REALISTIC ENVIRONMENTS FOR ONLINE MARITIME SIMULATORS

Gabriel Raicu, Eugen Barsan, Arsenie Paul, Radu Hanzu, Laurentiu Chiotoroiu

Constanta Maritime University, Faculty of Navigation and Naval Transport,
90663, Mircea cel Batran Str. 104, Constanta, Romania

graicu@imc.ro (Gabriel Raicu)

Abstract

This paper presents the 3D and network principles and methods for graphical development in realistic naval simulation.
The aim of this project is to achieve a good simulation quality in large networked environments using open source solution approach for educational purposes. Realistic rendering of maritime environments requires that the sunlight and skylight illumination are correct and the water surface is modeled accurately.
For online simulators the problems that you have to deal with depend a lot on the type of network you are using. Whatever the platform is, you have to deal a multiprocessing situation. LANs make a very easy platform for writing networked simulators, but unfortunately it means that the participants have to have their computers connected to high speed network in order to be able to run the software. This limits the number of workstations in use. The Internet has one thing going for it: there are a lot of potential participants (students, instructors) on it at all times. TCP is a full duplex connection-based reliable transport protocol. It offers reliability at the cost of increased latency variance. Network errors cause automatic retransmissions from the TCP protocol, so at times connection latency can be several times higher than optimal. Obviously, the main advantage of networked simulators is that you get to participate with new people and possibly even make new teams. To achieve this goal, the simulator should be as accessible as possible.

Keywords: realistic simulation, online participants, maritime environments, networked environments.

Presenting Author's biography

Gabriel Raicu. The main author activity as lecturer at Faculty of Navigation and Naval Transport in Constanta Maritime University consists in large scale online educational developing services and electronic navigation aid teaching.
Also all the co-authors are maritime lecturers, associate professors or assistant professors at Faculty of Navigation and Naval Transport in Constanta Maritime University.

ISBN 978-3-901608-32-2

1 Online simulators architecture

Client-server approach are well known in networked environments. In a client-server architecture all the simulator participants, or "clients", are connected to a central machine, the server. The server is responsible for all important decisions, managing state and broadcasting this information to the individual clients. A single view of the world is maintained by the server, which obviously helps with keeping things consistent.

As a result, the server becomes a key bottleneck for both bandwidth and computations. Instead of distributing the load among all the participants, the server must do all the work. And, of course, it has to send and receive N independent streams of data, so its network connection is similarly taxed.

Sometimes the server will be running on a participant's machine as a "local server" or a "listen server". The rules still apply in this case, because the client and server are logically decoupled even if running on the same physical system.

A peer-to-peer system spreads the computational load out among all the participants. If you have 8 participants, each with a computer, then it's nice to leverage all the available computing power. The downside, of course, is that "computation" means "decision making", so cheating can become rampant (each client can be hacked to report results favorable to that specific participant). In addition, the system is more susceptible to consistency errors since each peer has to make sure that it broadcasts its "decisions" and it must base this on the data provided by the other peers. If a peer falls off the network or doesn't get correct information in a timely manner, synchronization failures can and will occur since it's analogous to a CPU failing in a multiprocessor computer.

The advantage of a peer-to-peer server is that overall bandwidth and computational requirements for each system are reduced, and you don't need a single beefy server responsible for managing the entire simulator.

2 The best network architecture

In reality, most architectures are hybrid systems as going to an extreme in either direction can lead to significant problems.

For example, in a true client-server system, the client would never move the participant until the server responded with a "based on your last input, here is your new position". This is fine, assuming you have client-side prediction (discussed later), but this means that the server is handling all collision detection. This excessively computationally expensive, to the point that it's not tenable for large worlds.

A compromise would be to allow the clients to manage their own movement, and they in turn report their location to the server (which likely does some basic sanity checking on the reported movement). This leverages the computing power of each client and off loads a tremendous amount of work from the server.

