SimShip

Edouard Halbert, 2025

SimShip is a real-time simulation of a ship sailing across an infinite ocean.

Real-time is arbitrarily defined as the lower limit of 150 frames per second in full screen at 2560×1440 (WQHD) on a computer equipped with an NVIDIA RTX 4070 (Intel Core i9-13900K microprocessor).

The project follows, years later, on a limited approach published on YouTube: (Basic simulation of ship motion in regular waves). At the time, it was a simple demonstrator focused on hydro-static calculations. Since April 2012, the video has been viewed 133,000 times.

The simulation uses several domains: ocean, sky, sun, lighting, clouds, mist, fog, camera, ship, terrain, sounds, smoke, interfaces, and ship autopilot.

Before beginning, the SimShip world (the scene) is geo referenced (longitude and latitude). A unit of 1 in the simulation is equal to 1 meter. This means that all objects in the scene (waves, ship, buoys, terrain, etc.) have the same coordinate system. There is no curvature of the Earth.

The simulation was programmed in C++ (ISO C++20 standard) using the free Microsoft Visual Studio Community 2022 editor. Numerous open-source code libraries were used, including OpenGL (4.3), GLFW, Glad, GLM, Assimp, libigl, ImGui, nanovg, stb, Eigen, FFTW3, Pugixml, Nova, Clipper, and OpenAL.

The purpose of this document is to explain the techniques used in the code.

OCEAN

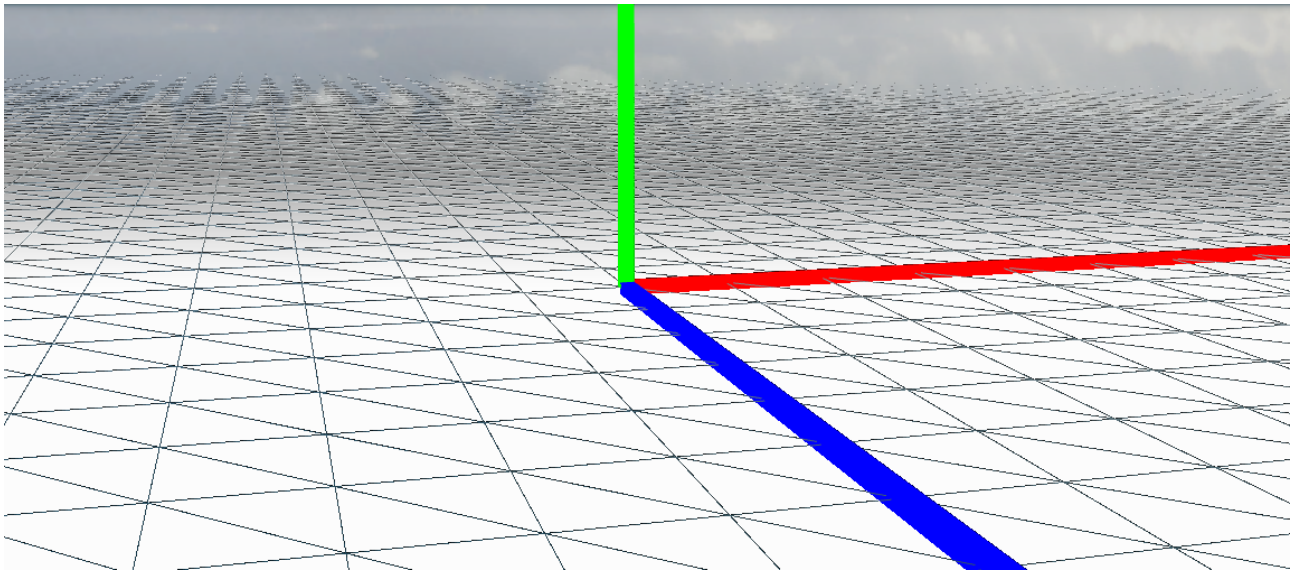
In this area of simulation, the most important paper is that of Jerry Tessendorf (coursenotes2004.pdf). This is a 26-page article explaining the technique used to animate an ocean in real time. The paper is remarkable, but reading it—very important for understanding the simulation and the techniques used—does not allow you to move directly to coding.

For this, I relied heavily on the Asylum Darth blog which offers a video, the code and detailed explanations of the implementation of its ocean based on Jerry Tessendorf's model. Blog Ocean rendering, video on YouTube OpenGL Ocean Rendering and code on GitHub https://github.com/asylum2010/Asylum_Tutorials.

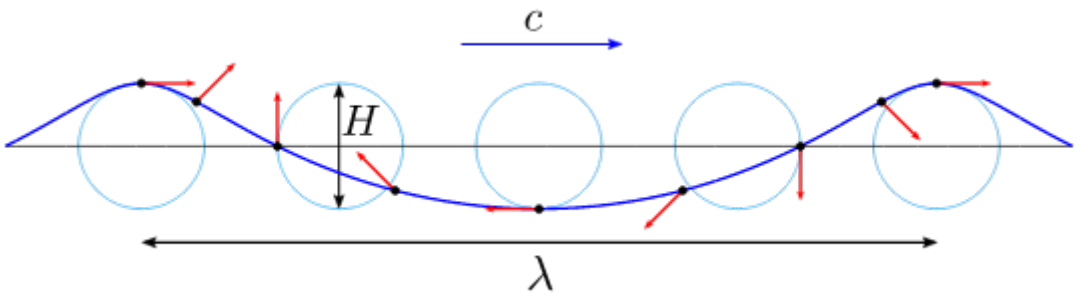
The ocean is modeled from a single patch. Then 40,000 identical patches (200×200) are displayed on the screen to give the impression of a large ocean. These patches move with the camera, giving the impression of an infinite ocean.

The patch is a 100 m x 100 m (1 hectare) grid with 256×256 squares, each composed of 2 triangles. There are therefore 131,072 triangles per ocean patch (256×256×2). Each point on the grid has 3 coordinates: x, y, z. To place it in the usual OpenGL reference frame, y is the height and xz is the

water plane. Without any waves, that is to say when the water is at rest, y is 0. y is positive for a crest and negative for a trough.

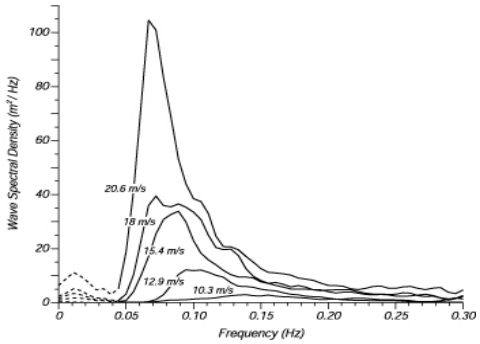


The basic wave model is the Gerstner trochoidal wave. In fluid dynamics, the trochoidal wave is an exact solution to the Euler equations, which are a set of nonlinear partial differential equations that describe the motion of a perfect fluid, i.e., an ideal fluid without viscosity or thermal conductivity. They express the conservation of mass and momentum under simplified assumptions. Discovered in 1802 by Baron von Gerstner, this wave model describes periodic gravity waves propagating on the surface of an incompressible fluid of infinite depth, in a steady state. The free surface of the flow is a cycloid (or trochoid, to use Gerstner's term).



The drawing above shows a wave, which we'll call a base wave. It is defined by a frequency (more or less fast), a time lag (the wave starts somewhere on the blue line), and an amplitude (the height between the bottom and top of the blue line). On the surface, the Gerstner wave is quite simple. At its core, it's a sinusoid. To add complexity, we can add sinusoids that have different frequencies, different phases, and different amplitudes. To achieve a realistic appearance, the ocean patch is composed of many base waves.

All waves in SimShip are the sum of many basic waves whose frequency, phase, and amplitude are random and determined at the beginning of the simulation. They are then updated every frame. Thus, each patch contains 512×512 basic waves defined by their frequency, phase, and amplitude.

