Fredrik Limsäter Mårten Larsson

Abstract

In this paper we implement some of the known water models for simulating surfaces in real-time. We also implement an easy-to-use text interface, allowing the user to change conditions in run time.

Introduction and Goals

This paper implements some of the known water models for surface simulation.
This includes:

- Gerstner Waves
- Fast Fourier Transforms
- Statistical Wave Models (Phillips)

Our goal has been to make it possible for us to render this in real-time with a quality that resembles of real ocean surface. These notes will describe for you how to make a height-field displacement mapped surface that will look like figure 1. We will not cover optical effects though, like reflection/refraction and transmission, since it is out of scope for this project.

Ocean Wave Algorithms

To generate an ocean surface we need to build a height-field. To get the best result we use a method based on Fast Fourier Transforms (FFT), although we will start with a simpler description called Gerstner Waves. Complete description can be found in [1].

Gerstner

Gerstner waves are almost 200 years old and were used to approximate the solution to fluid dynamics. It describes the surface in terms of the motion of individual points on the surface. If a point on the undisturbed surface is labelled $\mathbf{x}_0 = (x_0, z_0)$ and the undisturbed height $y_0 = 0$, the point on the surface is displaced at time t to

$$x = \mathbf{x}_0 - (\mathbf{k}/k)\, A\sin(\mathbf{k}\bullet \mathbf{x}_0 - \omega_0)$$
$$y = A\cos(\mathbf{k}\bullet \mathbf{x}_0 - \omega_0)$$

when a single wave with amplitude A passes by.
The wave vector $\mathbf{k}$ points in the direction of travel of the wave, and it has magnitude

$$k = 2\pi/\lambda$$

This will only result in a single wave passing the surface, not very realistic. Instead we are summing a set of sine waves to create a more complex profile.

$$x = \mathbf{x}_0 - \sum((\mathbf{k}_i - k_i)\, A_i \sin (\mathbf{k}_i\bullet \mathbf{x}_0 - \omega_i t + \Phi_i))$$

$$y = \sum(A_i \cos (\mathbf{k}_i\bullet \mathbf{x}_0 - \omega_i t + \Phi_i))$$

To animate the Gerstner waves we change the frequency ω_i. There is a known relationship between these frequencies and the magnitude of the wave vector k_i.

$$\omega^2 (k) = gk \qquad (g \text{ is } 9.8 \text{ m/sec}^2)$$

FFT

Here we will present the algorithm that produces the best result and it has been used commercially several times. It uses statistical models based on observations of the real sea. (Used in Titanic and Waterworld).

In the statistical model of sea, wave height is a random variable of horizontal position and time

$$\mathbf{h}(\mathbf{X},t)$$

It decomposes the wave height-field into a set of sinus waves with different amplitudes and phases. We use FFT to evaluate these sums.

FFT allows us to evaluate the following

$$\mathbf{h}(\mathbf{X},t) = \sum(h\,(K,t)e^{iK*K})$$

The wave vector K points in the direction of travel of the wave, and it has magnitude

$$k = 2\pi/\lambda$$

When we want to generate a height-field we start by calculating h_0

$$h_0(K,t) = 1/\mathrm{sqrt}(2)(\mathrm{Gaussian} * (\mathbf{i})\mathrm{Gaussian})\mathrm{sqrt}(P_h(K))$$

where

$$P_h(K) = a\,(e^{-1/(kl)\hat{}2})/k^4 \;\; |KW|^2$$

is the Phillips spectrum.

The parameter l is the windspeed^2 / gravity

The last term $|KW|^2$ eliminates waves moving perpendicular to the wind direction.

Now when we have calculated $h_0(K,t)$ we can animate the set with

$$h(K,t)= h_0(K,t)\, e^{i\omega(K)t} + \mathrm{conj}(h_0(K,t))\, e^{-i\omega(K)t}$$

$$\omega^2 (k) = gk \qquad (g \text{ is } 9.8 \text{ m/sec}^2)$$

Choppy

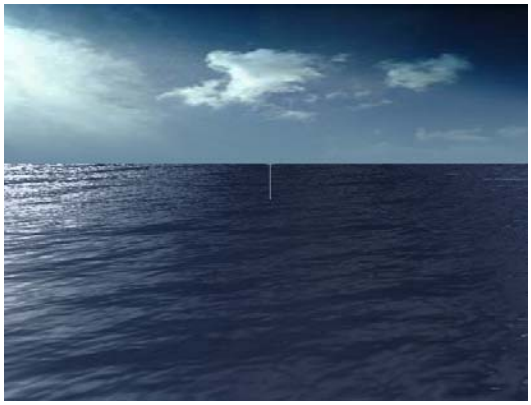
The FFT transform generates nice looking waves that all have round tops. This is fine for nice weather situations, but no good for stormy conditions. In order to make the waves have more sharp tops an algorithm can be implemented.

The idea is to displace the grid points slightly in the horizontal plane to make the waves more sharp and the valleys between the waves wider.
The equation used looks as follows:

$$X = X + \lambda D(X,t)$$

Where X is the vertical position X=(x, z), λ is a scale factor and D is the displacement vector calculated with the FFT:

$$D(X,t) = \sum_k -i\frac{K}{k}\tilde{h}(K,t)e^{iKt}$$



Figur 1 FFT waves with resolution 512*512

We have noticed that a resolution of 128 can be rendered in real-time with a good visual result. However, a resolution of 512 gives the best result compared to rendering time.

Figur 2 FFT waves with resolution 512*512

Rendering and Animation

The dataset can be rendered in several different ways. The user can render the vertecies as points, or as a triangle mesh (wire frame or solid).
The surface is texture mapped with reflection mapping that can be blended with surface colors by the user. The reflection texture can also be changed in runtime.
It is also possible to render the surface normals.
The user can also render the amplitudes fed into the Fourier transform.

Functions to save renderings from the application makes it possible for the user store images to a directory. These images are numbered in a sequence and can be used to make an animation of a dataset that is too large to be rendered in real time.
The system uses a dynamic step size that depends on frame rate. When rendering to a file is turned on the step size is set to a fixed 0.04 seconds per frame (25 frames per second).

All data in the application is stored in one dimensional arrays to speed up the application.
The double values used to set normals and grid points are fed directly in to the calculations to avoid extra loops and gain performance. At the moment the application runs in 30-50 frames per second with a grid resolution of 64*64 on a 700 MHz laptop with a ATI Mobility M4 graphics card.

To control the application the user is given a text console. Commands can be typed in to set variables, switch rendering or normal mode, or save images.

We have implemented the algorithms using a PC platform with windows. We programmed it using C++ and OpenGL. The FFT came from [3].

Summary

We feel that we have achieved our goal to implement water surface animation models in a real time application. It runs 30-50 frames per second on a 700 MHz Pentium laptop in resolution 64*64 on the grid. We have also gone beyond this goal and implemented a save rendering function that makes it possible to save renderings of datasets too high for real time performance.
Our application can also be used to compare two different surface animation models, and to demonstrate the fast Fourier transform.

References

[1] Jerry Tessendorf "Simulating Ocean Water" SIGGRAPH 2001 Course Notes

[2] Lasse Staff Jensen "Deep Water Animation and Rendering" Funcom Oslo AS

[3] "Numerical Recipes in C++ , The Art of Scientific Computing"
2nd Edition ISBN 0521750334

[4] "OpenGl Programming Guide, 3rd Edition"
ISBN 0201604582