2.1 Online simulators paradigm

This is a face-by-face approach between client-server and peer-to peer models.
A set of problems may require most of the efforts:

- networking topology: client-server vs. peer-to-peer
- computing model: distributed object vs. message passing
- which protocol to use? tcp, udp, reliable udp
- bandwidth limitation
- latency limitation

Simulation synchronization is the most important condition: order moves by their times of occurrence because out-of-synch worlds are inconsistent. Small inconsistencies not corrected can lead to large compounded errors later on.
How long do you have to wait for the other participant moves before rendering them?
Each participant receives all other moves before rendering next frame. Some problems may occur:

- long Internet latency
- variable latencies
- speed determined by the slowest participant

Every participant must see the EXACT same world and each participant simulates its own copy of the world.
All the worlds must be in sync using bucket synchronization, each participant sends moves to all other players.

2.2 Physics and integration issue

Simulation-based software generally calculate entity state in conjunction with some physics code. This may be as simple as calculating an object's new position based on its velocity, or may be as complex as a full vehicle representation in a specialized engine, such as Havok. In either case, the calculation can be viewed as a numerical integration method. For example, you may integrate an object's velocity over time to find it's new position, i.e.,

Unfortunately, the results of these kinds of methods may diverge depending on the granularity at which you run them. If you simulate two initially identical physics objects at 10 Hz and at 20 Hz, they will end up in different states. This is true for any object whose physics is of higher than order(1) with respect to simulation time. For example, if we added acceleration to the object that previously had a constant velocity, we will cause integration error. In generation, higher order physics leads to a greater amount of integration error. Integration error always exists in comparison to the hypothetical "real" state,

ISBN 978-3-901608-32-2

which would have to be calculated with infinitely small granularity. It is only problematic in simulators where the discrepancy may be noticed. Games and simulators in which different computers simulate at different rates (e.g., PC games) are an example.

2.3 Latency

Latency, sometimes called "lag" refers to the delay between a piece of data being sent on a network, and that piece of data being received. Latency may be affected by various algorithms that act on the data in order to get it to its destination. For example, if a piece of data is lost on the network, it will eventually be retransmitted after a certain amount of time. From the receiver's point of view, this amount of time increases the latency. From the point of view of a piece of code in a computer simulator, latency may also be increased by the rate at which the simulator loop runs. For example, if a simulator runs at 30 Hz, it can only send and receive data at intervals of one thirtieth of a second. This may add up to a total of an extra 60 milliseconds latency from the point of view of the sending and receiving code.

2.4 Quantization Error

This is sometimes called "sampling error". For bandwidth conservation purposes, it is common practice to reduce the precision with which you send data values. For example, you may represent the avatar's camera angles with 8-bit values instead of with 32-bit values. The quantization error present in a value is the difference between its original value and its value after it has been quantized to a lower precision value.

The term "Quantization Error" is sometimes used to mean the same thing as integration error, which may be valid from the point of view of a hypothetically continuous simulation state being sampled at discrete intervals.

2.5 Prediction - a corrections approach

Sometimes just called "prediction", this technique allows a participant's computer to predict simulation events in an attempt to minimize the effects of network latency. It is normally used in the context of client/server structure, in which it refers to clients that optimistically predict the server's response to user commands. This mostly eliminates latency that would otherwise be perceptible to the user. When predicting an object (such as the participant's own object) the client mimics the operations that the server performs when it receives client commands. This might include the application of user commands to the predicted objects, and stepping the simulation forward in time. When the participant eventually receives an update about its avatar from the server, it must merge the update with its own predicted view of the world. Under the best case scenario, the client and server predictions will always be subject to divergence, due at the very least to integration error, introduced by their differing simulation rates. Prediction is further complicated because the predicted entities are simulated into the future with respect to simulator time (i.e. server time), and that when updates are received from the server, they describe state with respect to some time in the past. Prediction is usually a necessary technique in a real-time simulator, however its implementation interacts with many other net code features, and a great deal of testing is generally necessary to verify that the implementation will work under real network conditions, and deliver the best results possible. Prediction can also be performed in peer-to-peer approaches, in which event each participant's computer predicts their own avatar beyond the official agreed state of the simulation.