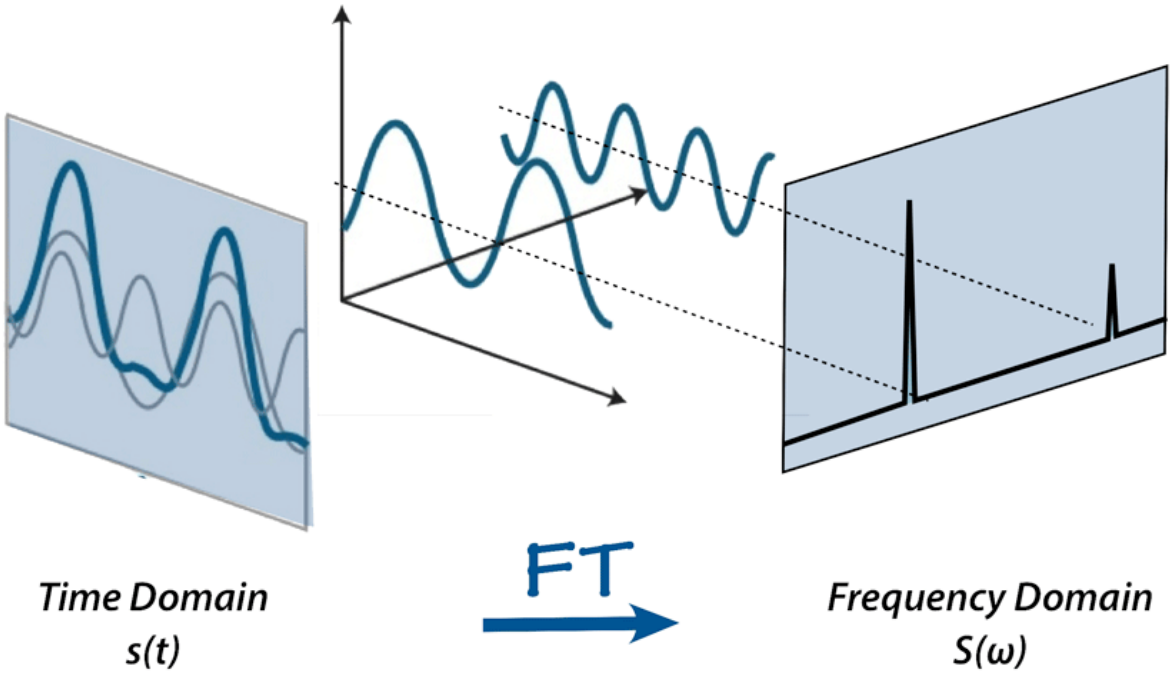


The frequencies are not chosen entirely at random. The frequency spectrum of these basic waves is based on an analysis of real ocean waves. There are several spectra that differ depending on wind conditions and fetch (distance the wind blows across the water). The spectrum chosen is the Phillips spectrum. This is the one described in Jerry Tessendorf's article. It isn't necessarily the most realistic, but it's very pleasant to look at.

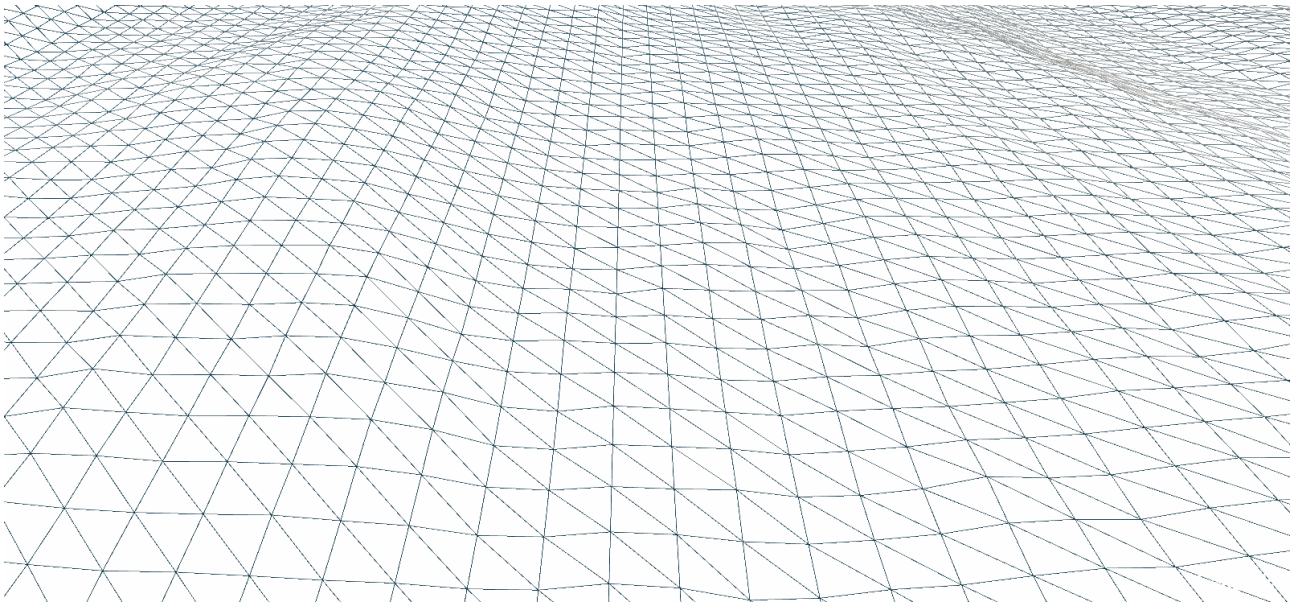
After extensive testing, it appears that no wave spectrum adequately models waves for all wind speeds ranging from 1 kt (calm) to 60 kt (storm). This is possible unless the number of patches is multiplied. Thus, some advanced simulations use four patches to model the entire patch, with these four patches having very different frequencies. One patch simulates long waves, another shorter waves, and so on. The problem is that multiplying the number of patches reduces the frame rate, given that modeling the boat also requires computational time. For now, SimShip only includes one spectrum for each patch.

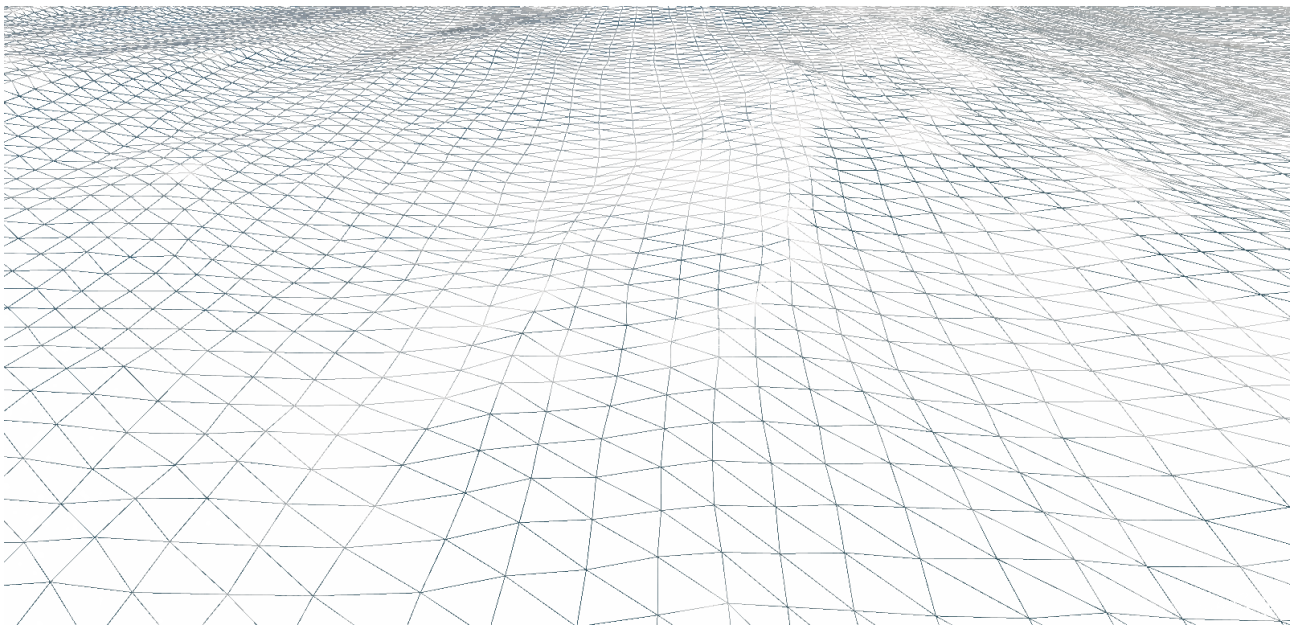
The Phillips spectrum has been slightly modified in this simulation to focus on winds from 1 kt to 20 kt, those most frequently encountered in reality. Beyond this, the waves are less representative of what they are in reality.

