simulations are very common nowadays in the film and video game industry. In computer graphics, as the procedural simulation can use a limited set of rules to generate a large and complex scene with randomness, it is very widely used in natural phenomena simulation, i.e. forest generation, automated map generation and bio-mechanics simulation.

In this project, we work mainly on the ocean waves simulation for the animation part and a Physical-Based Rendering for the rendering part. There are already many classical approaches to simulate the ocean in computer animation. The Fast Fourier Transform is used by many. This approach is to sum several Fourier transforms for each vertex. So the advantage is the vertices' position and normal can be computed directly by integrated functions, and the visual result is satisfying.

Therefore, we use a procedural simulation to generate a dozen trochoids and sum all waves up to get a random phenomenon. To add some detail, we then add some Perlin noise and some rendering enhancements like sea foam. To enrich the scene, we later add a boat floating with the waves in the middle of the sea. For a better visual result, we also add shaders for waves and boat rendering using a complex layered BRDF.

II. Previous Work

A. Computer Graphics Models

At present, there are three main approaches to modelling waves. The first is a physically-based one in which the fluid's physical properties are obtained by computing the waves' Navier-Stokes equation. For example, Foster [ND00] obtains a wave model by solving the Navier-Stokes equation for the initial and boundary conditions, depending on the physical environment of the fluid. This method gives a sophisticated visual effect with realism, but it is computationally intensive and cannot meet real-time requirements.

The second approach is based on statistical modelling, for example, using empirical statistics to synthesise different sinusoidal waves to obtain a sea surface simulation. Premoze[PA01] proposes a natural sea surface modelling method that combines physical models with statistical data. The data models involved in this method are very complex, computationally intensive and have poor real-time performance.

The third approach is geometry-based, where the shape of the water waves is modelled by constructing certain mathematical functions to create an ocean surface. Peachey [Pea86] use a linear combination of sinusoidal functions to calculate the height field to represent the sea surface. And Fournier and Reeves [FR86] use a parametric surface controlling the water surface to solve the problem of wave modelling with curling crests. The Fast Fourier Transform is an improvement of this method. The basic idea is to produce a height field having the same spectrum as the ocean surface by filtering white noise with Pierson-Moskowitz's or Hasselmann's filter and then calculating its Fast Fourier Transform (FFT). The main benefits of this approach are that many different waves are simultaneously simulated, with visually pleasing results. However, animating the resulting ocean surface remains challenging.[HNC02]

B. Implemented Model Description

Our simulation model is based on the paper of Damien Hinsinger, Fabrice Neyret and Marie-Paule Cani.[HNC02] Ocean waves are generated by the wind in fetch areas and can propagate far from these locations. Here we simulate the waves' amplitude and frequency mainly by the wind's speed and direction.

We keep using a mesh whose position in world space will change dynamically. It leads to using a procedural wave model, following the third family of Computer Graphics approaches mentioned above. Our model is based on the Gerstner swell model and simulates trochoids by taking into account the combination of many different waves. To do so, we generate wave trains in a way that approaches a known wave spectrum, in the same spirit as Thon et al.[SDD00].

III. Waves Simulation

A. Main Structure

We simulate ocean waves using a series of wave trains that homogeneously cover the world. Their amplitude a_i, frequency $\frac{\omega_i}{2\pi}$ and direction $\frac{\mathbf{K_i}}{|\mathbf{K_i}|}$ are chosen randomly based on the wind's magnitude and direction. $\mathbf{K} = (\mathbf{K_x}, \mathbf{K_y})$ is a wave factor, with magnitude $|\mathbf{K}| = K$ and direction of the wave propagation $\frac{\mathbf{K_i}}{|\mathbf{K_i}|}$. The value of K is related to the

wave's angular velocity ω_i, so here we generate K by the following relation: $K \propto \frac{\omega^2}{g}$.

We consider the mesh representing the ocean surface at a given animation step. Each mesh vertices follows a circle trajectory corresponding to the Gerstner model:

$$\begin{cases} \mathbf{X} = \mathbf{X_0} + \sum_i a_i \frac{\mathbf{K_i}}{|\mathbf{K_i}|} \sin(\omega_i t + \mathbf{K_i} \cdot \mathbf{X_0}) \\ z = z_0 + \sum_i a_i \cos(\omega_i t + \mathbf{K_i} \cdot \mathbf{X_0}) \end{cases} \quad (1)$$

where $\mathbf{X_0} = (x_0, y_0)$ is the vertex position on the surface at rest and z_0 its altitude at rest.