2.6 Enhancement techniques

This is sometimes referred to as "smoothing" or "blending". It is a technique that enables participants that are updated at discrete intervals about each other's state to render the transition smoothly. In simulations where participant movement is due mainly to a input as opposed to the passing of time (e.g., participant movement in an scene), interpolation is often used as the primary method by which to move avatars. In this case, the each avatar is interpolated towards a target state over the average update interval. This requires very regular updates, and results in interpolated entities being rendered one update cycle in the past. However, interpolation produces a correct (if slightly old) rendering of highly unpredictable entity movement. Interpolation also has an important role to play when applying updates to extrapolated entities. Small errors are inevitable (due to integration error), and interpolation algorithms can be re-used to perform a blending of the two states, avoiding perceptible state "snaps".

2.7 Implementations

Most 3D engines implement either Level-of-Detail or BSP algorithm (or both) to improve the performance of the renderer. Where possible, these algorithms should be re-used to improve networking performance. For example, a Level-of-Detail algorithm could be used to reduce the frequency of updates about distant entities, or entities that are not of great interest. Similarly a space-partitioning algorithm could be used to avoid sending update about entities that are neither visible nor audible. Such algorithms significantly increase network scalability and efficiency, however they complicate the implementation of delta compression, because they result in different base-line states from which any deltas must be calculated.

3 Quality of simulations

Realistic effects require a lot of computational power, but modern dual-core CPU's and SLI/Crossfire video architecture can satisfy this requirements.

Ordinary ocean waves are created by the wind in fetch areas and can propagate far from these locations. Several models

have been proposed to account for the amplitude, frequency and direction spectrum according to the wind strength and duration. A classical one is The Pierson-Moskowitz filter giving the amplitude F_{PM} in function of the frequency f:

$$F_{PM}(f) = \frac{ag^2}{(2\pi)^4 f^5} e^{\frac{5}{4}(\frac{f_m}{f})^4} \quad (1)$$

where:

g is the gravity acceleration

a is the Phillips constant

$f_m = \frac{0.13g}{U_{10}}$ corresponds to the peak in the spectrum (which has a Gaussian-like shape), depending on the wind velocity at a 10m altitude.

The ocean environment, for our purposes, consists of only four components: The water surface, the air, the sun, and the water volume below the surface. In this section we trace the flow of light through the environment, both mathematically and schematically, from the light source to the camera. In general, the radiosity equations here are as coupled as any other radiosity problem. To a reasonable degree, however, the coupling can be truncated and the simplified radiosity problem has a relatively fast solution.

The light seen by a camera is dependent on the flow of light energy from the source(s) (i.e. the sun and sky) to the surface and into the camera. In addition to specular reflection of direct sunlight and skylight from the surface, some fraction of the incident light is transmitted through the surface. Ultimately, a fraction of the transmitted light is scattered by the water volume back up through the interface and into the air. Some of the light that is reflected or refracted at the surface may strike the surface a second time, producing more reflection and refraction events. Under some viewing conditions, multiple reflections and refractions can have a noticeable impact on images. For our part however, we will ignore more than one reflection or refraction from the surface at a time. This not only makes the algorithms and computation easier and faster, but also is reasonably accurate in most viewing conditions and produces visually realistic imagery. At any point in the environment above the surface, including at the camera, the total light intensity (radiance) coming from any direction has three contributions:

$$L_{ABOVE} = rL_S + rL_A + t_U L_U \quad (2)$$

where

r is the Fresnel reflectivity for reflection from a spot on the surface

of the ocean to the camera

t_U is the transmission coefficient for the light LU coming up from the ocean volume, refracted at the surface into the camera.

L_S is the amount of light coming directly from the sun, through the atmosphere, to the spot on the ocean surface where it is reflected by the surface to the camera.

L_A is the (diffuse) atmospheric skylight

L_U is the light just below the surface that is transmitted through the surface into the air.