The patch is created at each screen refresh using the Fast Fourier Transform (FFT), an algorithm that efficiently converts from the frequency domain to the spatial (or time) domain. Without going into detail, this involves updating all 262,144 basic waves (512×512 trochoids) in the frequency domain at each frame using the Fourier Transform. The inverse Fourier transform (iFFT) allows the spatial ocean surface (what we see) to be reconstructed from the frequency spectrum (what changes in each image), which is faster than individual wave summation.



At the end of this process, we obtain a small displacement dx, dy, dz for each point of the grid (256 points x 256 points for a patch of 100 m x 100 m). The height displacement dy simulates the crests and troughs of the waves while the displacements dx and dz simulate the horizontal movement of the water.



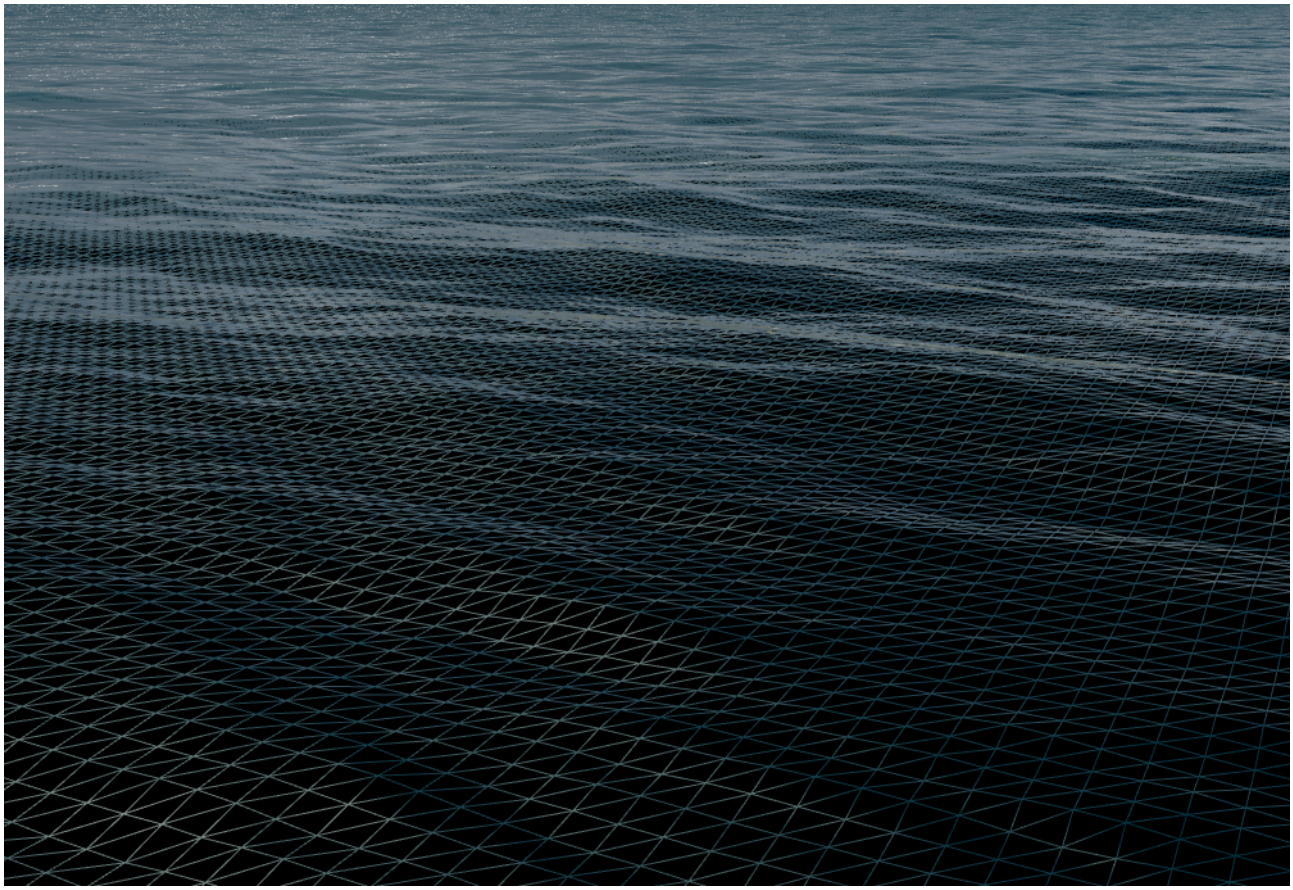


Note that the Fourier Transform can be described as fast if it is applied to a finite number of signal samples, with discrete (discontinuous) frequencies. It uses a trick of calculating only half of the grid (with a diagonal separating the two halves), the other half being deduced as a mirror image of the first. This has an impact on the ocean patch, namely that the left edge of the patch is equal to the right edge and the front edge is equal to the back edge. Thus, when one patch is placed next to another, since it is the same patch and the edges are two by two equal, the patches are contiguous. However, they draw the same waves on the screen, forming a repetitive pattern, especially if the waves are large and foamy. This is the limitation of this approach.

The FFT performs a large number of calculations, but these can be performed simultaneously, i.e., in parallel. No need to wait for one calculation to complete before doing the next. Whereas a computer's microprocessor performs operations one by one, modern graphics cards have many small processors that can perform operations in parallel. For example, an NVIDIA RTX 4070 graphics card has 5888 CUDA cores, which are the main parallel computing units for processing floating-point data. Therefore, all the calculations in the patch are performed by the graphics card. This requires a special programming language, GLSL (OpenGL Shading Language) in the case of SimShip. GLSL is a high-level programming language designed to write "shaders" in the OpenGL environment. Shaders are small programs executed directly on the GPU (Graphics Processing Unit, the graphics card) and allow precise and efficient control of graphics rendering: lighting, shadows, textures, advanced visual effects, etc. Vertex shaders process each vertex (point) of a 3D model (a mesh) to transform it, from the 3D space of the model to the 3D space of the scene and then to the 2D surface of the screen. Fragment shaders (or pixel shaders) perform for each fragment (potential pixel), the calculation of color, transparency, texture, light, etc. This is what gives the final look to each pixel of the image. Finally, there are Geometry shaders, Tessellation shaders and Compute shaders which are additional steps allowing more advanced manipulations of geometry or general calculation, but less common than vertex and fragment shaders.