B. Uniform Wave Distribution

More precisely, the angular velocity ω_i is chosen randomly between $[0, 2\pi]$, the direction is chosen according to the wind direction with an additional rotation within a range of $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Then, the wave amplitude a_i is computed as follows:

$$a_i = r_i * \frac{W_{intensity}}{N_{waves}} * \left| cos(W_{dir}, \frac{\mathbf{K_i}}{|\mathbf{K_i}|}) \right|^c$$

With r_i a random uniform value between 0 and 1, $W_{intensity}$ the amplitude of the wind, W_{dir} the direction of the wind, N_{waves} the total number of waves and finally c a constant called *coherent factor*. The coherent factor makes the waves follow more or less the direction of the wind. With a value of 0, there is no constraint. And with a value close to infinity, it gives an amplitude of 0 to waves not following the wind direction. In our implementation, four yields usually give pleasing results.

After summing twenty waves, we get the following result:

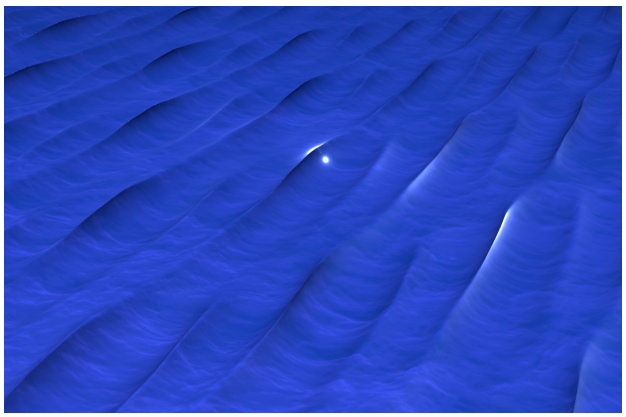

Fig. 1: Superposition of 20 waves

C. Small Scale Details

We then use the approach of Thon et al.[SDD00] to add details. The idea is to add a 3D turbulence function as small details to the main structure. Here we chose a Perlin noise to generate some random effects. The Perlin Noise used takes as an argument a vector of dimension 3 and returns a scalar representing the additional height for a given vertex. The inputs are the time and the planar position after deformation. It is essential to use the deformed planar position to make the local details follow the wave. Moreover, it gives a less static effect to the random terrain added on top of the waves. Regarding the output, a 3D deformation could have been used but would lead to even more self-intersection which we want to avoid. As the 1D deformation gives satisfying results, we decided to stick with it.

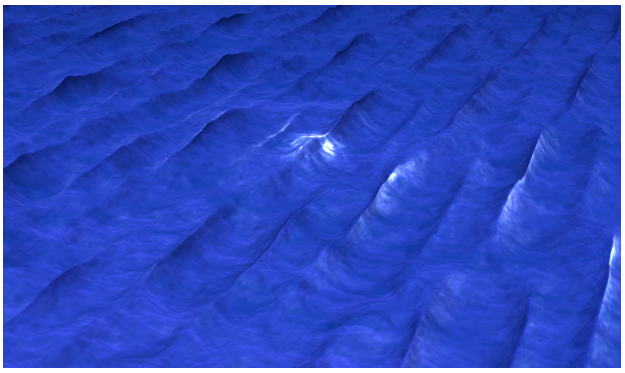

Fig. 2: Result after adding Perlin Noise *(amplitude=0.36, octaves=5, persistency=0.4, frequency gain=2.2)*

IV. Boat Animation

Our approach for simulating waves is described so far as a process that computed vertex by vertex. Its advantage is being extremely efficient as it can run on the GPU. However, it makes the simulation of floating objects tricky. Often, floating objects are simulated thanks to a group of particles in a liquid also represented as particles. It is then straightforward to simulate a floating object by simply computing a few forces on each particle.

As we simulate the sea surface by deforming a mesh, the method we mentioned above is not possible. We propose a simple approach, attaching a rigid triangle to the ship, which always stays on the surface. The ship's movement then follows the given algorithm:

- Save the position of the triangle on top of the water without waves.
- Compute the moved position of each point due to the waves.
- compute the translation of the center between the two poses as well as the rotation of the triangle.

One important aspect is that the triangle ratios will change due to the deformation. It is not critical as we only use the centre of the triangle and its orientation.