While equation 1 appears to have a relatively simple structure, the terms L_S, L_A, and L_U can in principle have complex dependencies on each other, as well on the reflectivity and transmissivity.

3.1 Wawes shape

Several models characterize the shape of waves by studying eigenmodes of the Navier-Stokes equation at the water-air interface. A convenient one is the Gerstner swell model, which describes the trajectory of water particles as circles of radius equal to the wave amplitude A around

the location at rest. Two particles along the direction of wave propagation having a distance at rest of l follow their circular trajectories at angular velocity w with a phase difference of kl:

$$\begin{cases} x - x_0 = Ae^{kz_0}\sin(\omega t - kx_0) \\ z - z_0 = Ae^{kz_0}\cos(\omega t - kx_0) \end{cases} \quad (3)$$

where t is the time, z the vertical axis and (x0; z0) the particle location at rest. This generates a trochoid wave shape, similar to a sinusoid only for very small amplitudes. For high amplitudes the waves get choppy, up to a value for which the curve crosses itself, which is no longer physical since the wave should break.

Other models, like Stokes and Biesel's ones, take into account the shallow water case, for which the circles turn into ellipses, at the price of more complicated formulas. Since the depth variation changes the wave velocity, the phases are no longer linear with the distance. Biesel's model thus evaluates the phases as $\int_0^{x_0} k(x)dx$. This change is responsible for the refraction of wave trains1close to the shore.

3.2 Water wawes model

We are looking for a wave model that does not constrain us to simulate a predetermined and regularly sampled surface region. Moreover, we are not willing to compute a high field, which would not cover the case of stormy seas. We keep using a mesh (since we cannot afford pixel size elements), but its location in world space will change dynamically and its density varies in space.

Our model is based on the Gerstner swell model and simulates trochoids. However, we want to take into

account the combination of many different waves. To do so, we generate wave trains in a way that approaches a known wave spectrum.

Although our method can be applied to various kinds of waves this paper only focuses on the main ones, i.e. gravity waves. We simulate gravity waves using a series of wave trains that homogeneously cover the simulated world.

Let us consider the mesh that represents the ocean surface at a given animation step. The mesh vertices are considered as particles, and thus follow the circle trajectory corresponding to the model:

$$\begin{cases} X = X_0 + \sum_i a_i \frac{K_i}{|K_i|} \sin(\omega_i t - K_i . X_0) \\ z = z_0 + \sum_i a_i \cos(\omega_i t - K_i . X_0) \end{cases} \quad (4)$$

where X0 = (x0; y0) is the location of the particle at rest on the surface and z0 its altitude at rest. Note that the only information that needs to be stored in memory is the specification of the wave trains: particles are evaluated on the fly, and can be at different locations from one frame to the other. Surface displacement is thus evaluated much like a procedural function.

3.3 Water Waves simulations - an alternative consideration

The mechanism behind this effect is remarkably simple. It was invented long time ago by observation while experimenting with area sampling.

Area sampling is a very common algorithm in computer graphics. Considering a two-dimensional map, the value at (x, y) is affected by values surrounding position (x, y), such as (x+1,y), (x-1,y), (x,y+1) and (x,y-1). Our wave simulation actually works in three dimensions, but the principles is show here for 2D. Blurring a map is very simple. You'll need two maps: one containing the data you want to blur, and one for the resulting map. The algorithm (using five sample values) looks like this:

```
ResultMap[x, y] := (SourceMap[x, y] +
                    SourceMap[x+1, y] +
                    SourceMap[x-1, y] +
                    SourceMap[x, y+1] +
                    SourceMap[x, y-1] ) DIV 5
```

For 3D waves while calculating our wave simulation, we have to know how the waves looked like one moment earlier. The resulting map becomes the source map for the next frame.