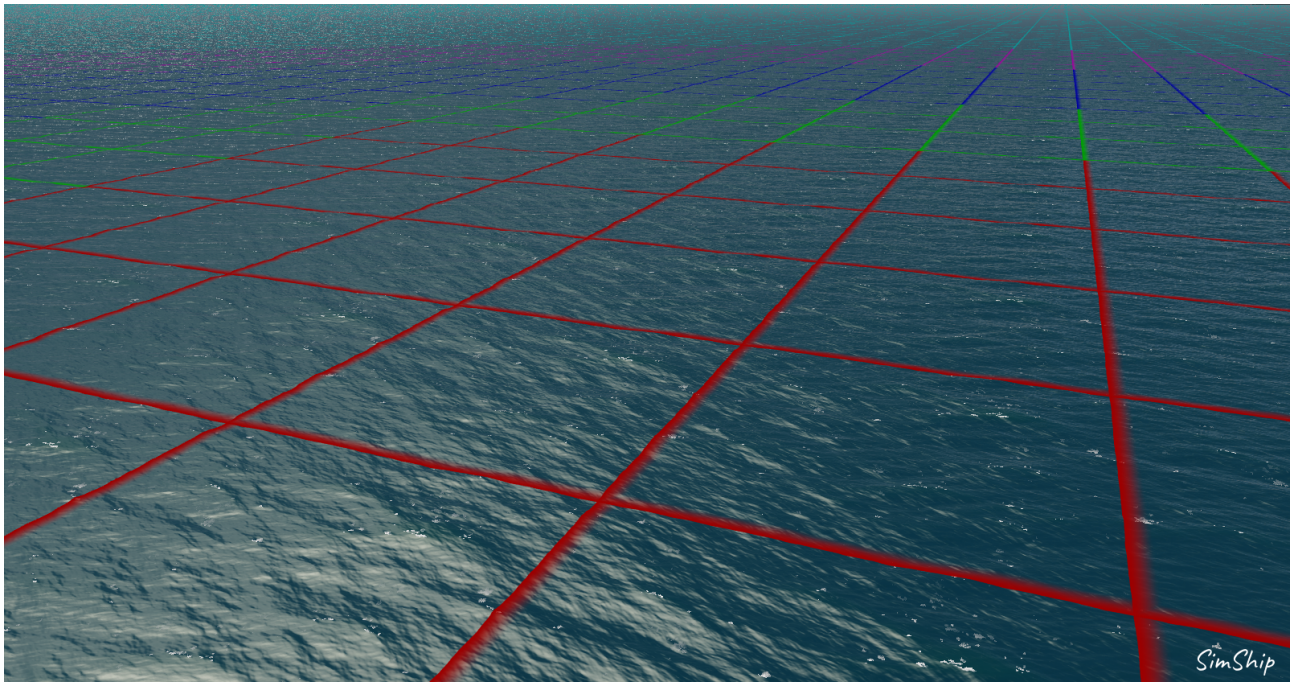
SimShip's ocean patch makes extensive use of Compute shaders. We pass the trochoid data to the graphics card. We also pass the calculation code (the Compute shaders), that is, the algorithm for updating these sinusoids, and the graphics card calculates all of this in parallel. At the end, we obtain the result (the displacements dx, dy, dz) on an image where each pixel represents these

values. We also obtain the gradient between the points of the patch's grid, which allows us to determine whether foam should be drawn where the waves are steep.



Once a patch has been obtained, 40,000 instances of this patch are generated by OpenGL and placed around the camera's position. Only patches visible from the camera's view are drawn (frustum culling). Frustum culling is an optimization technique in 3D computer graphics that consists of not drawing objects located outside the camera's cone of vision, called the frustum.

The patches that appear on the screen have a different level of detail (LOD) depending on their distance from the camera. The LOD is said to be adaptive because it is updated with each frame. The very close patches are drawn without loss of information (so with a grid of 256 points x 256 points (representing 100 m x 100 m) but the very distant ones have only 16 points x 16 points (still for 100 m x 100 m) and in between, there are grids of 128×128, 64×64 and 32×32.



For patches where the ship and its wake are on it, a special rendering is performed for the wake (explained later).

Here's the ocean geometry. We have moving points that represent the waves. For each image, 512×512 sinusoids are updated by the graphics card, resulting in the displacements of the grid points (dx, dy, dz) and the gradients that provide information about the breaking waves. Then we take this patch and duplicate it with varying degrees of definition (LOD) to obtain an ocean visible for several dozen kilometers (to the horizon).

Next, we need to properly light the waves to give them the appearance of water. This is also done with shaders. Vertex shaders take the points and position them correctly on the screen according to the camera's view. Then fragment shaders create the color and transparency that will be applied to each point. And in the end, we have an animated ocean, at over 300 frames per second, full screen. The graphics card does all the work.

Wave lighting is complex. It depends on the position of the sun and its light. The water has many reflections, both from the sun and from the ship. Foam is also added according to the gradient of the points seen previously to represent the breaking of the waves. Transparency also changes depending on the viewing angle. The more vertically you look, the more transparent the water becomes. The physical model used for ocean lighting is a BRDF (Bidirectional Reflectance Distribution Function) model associated with the Ward model and Fresnel reflection.

A word about foam, the calculation of which is based on the Jacobians of the water surface. A Jacobian is the matrix of partial derivatives of a vector function that describes how this function changes locally. To estimate wave slope in a shader, the Jacobian is applied to the ocean surface displacement function: it allows the calculation of the spatial derivatives of wave height (the local variations of the surface). These derivatives provide the slope at each point, which is used to determine where to place the foam, for example, in areas with steep slopes or breaking waves. Thus, the Jacobian transforms the surface deformation information into a slope vector that can be used to

dynamically represent waves in the shader. This is sufficient for a simple representation of the ocean because limiting ourselves to this technique would miss the breaking water. This would be feasible with particles.

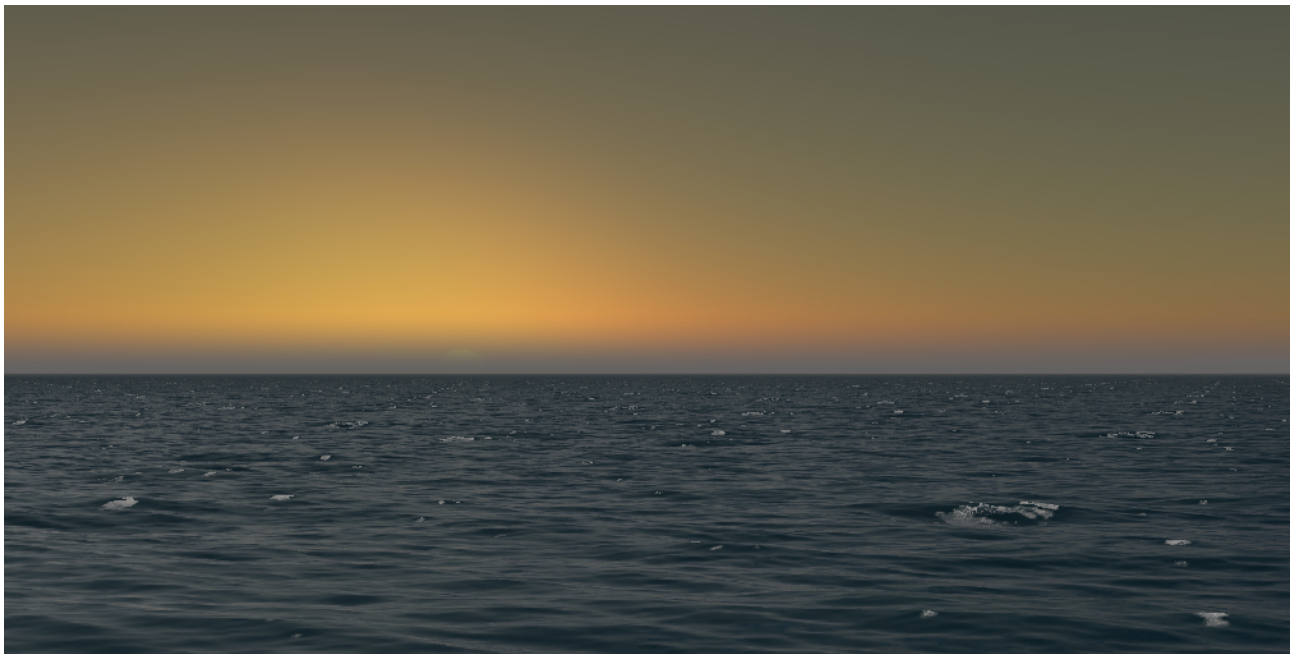
Finally, the ocean can have a cinematic quality but be physically unrealistic. It is therefore important to verify that the wave period and significant height (the upper third) are similar to data observed in real-life situations. SimShip allows spectral analysis of waves. To do this, it takes the successive water heights at each image at the point (0, 0) of the scene (where a red-green-blue axis can appear). With the sequence of water heights, it determines – again with FFTs – the distribution of frequencies and their density, from which it gives the characteristics of the waves.