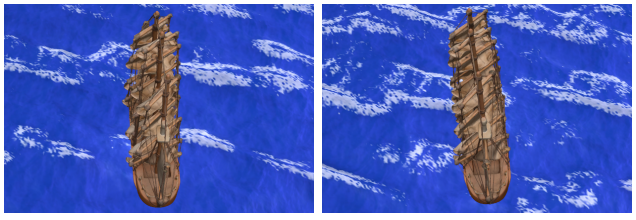

Fig. 3: The ship tilting due to the movement of the waves

To avoid jittering, there are three improvements. First, we reduce the number of octaves in the Perlin noise considered for the boat simulation. This small deformation changes bearly the water level. But it results in major orientation differences, which lead to the boat tilting way too much.

Secondly, using more triangles can help to stabilise the boat. The first step is to approximate the outline of the boat hull (at rest) to a simple ellipse. Then we sample triangles by taking the centre of the ellipse and two points on the ellipse (regularly spaced: $\theta = 2\pi/N_{triangles}$). Finally, we compute the movement caused by each triangle and take the mean of both the translation and the rotation.

Finally, the last improvement is to add some *pseudo-physics* in the movement computation. We add the notion of *mass (m)* and *friction (K)*. After different analogies to force computation, the following gives the best result (with the notation O the orientation, i.e. rotation or translation):

$$O_n = O_{n-1} + \frac{\hat{O_n} - O_{n-1}}{m\sqrt{(1+K)}} d_t$$

with $\hat{O_n}$ the orientation computed at time n with the movement of the triangles.

V. Rendering and additional animation

A. Layered BRDF

In order to get a simulation as realistic as possible, it is important to not only work on the animation but also on the rendering. The interaction of the light with the surface of an object can be computed as incident and outgoing light by the BRDF (Bidirectional Reflectance Distribution Function) lighting model. Based on this, we can easily render light from ideal light sources, i.e. point and directional light sources. But the illumination of real-world environments is much more complex than these two ideal light sources. On the one hand, there is the presence of regional light. For example, the sea surface does not only receive sunlight but also reflects light from the sky. In fact, in the real world, most light sources exist in the form of area light. On the other hand, the phenomenon of light bouncing between the surfaces of objects is also a very important component of light, i.e. indirect light.

Therefore we have implemented a layered **BRDF model** with diffuse, specular and global terms. The renderisation follows the equation:

$$C = K_a * obj_{color} + \sum_{j=0}^{N_{lights}} r_j$$

with r_j the diffuse and specular contribution of the light j computed as follow:

$$r_j = (K_d * f^d(\omega_i, \omega_0) + K_s * f^s(\omega_i, \omega_0)) * max(0, < n, \omega_i >)$$

with K_a, K_d and K_s the proportion of the ambient, diffuse and specular terms.

and n the normal, ω_i the light direction and ω_0 the outgoing direction. For convenience later, we define $\omega_h = \frac{(\omega_i + \omega_0)}{||\omega_i + \omega_0)||}$.

The diffuse term f^d is defined as $f^d(\omega_i, \omega_0) = \frac{obj_{color}}{\pi}$

The specular term f^s follows the Microfacet model:

$$f^s(\omega_i, \omega_0) = \frac{D_{GGX}(\omega_i, \omega_0) F(\omega_i, \omega_0) G(\omega_i, \omega_0, \omega_h)}{4 < n, \omega_i >< n, \omega_0 >}$$

with D_{GGX} the normal distribution following (here GGX):

$$D_{GGX}(\omega_i, \omega_0) = \frac{obj^2_{roughness}}{\pi(1 + (obj^2_{roughness} - 1).< n, \omega_h >)^2)^2}$$

and F the Fresnel term. Using the Schlick Approximation, with F_0 the Fresnel reflectance, we get:

$$F(\omega_i, \omega_0) = F_0 + (1 - F_0)(1 - max(0, < \omega_i, \omega_h >))^5$$

$$F_0 = obj_{color} + (1 - obj_{color}) * obj_{metalicness}$$

and finally G the GGX Geometric Term. Using the Smith operator, using the notation $\alpha = obj_{roughness}$:

$$G^{Smith}(\omega_i, \omega_0) = G_1^{Smith}(\omega_i) G_1^{Smith}(\omega_0)$$

$$G_1^{Smith}(\omega) = \frac{2 < n, \omega >}{< n, \omega > + \sqrt{\alpha^2 + (1 - \alpha^2) < n, \omega >^2}}$$

As the sea is quite homogeneous, we use constant values and no textures. Only the color (often called Albedo) is sampled. No texture is required because we already added some Perlin noise when computing the vertex positions and normals. Regarding the specular coefficient, we chose a constant value for the entire sea.

