Storm Breakers is an all-in-one Unity package designed for epic deep water simulations. Featuring a unique wave model, realistic breaking waves with visual, audio and physical effect, an extra-documented buoyancy physics adjustable for any gameplay, a scalable splash visual effect synchronized to procedural audio, and a dynamically textured water with realistic rendering.

This package is suitable for both simulations and arcade games, it allows for unique gameplay like surfing waves, surviving a storm and satisfying experiences.

About this document

This documentation is a fairly big one with its 140+ pages. This is because the Storm Breakers package has many features and this document explains them all in broad terms with many images so as to provide a good understanding and how to use them.

This document is designed to be quickly browsed through. Take a look at the whole document by reading the titles, the underlined text and looking at the image and their legend. You'll know about what is important to notice and all the existing features. Then you can come back to read thoroughly when you need more info.

To start using the Storm Breakers package, you may read the Getting Started section that is written as a tutorial. Some tutorial videos will be edited soon to improve the user experience.

This document is not designed to be printed. If you wish to, please ask for a printable version in the Discord support server.

Support

To get support for using Storm Breakers package, join the Discord support server or send an email at lysandre.creations@gmail.com

Sponsor

I, Lysandre, a French engineer passionate about boats, sea and waves, am currently the only developer of the whole package : the innovative ocean model, the writing of shaders, the physics, the visual effects, procedural audio and all the explanations/diagrams of this documentation.

Me sailing.

It took about a year to complete for the first release, it was mainly a job of passion done in full time mostly. I thank all my relatives for bringing me support during the developpement that otherwise would not have been possible.

I sell this package so as to earn back a bit of what it cost me to develop it, but the main goal is to share my vision of the ocean : an epic terrain of adventure ! There are not enough game mechanics based on the waves in the market, and I hope by providing you this package you will change this by making awesome games !

That being said, the quality of the whole package is the one I could provide being solo and still a Unity learner (this is one of my first Unity projects), there might be some unspotted bugs, some visual quality to be improved and so on. The package is still in active development and is meant to be updated in time for increasing quality and new features, this documentation tells about all the planned updates.

To upgrade this package to another level, any support from you will greatly help. Also another project is planned to make shallow water simulation with real plugging waves. This is even harder to do than it was to code Storm Breakers, so if you want real surfing waves consider supporting me !

You can support me with technical support or by donating to the demo at Itch.io !

Table of contents

Release note : Describes all the updates on the Storm Breakers package.

Getting started : Tutorial to use the Storm Breaker package quickly.

Global architecture overview : How the engine is structured.

Waves model overview : A description of the innovative wave model.

Physic overview : How the physics is implemented and how you can adjust it.

Rendering overview : A description on the rendering.

Visual effect overview : How the particle systems are processed.

Audio overview : How the audio is implemented.

Gameplay overview : Some gameplay considerations.

Unity editor interface overview : Some words about the Unity interface of the package.

Performance overview : Considerations about the performances of Storm Breakers.

Example scenes : What the example scenes contain.

Prefabs references : How to use the prefabs provided in the package.

Scripts references : Extended descriptions of all the scripts and how to use them.

Visual effect references : Descriptions of the visual effect properties.

Materials references : Description of the materials and their properties.

Release note

- 01/2023 : Initial Release.

Getting started

This section explains the steps to use the Storm Breakers package in a project.

Requirements

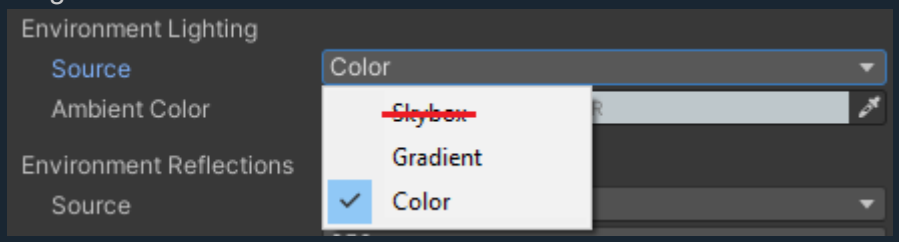
Here are the following requirements that need to be checked before importing and using Storm Breakers.

- Unity 2021 LTS +
- Universal Render Pipeline (URP)
- Visual effect graph package

Limitations and known bugs

The Storm Breakers package is still in active development and some limitation and bugs are not yet fixed:

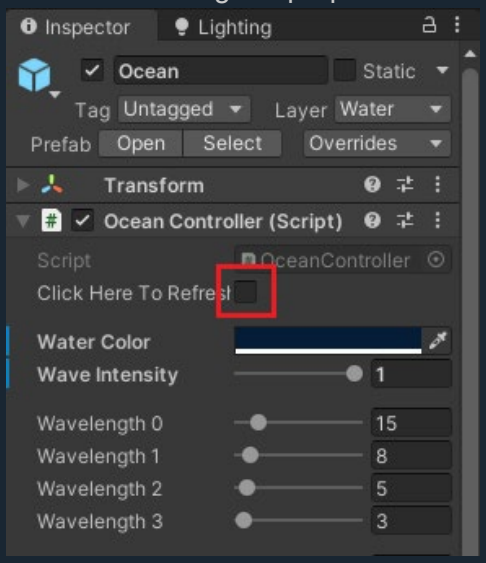
- Skybox ambient light mode is currently not supported, if you use this mode then the particle lighting might be wrong.