SKY

The sky is also a much more complex area than one might initially imagine.

First, there is the light source represented by the sun. Its position in the sky is determined by the location of the view (the camera), the date, and the time. In SimShip, the initial camera position is that of the island of Houat in southern Brittany, France. Based on this position, the current date and actual (solar) time, the azimuth (horizontal direction), and the elevation (angle relative to the horizon) of the sun are calculated.

Then, these two angles will give the color of the light, orange when the sun is low on the horizon and almost white when the sun is at its zenith.



The sky is then calculated based on all the light effects generated by the diffusion, reflection, refraction and absorption of sunlight in the Earth's atmosphere. The calculation code is based on that of Eric Bruneton (precomputed_atmospheric_scattering). Its atmospheric model is an advanced method for real-time rendering of the Earth's atmosphere, taking into account the complex interaction of light with atmospheric constituents such as air molecules, aerosol particles, and ozone. This model notably simulates the multiple scattering of light, integrating Rayleigh and Mie scattering effects, to realistically render the colors of the sky, including diurnal variations (day,

twilight) and aerosol effects. Here again, once the two angles (azimuth and elevation) have been determined, the graphics card is called upon using a compute shader. The result is an image the size of the SimShip window that represents the sky with a horizon and the color gradient.

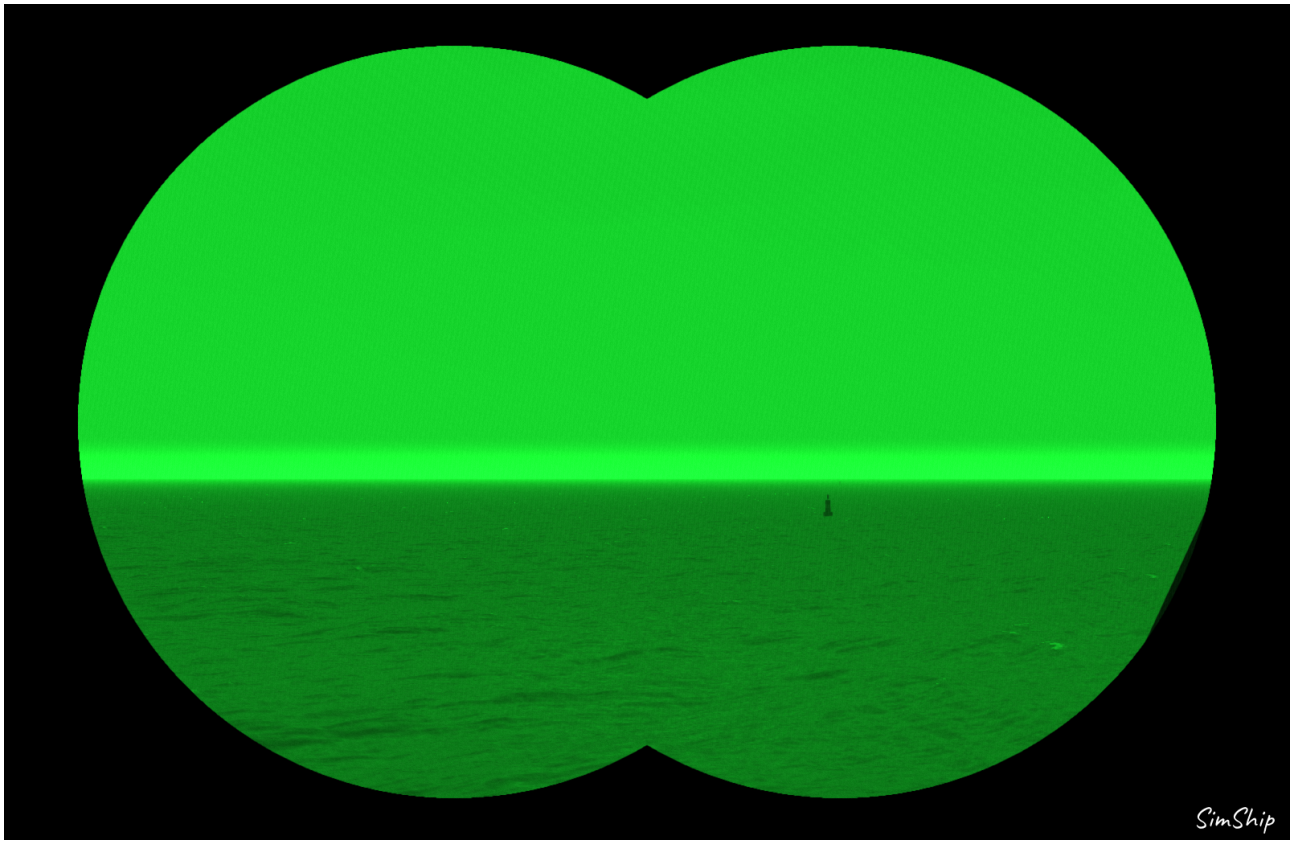
After the sun position and the sky, the clouds need to be composed. There are two generic ways to approach this. The simplest and least computationally expensive approach is to draw random patterns that resemble clouds on an image and then apply this image above the camera. This works, but the clouds lack "thickness," and it shows. The other, much more complex and computationally expensive technique is to model clouds in 3D, called volumetric clouds. Some code examples can be found on the Internet. The source code used for SimShip is by Federico Vaccaro (TerrainEngine-OpenGL). He himself drew inspiration from many sources (see his website). These clouds are volumetric, they move in the direction of the wind and evolve in a very realistic way. Without going into details, the calculation uses 3-dimensional images (3D textures) and the ray-marching technique which proceeds by advancing a ray iteratively in small segments along its trajectory. It is very expensive in terms of calculation time... but it is rather successful. Here again, it is the graphics card that does all the work.

Finally, mist and fog are exponential models that correspond to a light extinction that decreases exponentially with the distance traveled in the suspended mass of water (microscopic droplets). Unlike fog, mist is contained around the horizon.

So to summarize this part, we calculate the position of the sun, we create the atmosphere with its color gradients according to the time of day, we apply volumetric clouds and then mist or fog if necessary. And we also apply an exposure to the whole thing to simulate night, dawn and dusk.

CAMERA

Before discussing the ship, which is a big piece, a few words should be said about the camera. A good simulation requires a good camera. It's what allows you to move around the scene. Thus, the camera offers several modes: an orbital mode where you rotate around the ship, a free mode (first-person shooter), and views from the bridge or somewhere on the deck, depending on the ship. Then, movements must be done very gradually with the mouse or keyboard. There are several interpolation modes. There is also the possibility of zooming as if you had binoculars. Finally, the Camera class provides all the matrices and vectors necessary for the various simulation calculations.



SHIP

Here again, this is a fairly complex part that uses the microprocessor for calculations (CPU, Central Processing Unit) and relatively little use of the graphics card.