B. Seafoam Rendering

To add even more realism, we decided to add seafoam. While previous works have already tackled this problem, we decided to implement our method. Here are some factors that lead to the generation of seafoam:

- The fragment is on top of a wave.
- The gradient of the normal at this fragment is high.
- The normal of the fragment tends to be horizontal, which means the fragment is on a steep slope.

To add seafoam, for each fragment, we compute a seafoam coefficient c_{sf} and above a certain threshold $c_{sf_{min}}$ we display seafoam.

To take into account all the previously mentioned points, we propose the following formula:

$$c_{sf} = min(1, c_{grad}.c_z.c_n)$$

$$c_{grad} = \alpha_g.max(0, \max_{\{u \in \mathbb{R}^2, ||u||=1\}} \nabla_u n)$$

$$c_z = min(1, p_z - h_0)$$

$$c_n = (1 + h_\epsilon - < n, u_z >)$$

c_{grad} represent the variation of the normal, c_z the influence of the height and c_n the steepness of the wave.

To compute c_{grad} we need to compute the maximum directional derivative of n. To do so, we iterate through a few directions but this is highly ineffective. So we choose to parametrize u to simplify the expression. With $u = (cos(\alpha), sin(\alpha))$:

$$\max_{\{u\in\mathbb{R}^2,||u||=1\}} \nabla_u n = \max_{\{\alpha\in[0,2\pi]\}} (cos(\alpha)\frac{\partial n}{\partial x} + sin(\alpha)\frac{\partial n}{\partial y})$$

If $|\alpha| \neq \frac{\pi}{2}$, the derivative of the previous function to maximise is equal to 0 for $tan(\alpha) = (\frac{\partial n}{\partial y})/(\frac{\partial n}{\partial x})$. After reinjecting this into the function to maximise, we find an explicit formula for the maximum which only requires to compute the gradient:

$$\max_{\{u\in\mathbb{R}^2,||u||=1\}} \nabla_u n = \begin{cases} |\frac{\partial n}{\partial y}|, & \text{if } \frac{\partial n}{\partial x} = 0 \\ |\frac{\partial n}{\partial x}|\sqrt{1 + \left((\frac{\partial n}{\partial y})/(\frac{\partial n}{\partial x})\right)^2}, & \text{if } \frac{\partial n}{\partial x} \neq 0 \end{cases}$$

It is especially efficient if we compute all the derivatives and gradients with finite differences and keep track of the points computed. Only 6 points are required:

$$\frac{\partial n}{\partial x}(u,v) \approx \frac{n(u+\epsilon,v) - n(u,v)}{\epsilon} \quad (2)$$

$$= \frac{\frac{\partial f}{\partial x}(u+\epsilon,v) \wedge \frac{\partial f}{\partial y}(u+\epsilon,v) - \frac{\partial f}{\partial x}(u,v) \wedge \frac{\partial f}{\partial y}(u,v)}{\epsilon} \quad (3)$$

which can be computed in terms of $f(u+2\epsilon,v)$, $f(u+\epsilon,v)$, $f(u+\epsilon,v+\epsilon)$, $f(u,v+\epsilon)$ and $f(u,v)$. We get the same dependence for $\frac{\partial n}{\partial x}(u,v)$ with in addition $f(u,v+2\epsilon)$.

Now that a coefficient c_{sf} can be computed for each fragment, we need to display some seafoam for all $c_{sf} \geq c_{sf_{min}}$. To do so, we replace the fragment colour previously equal to obj_{color} by:

$$f_c = obj_{color} + (foam_{color} - obj_{color}) * \left(\frac{c_{sf} - c_{sf_{min}}}{1 - c_{sf_{min}}}\right)^{\alpha_e}$$

Even after tuning values for this model, it is hard to have a good visual result. The high frequencies in the Perlin noise add so much detail that the normal variation can be huge where we do not want to see seafoam. To solve this issue, we compute the derivative of the normals using a Perlin noise with fewer octaves and a fine step for the finite difference calculus. After some tests, we found that 3 octaves are the maximum yielding good results.

As an improvement, to reduce the effect of seafoam flashing for several pixels due to the value of the seafoam coefficient c_{sf} varying close to $c_{sf_{min}}$, it would be a good idea to add hysteresis. However, to do so, the previous seafoam value is required. This could be done with a texture.