This is the actual wave simulation algorithm:

```
ResultMap[x, y] := (( CurrentSourceMap[x+1,
y] +
                    CurrentSourceMap[x-1, y] +
                    CurrentSourceMap[x, y+1] +
          CurrentSourceMap[x, y-1] ) DIV 2 ) -
                       PreviousResultMap[x, y]
```

As presented so far, Gerstner waves are rather limited because they are a single sine wave horizontally and vertically. However, this can be generalized to a more complex profile by summing a set of sine waves. One picks a set of wavevectors ki, amplitudes A_i, frequencies ω_i, and phases φ_i, for i = 1, . . . ,N, to get the expressions:

$$\mathbf{x} = \mathbf{x}_0 - \sum_{i=1}^{N} (\mathbf{k}_i / k_i) A_i \sin(\mathbf{k}_i \cdot \mathbf{x}_0 - \omega_i t + \phi_i)$$

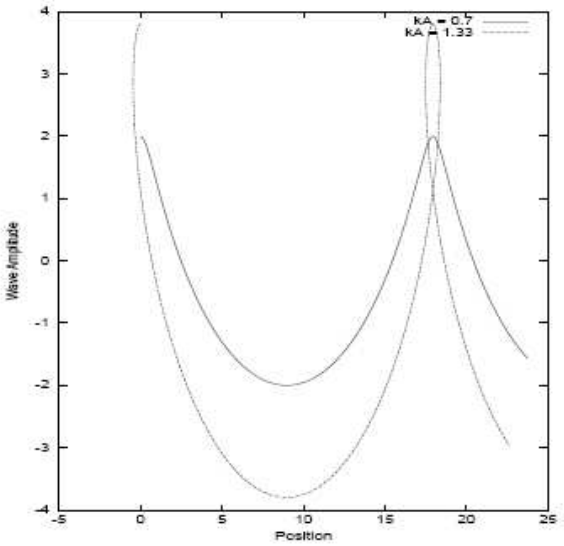

Fig. 1 Profiles of two single-mode Gerstner waves, with different relative amplitudes and wavelengths

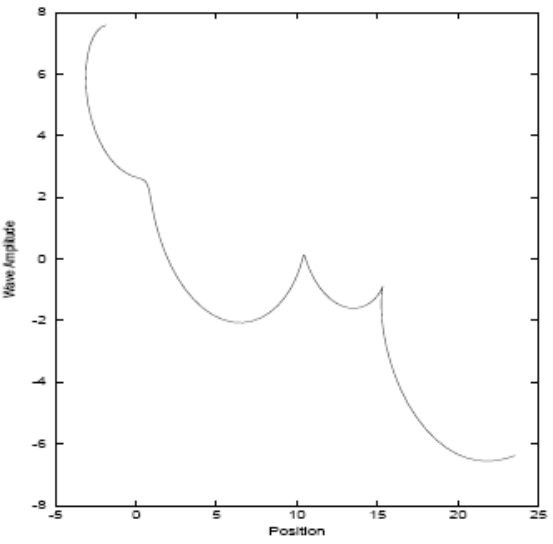

Fig. 2 Profile of a 3-mode Gerstner wave

4 Conclusions

One of the benefits of our approach is its flexibility: the use of trochoids enables to model a wide range of ocean surfaces, from calm to stormy seas. Since the displacement of a sample point is computed as a sum of wave contributions, adding extra effect such as ship

waves is easy. The camera position can be arbitrarily chosen without changing the amount of computation nor the image quality (no cyclicity will appear). As a consequence, the user can interactively fly over an unbounded ocean surface, which makes the method promising for video simulator applications. Lastly, the quality/cost ratio is tunable, so higher-quality images can be computed using the same model. All computations are exclusively concentrated onto the visible part of the ocean, which yields real-time performance with a relatively good image quality, even in our non-optimized implementation.

Concerning future work, we plan to implement the optimizations, to include other kinds of waves such as ripples or ship waves, and to study how waves reflection and refraction could be introduced. We also plan to simulate the glittering of the ocean waves near the horizon due to the multiplicity of normals.

5 References

[1] CRAWFORD, JR, F. 1977. Waves. McGraw-Hill.