The ship must be visible on the screen (preferably attractive) and move in response to the waves (preferably realistic). It's worth mentioning right away that these two qualities are antagonistic: the more beautiful a ship looks on screen, the more detailed it is (its 3D model is composed of hundreds of thousands of faces), whereas for calculating buoyancy, the number of faces (triangles) is a handicap. The fewer there are (up to a certain limit), the faster the calculations. Note that the entire on-screen display of a 3D scene relies on triangles to be displayed, onto which an image called a texture is applied. Thus, the word "triangle" is preferred to "face" (a face can have 4 or more vertices).

The ship can have animated parts such as one or more radars, one or more rudders, and one or more propellers. Therefore, the simulation must have as many 3D files as there are parts to be animated. For example, a ship has a radar, a rudder, and a propeller. That makes 4 files: one file for the ship without its radar, rudder, and propeller, one file for its rotating radar, one file for its pivoting rudder, and one file for its rotating propeller. But there is also—and this is perhaps one of the specificities of SimShip—a file solely for the hull (just the hull). This file is used to calculate its buoyancy. Having a separate file from the model to be displayed is necessary because buoyancy calculations only concern the hull. Then, there is an optimal number of triangles for the calculations. For example, if a tanker has a large section of hull in the middle that does not show any evolution in shape, the 3D model will have few triangles there. With the power of today's computers, 2,000 to 8,000 triangles seem like a good compromise. More triangles and the ship's movement in the waves

is perfect, but the simulation is jerky. Fewer triangles and it's the ship's movement that seems jerky (when one of its large triangles enters or leaves the water).

There are therefore 3D models to display for each ship (*.glb or *.gltf files) and one 3D model (*.obj file without a *.mtl material file) for calculations.

For the models to be displayed, the glTF (GL Transmission Format) format was preferred. It is an open and lightweight format intended for storing and exchanging 3D scenes and models. It uses JSON to describe the scene structure, materials, animations, and geometries, facilitating fast loading and good interoperability. The GLB format is the compact binary version of glTF, grouping all JSON, binary, and image data into a single file, simplifying the management and downloading of 3D models.

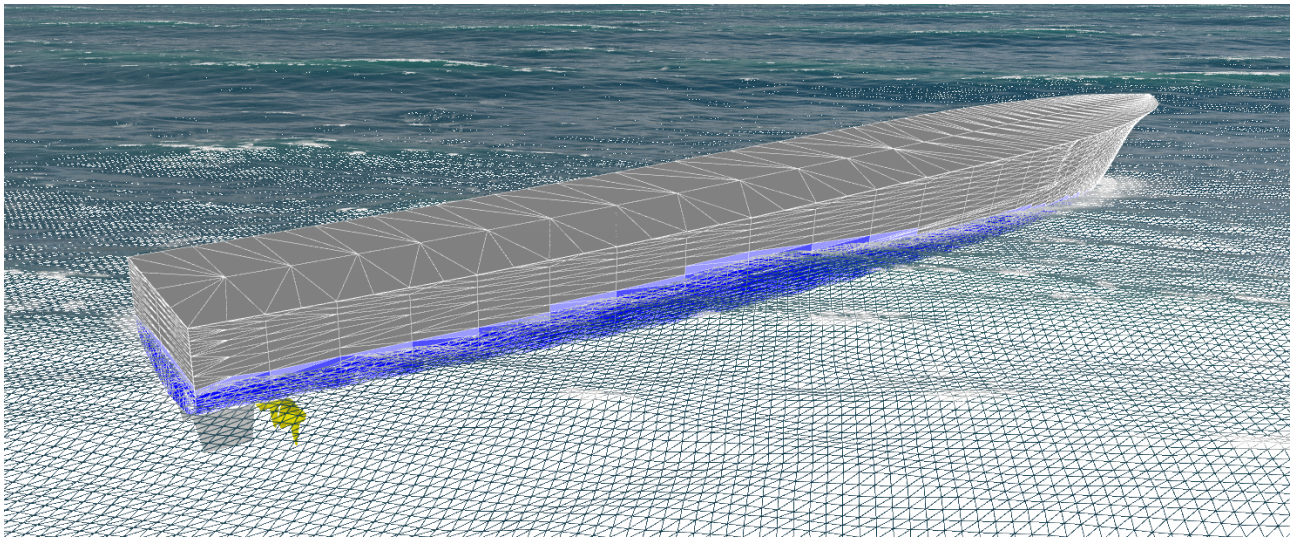
When loading the ship into the simulation, SimShip opens the ship's *.obj file to determine all the ship's vertices and triangles. The obj format is a standard text file format used to describe the geometry of 3D models. SimShip uses the minimalist version of this file type. It stores only vertex coordinates and faces defined by vertex indices. No additional texture or material definition files, which are possible with this file format, are required.

At each screen refresh, the ocean is recalculated to give the dx, dy, and dz of all points in the patch. This patch can be repeated because the edges are contiguous, meaning that even if a ship appears to be on two patches, it is actually only on the single patch.

At each screen refresh, all the vertices of the ship's triangles are evaluated to determine whether they are underwater or above water. This determines the list of triangles on the ship's hull that are submerged, emerged, or partially submerged. For each submerged (or partially submerged) triangle, the hydrostatic pressure force it is subjected to is determined, knowing the surface area of the triangle and the height of the water column above it. By summing all these hydrostatic forces, we determine the buoyancy force and the center of the hull volume (the center of flotation where the buoyancy force is applied). When we combine the force of gravity, known from the ship's mass and the position of the center of gravity, with the buoyancy force, we obtain a new force that translates the ship's vertical displacement when related to its inertia. The same applies to roll and pitch, associated with their respective transverse and longitudinal inertia.

A significant complication arises when calculating the water heights above the vertices. If the patch always had fixed points, without dx and dz displacements, it would be easy to interpolate because the patch grid is regular. When we add the dx and dz displacements to the grid points, the grid becomes irregular. Classical interpolation is impossible because we don't know which triangle to interpolate under. Several iterations are then required to determine which water triangle should be interpolated. Generally, 5 to 6 iterations are required per hull vertex. However, the hull often has between 2,000 and 8,000 vertices (to be done 5 to 6 times for each image).

Also, note that the intersection of the ship's triangles with the intersection of the sea triangles is of little interest. This is very time-consuming and adds little value, even if it is normally physically accurate. The simplifying trick is to assign a coefficient to the hydro-static pressure of a partially submerged triangle based on the number of submerged points. This approximation is smoothed out by the high number of hull triangles.

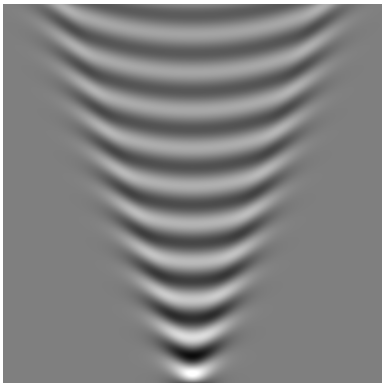


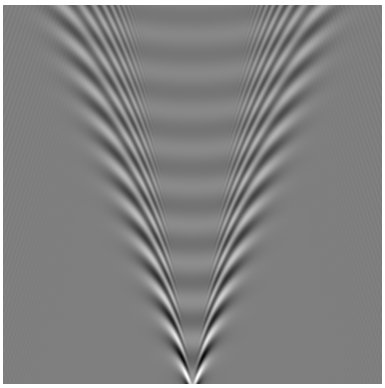
In addition to these two essential forces (Archimedes and gravity), there is the force produced by the engine-driven propeller, the force produced by the bow thruster (if applicable), the rudder forces (lateral thrust and resistance to forward movement), the viscosity force depending on the ship's speed and its wetted surface area, the wave resistance force, which increases rapidly with the ship's speed, and the centrifugal force, which causes the ship to tilt into a turn. Most of these forces do not apply to the center of the ship. Thus, ships have a pivot point located most of the time toward the front of the ship when moving forward and toward the rear when moving backward. Note that there are also two other forces modeled, related to the action of the wind: a force that causes the ship to drift downwind, and a rotational force that eventually brings the ship to a crosswind position.