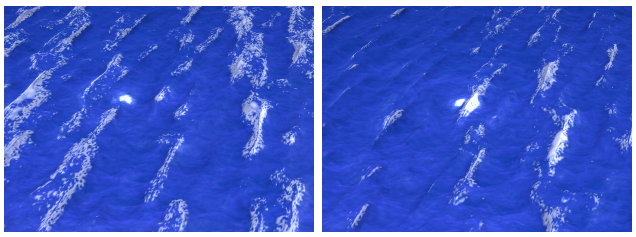

Fig. 4: Seafoam rendering with $N = 5$ octaves on the left and $N = 3$ octaves on the right

C. Reflections on the sea

One key aspect of the sea rendering is how it reflects the light. In fact, with no wind, the sea looks like a mirror, and even with waves, it is still important to show the reflected light. For example, during sunset the water becomes reddish.

To tackle the change of the water's colour based on the environment, we add a skybox, and we use an environment mapping. This simple method shows goods results. It allows having significantly different results from the same simulation, whether it is at night or during a sunset.

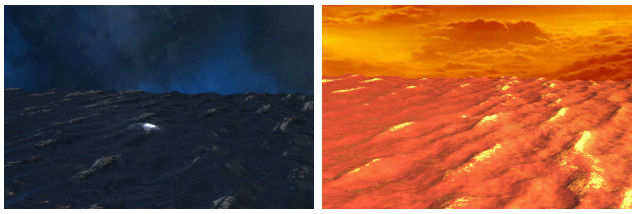

Fig. 5: Environment mapping during night time on the left and sunset on the right

As an improvement, for more complex reflections, like the boat, a good solution would be Screen Space Reflections(SSR). It consists of using deferred rendering to first render every useful component as textures into the g-buffer, such as normal, depth, positions, seafoam coefficient, etc. Then we perform a second pass, rendering the textures while doing some ray casting directly in the texture to approximate reflections. It is particularly efficient and should give pleasant results in this case.

VI. Discussion and Conclusion

Our method focuses on the generation of random waves according to the Gerstner model and their superposition for creating an ocean surface. After adding shaders for the ocean, we get a delightful scene. With the superposition of 40 waves, our ocean simulation algorithm can be run in real-time (120FPS on a Nvidia GTX1650 GPU and Intel i7 - 9th generation CPU) on a 500*500 mesh, which provides precise details. The algorithm's computational complexity of $O(1)$ per vertex, gives us a pleasant result.

However, the current animation only provides good results on calm seas. We are not able to get satisfying visual results

Fig. 6: Final result with the ship, 40 waves, seafoam and all the rendering improvements

for large magnitude waves. The FFT algorithm may provide a more realistic and detailed simulation of windy and rough seas with more randomness. Moreover, computing the deformation on the GPU as we did, does not allow to recompute the mesh connectivity, often leading to self-intersection with waves of high amplitudes. It could be solved by running everything on the CPU (which would kill the performance) or by writing to a texture the height of the water at the pixel corresponding to the horizontal position deformed. It would lead to some *holes* in the texture, but this can be fixed by linear interpolation.

Finally, our algorithm computes several times the same height for a vertex. To compute the normal and its derivative, we calculate six points per vertex. We can replace this step by using deferred rendering and multi-pass shading. The first step is simply computing and storing the deformed position of the wave into a texture. Then, we could use the texture to compute normals and their derivatives, ... *(having an analytic formula for normals and derivatives is not trivial due to the noise, and could require more computation than simply using a finite difference method as we did).*

References

[FR86] Alain Fournier and William T. Reeves. “A simple model of ocean waves”. In: *Dallas, August 18-22* 20 (1986).

[Pea86] Darwyn R. Peachey. “Modeling Waves and Surf”. In: *Dallas, August 18-22* 20 (1986).

[ND00] Foster N. and Metaxasn D. “Modeling water for computer animation”. In: *CormnunACM* 43(7) (2000).

[SDD00] THON S., J.-M. DISCHLER, and GHAZANFARPOUR D. “Ocean waves synthesis using a spectrum-based turbulence function”. In: *Computer Graphics International Proceeding* (2000).

[PA01] Simon Premoze and Michael Ashikhmin. “Rendering Natural Waters”. In: *COMPUTER GRAPHICS forum* 20 (2001).

[HNC02] Damien Hinsinger, Fabrice Neyret, and Marie-Paule Cani. *Interactive Animation of Ocean Waves.* 2002. eprint: 10 . 1145 / 545261 . 545288 (inria-00537490).