[2] DEBUNNE, G., DESBRUN, M., CANI, M.-P., AND BARR, A. 2001. Dynamic realtime deformations unsing space and time adaptive sampling. In SIGGRAPH'01 Conference Proceedings, Addison Wesley, Annual Conference Series, ACM SIGGRAPH, Los Angeles, CA.

[3] EBERT, D., MUSGRAVE, K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 1994. Texturing and Modeling: A Procedural Approach. Academic Press, Oct.

[4] FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. Proceedings of SIGGRAPH 2001 (August), 23–30.

[5] FOSTER, N., AND METAXAS, D. 1996. Realistic animation of liquids. In Graphics Interface '96, W. A. Davis and R. Bartels, Eds., 204–212.

[6] FOURNIER, A., AND REEVES, W. T. 1986. A simple model of ocean waves. In

[7] GAMITO, M., AND MUSGRAVE, K. 2000. An accurate model of wave refraction over shallow water. In Eurographics Workshop on Computer Animation and Simulation, 155–171.

[8] GONZATO, J.-C., AND SAËC, B. L. 2000. On modelling and rendering ocean scenes. The Journal of Visualization and Computer Animation 11, 1, 27–37.

[9] HASSELMANN, D. E., M.DUNCKEL, AND EWING, J. A. 1980. Directional wave spectra observed during jonswap 1973. J. Phys. Oceanogr. 10 (August), 1264–1280.

[10] KASS, M., AND MILLER, G. 1990. Rapid, stable fluid dynamics for computer graphics. In Computer Graphics (SIGGRAPH '90 Proceedings), F. Baskett, Ed., vol. 24, 49–57.

[11] KINSMAN, B. 1984. Wind Waves, Their Generation and Propagation on the Ocean Surface. Dover Publication.

[12] MASTIN, G. A., WATTERBERG, P. A., AND MAREDA, J. F. 1987. Fourier synthesis of ocean scenes. IEEE Computer Graphics and Applications 7, 3 (Mar.), 16–23.

[13] NEYRET, F., AND PRAIZELIN, N. 2001. Phenomenological simulation of brooks. In Eurographics Workshop on Computer Animation and Simulation, Springer, Eurographics, 53–64.

[14] HINSINGER, D. NEYRET, F., CANI, M.P., Animation of Ocean Waves, iMAGIS-GRAVIR, joint research project of CNRS, INPG,INRIA, UJFInteractive

[15] PERLIN, K. 1985. An image synthesizer. In Computer Graphics (SIGGRAPH '85 Proceedings), B. A. Barsky, Ed., vol. 19(3), 287–296.

[16] PFISTER, H., ZWICKER, M., VAN BAAR, J., AND GROSS, M. 2000. Surfels: Surface elements as rendering primitives. Proceedings of SIGGRAPH 2000 (July), 335–342.

[17] STAMMINGER, M., AND DRETTAKIS, G. 2001. Interactive sampling and rendering for complex and procedural geometry. In Rendering Techniques 2001 (Proceedings of the Eurographics Workshop on Rendering 01), Springer Verlag, K. Myskowski and S. Gortler, Eds., 12th Eurographics workshop on Rendering, Eurographics, 151–162.

[18] TESSENDORF, J. 2004. Simulating ocean water. In Siggraph Course Notes, Addison-Wesley.

[19] THON, S., AND GHAZANFARPOUR, D. 2001. A semi-physical model of running waters. In Eurographics UK.

[20] LIGHTHILL, J. 1978. Waves in fluids. Cambridge University Press.

[21] B. ECKEL, Thinking in C++ 2nd Edition, Free Electronic Book Volume 1 & Volume 2, 2003.

[22] http://www.ogre3d.org.

[23] http://www.gamedev.net.

[24] TANENBAUM, A., S., Computer Networks, Fourth Edition, Vrije Universiteit, Prentice Hall, Amsterdam, 2002.

[25] FOSNER, R., Real-Time Shader Programming, Elsevier, Morgan Kaufmann