In total, all these forces give the ship six degrees of freedom: three translations (x, y, z) (Surge, Sway, Heave) and three rotations around the x, y, and z axes (Roll, Yaw, Pitch).

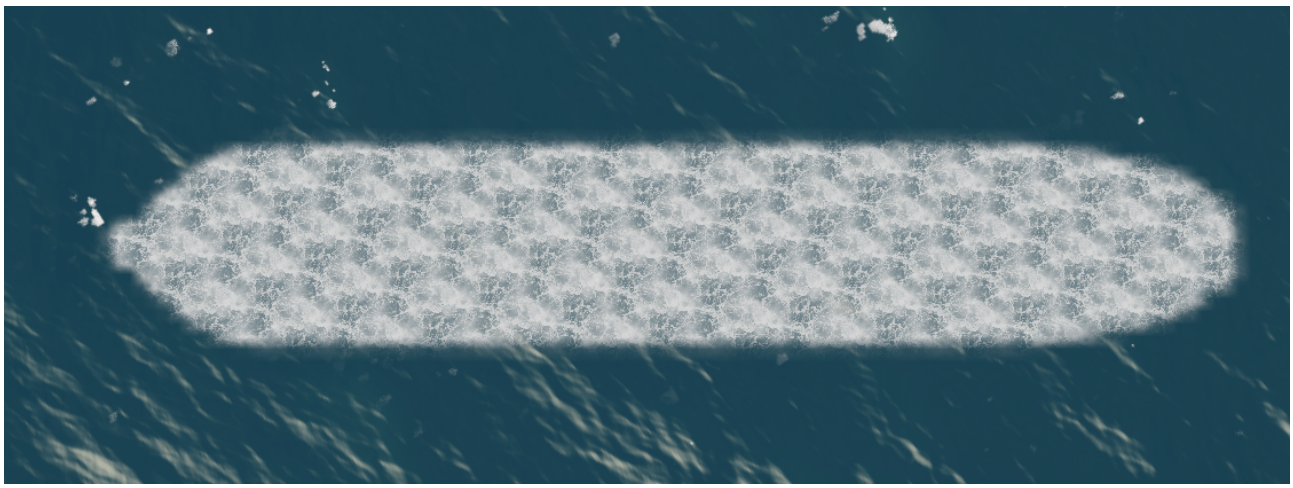
Finally, a ship on the water creates waves, foam around it and in its wake.

The waves created by the ship have been known for a long time. Wake waves are called Kelvin waves (Lord Kelvin), and they were described and explained by him in 1887. They can be expressed quite easily in equations. The only major problem is that these equations are not solvable for real-time simulation. To obtain somewhat realistic waves, it takes several seconds/minutes on a fast personal computer. There are some code examples on the internet, but they are valid under certain conditions (such as the Froude number). The solution chosen for SimShip is based on the application of pre-calculated images based on the Froude number (which is the speed of the ship related to its length). 100 images are thus calculated in high definition (for a calculation time of around half an hour) for a Froude number ranging from 0.01 (very slow ship) to 1.00 (very fast ship). Most SimShip ships have Froude numbers ranging from 0.0 at a standstill to 0.41 at maximum speed. Only one boat, a 43-foot launch, has a Froude number of 0.95.

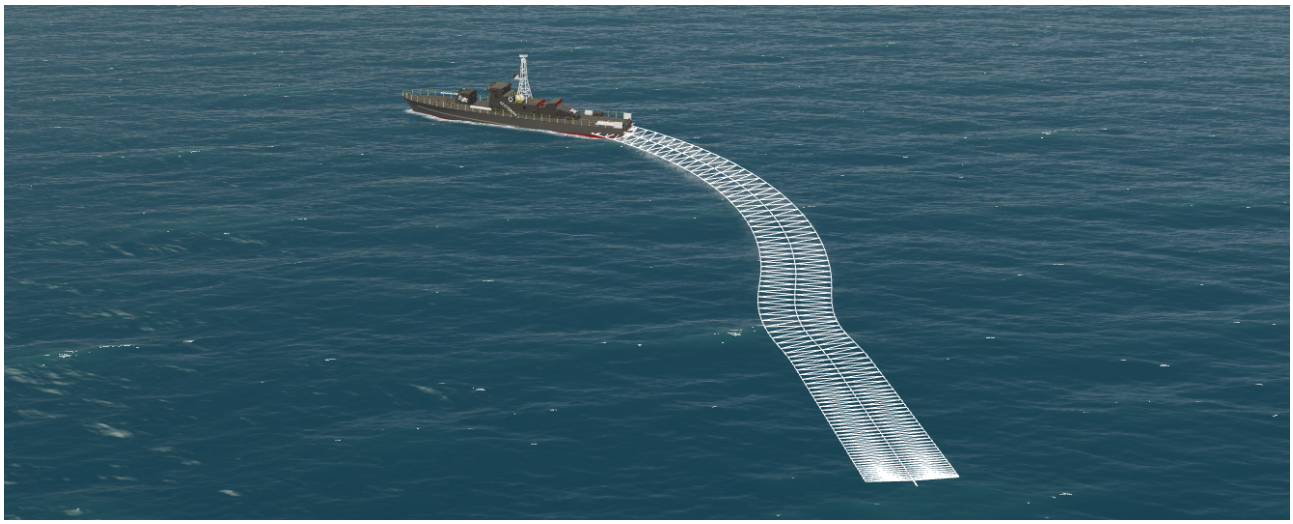




Foam and wake are modeled in two different ways. For foam around the ship, a hull contour is created when the ship is loaded into the simulation. This is the intersection of all hull points with the water (xz plane = 0). Then, the intersections are ordered and a closed contour is produced, a contour that is slightly widened in a uniform (non-scalar) way to extend beyond the ship on the outside. This contour is used to create a foam image. This image is applied to the ship's position at each frame.



For the wake foam, the technique used is different. Given the possible computational costs, the technique used is a bit unusual. Every x images (let's say 100), a point - located towards the rear of the ship - is added to a list. This list of points is used to create a polygon made up of triangles that represents the ship's wake from above. Then this polygon is used to create an image with foam, foam that will be more transparent depending on the time the point in this triangle was recorded. The older the point, the more the foam disappears. A short diagram is worth a long speech:

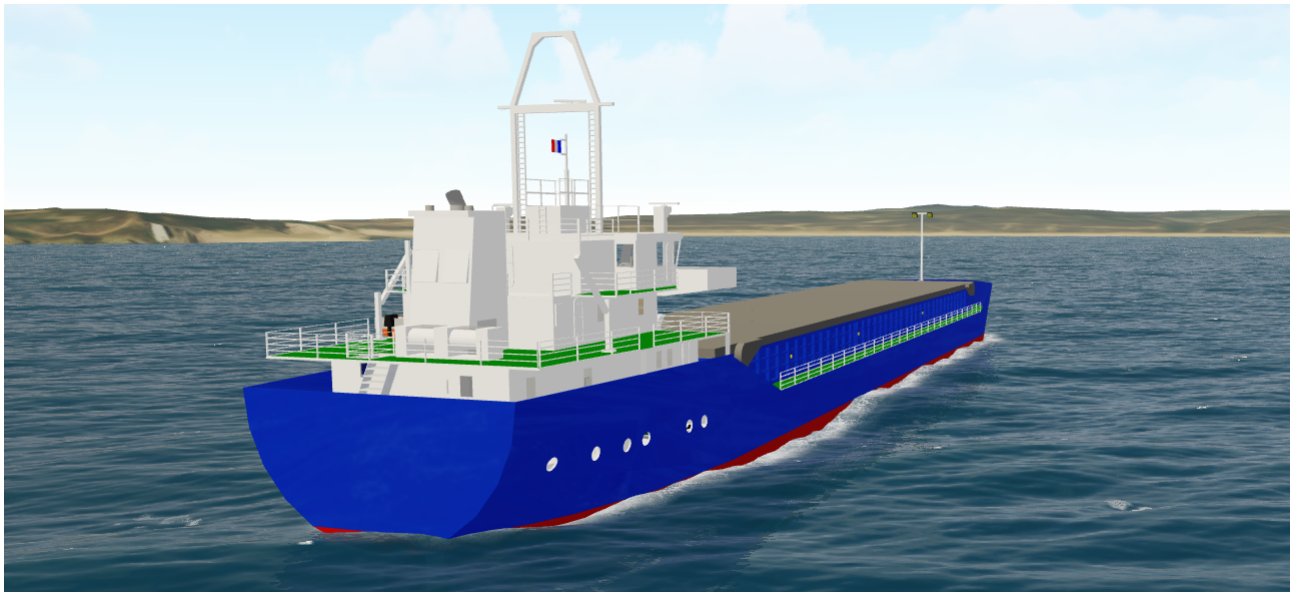


Finally, the ship is lit with the color of the sun and its position in the sky. The lighting model is a complex, physically realistic model. In addition to the animations of the radars, propellers, and rudders, there is also the smoke produced by the ship's engine.

The smoke is a particle system using the graphics card. Each smoke particle is emitted from one of the ship's smokestacks (position, speed, lifetime, and color). 10,000 particles are calculated for each frame based on the wind and its lifetime and displayed to simulate the smoke.

The foam on the ship's bow comes from the same particle system. 40,000 particles are emitted from a line bordering the front of the ship (contour). Their speed depends on the ship's pitch, especially when it falls heavily into the water, and of course, on the ship's speed.

Last but not least, a word about ships. SimShip currently offers 10. They represent a range of vessels, from small to large. Only 3D models that can be downloaded for free from the Internet are offered (and with a permitting license). Then, for a model to be accepted into the simulation, several conditions must be met: 1) the model must be attractive, 2) it must contain a reasonable number of triangles, and 3) the model must be able to be worked on using the free Blender software.




BOGA

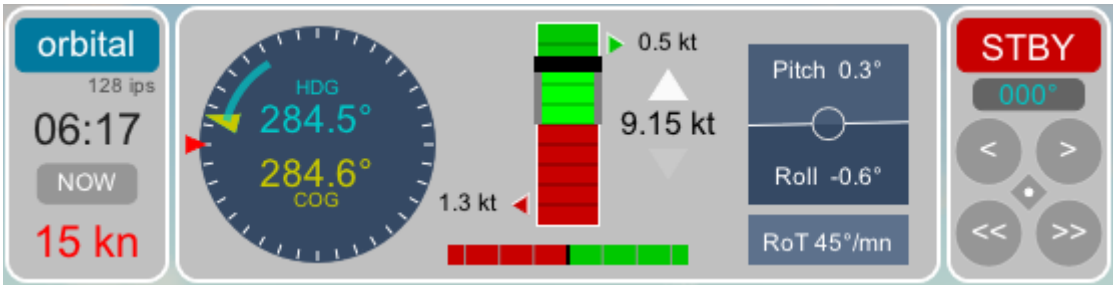

HORIZON


DOGA

CONTROL INTERFACE

The control interface includes several elements:
- general scene settings, wind parameters (direction and speed), time of day, sky, clouds, ocean (for which you can choose several colors, for example), and boat settings;

- actual vessel control (speed, heading over the water and bottom, rotation rates, rudder angles, engine and bow thruster power, roll and pitch rates);
- autopilot control.



AUTOPILOT

The vessel pivots with the wind and undergoes roll-induced rotation, so leaving the vessel with the rudder centered will not cause the vessel to steer straight. These effects must be compensated for by slight rudder input, and this compensation must be constant depending on sea and wind conditions. Thus, an autopilot that maintains a constant (or near-constant) heading has its place on board ships.

The pilot implementation uses a PID (Proportional-Integral-Derivative) controller to calculate the optimal rudder angle. The function first calculates the heading error. The proportional (P) component reacts directly to the current error. The integral (I) component accumulates the error over time, helping to correct persistent errors. The derivative (D) component anticipates future changes based on the rate of change of the error. The final rudder angle is the sum of these three components, limited to the maximum allowable angle.

In addition to these 3 parameters, there is possibly (option in SimShip) a dynamic adjustment which modifies the value of these 3 parameters depending on the conditions.

TERRAIN

To add an element of atmosphere, a terrain map was added. This is the island of Houat, located in southern Brittany. The island is 3.3 km long and 1.5 km at its widest point.

A separate SimShip program was created to create several terrain maps. This program identifies the island's outline within the global coastline contour file (coastlines-split-4326.zip) and then selects the correct terrain elevations from high-definition terrain data files produced by the French National Geographic Institute (IGN) to create a 3D model using satellite photos, also produced by the IGN. The resulting map forms a 3D model file that is read by SimShip.

MARKING

The buoys are also the subject of a dedicated program. Depending on the area, the list of beacons is extracted from a national catalog of French beacons. The beacon file produced is an XML file read by SimShip which positions a generic buoy (red, green, cardinal, isolated danger) in the scene.

SOUNDS

Only certain sounds are present: the sound of the ship's engine, the sound of its possible bow thruster, the sound of the ship's foghorn, and the sound of the seagulls. All sounds are spatialized. They diminish when they are far away and they shift from one side to the other depending on the view.

SCENE DISPLAY

So far, we have explained the calculations of the positions of the waves, the ship and all the objects like the sun, the clouds, the terrain, the buoys. We have also partially explained the lighting of the objects. So we have triangles composed of vertices displayed on the screen with for each a small piece of image (2D texture) and this composes the scene. In fact, not quite because the order of the display operations is important. An example: when we look at the waves from the bridge of the ship, that is to say through transparent windows, we must draw the sea first, then the ship and then the bridge because what is partly transparent is drawn last. The ocean is partly transparent as well as the windows of the ship, everything else is opaque. Some constituents of the scene are not 3D elements: this is the sky. Everything is calculated in 3D but in the end, the mini scene of the sky is an image that is projected on the screen. Finally, fog applies everywhere, to waves, terrain, buoys, and the ship. In fog, the front of the ship must be able to be "in" the fog.

In fact, the order of operations is a topic in itself. When looking underwater, that is, when the y component (height) of the camera is negative, we must no longer display the sky, clouds, etc., but we must completely change the atmosphere with reduced and rapidly decreasing visibility and suspended particles to create an atmosphere.

Thus, in this flow of combined operations, everything begins by calculating the motion of the ocean waves and the motion of the ship on these waves. Next, we create the texture (image) of the ship's reflection on the water. Then we draw objects other than the waves and the ship, such as the terrain, buoys, sky, and clouds. Then we draw either the ship then the ocean, or the ocean then the ship if we are on the bridge. We add the radar(s), the rudder(s), the propeller(s). When we draw the waves, we deform them according to the ship's wake waves, we add the foam around the ship, the foam in the wake, the reflection of the ship on the water, to which we add a slight floating effect like ripples on the water. We also add the foam on the breaking waves. The ocean fragment shader is the most complicated of the rendering shaders. All these displays are done in what we call a "multisample framebuffer," that is, in an off-screen, unaliased image (the lines are very thin and softened - antialiased).

Next, we apply the smoke particles and spray particles to the front of the ship.

After we have obtained this image of the scene, we still have a few operations to perform in a phase called post-processing before displaying it on the screen. Indeed (no pun intended), we add the localized haze around the horizon or fog, the underwater part, and the view through binoculars (black screen with two circles revealing a magnified scene).

Finally, we apply the image to the screen.

And we repeat all the positioning and lighting calculations, then recompose the scene. And this happens every 150 times per second.