- The breaking wave particles are not synchronized with the waves when *not* in play mode.

- It is recommended to use the provided skybox material to improve the water reflection. Otherwise you might face wrong reflections if the bottom of your skybox is of a different color than the sky.

- When saving the scene or refreshing the asset, the waves properties of the ocean material might revert to a previous state. Click on the tick box “Click Here To Refresh” of the Ocean Controller component to refresh the material with the good properties.

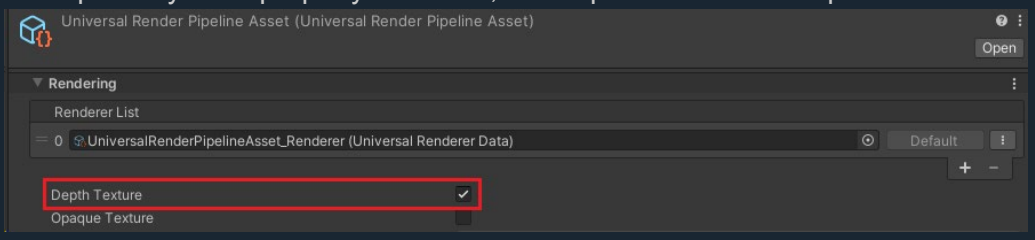


Importing Storm Breakers into a project

The steps for importing the Storm Breakers package into a project.

URP configuration

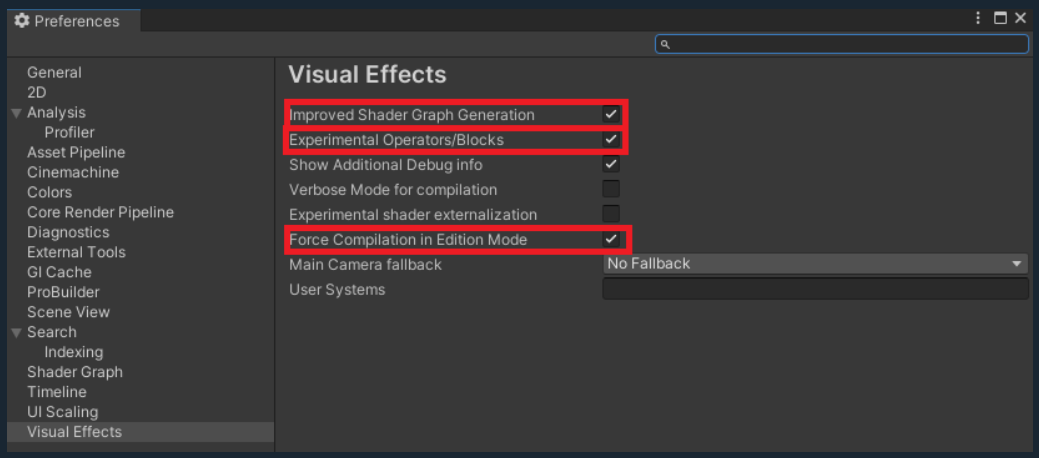
For the water transparency to be properly rendered, it is required to enable depth texture in the URP asset.



If you don't know which current pipeline is used or where it is saved, you can see it by going to Edit/*Project Setting/Graphics* and clicking on the scriptable render pipeline field, this will highlight the asset file in the project browser window.

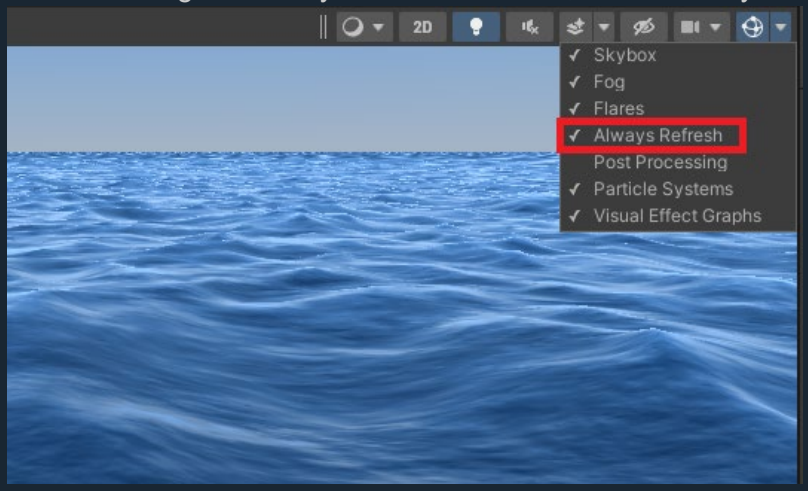
VFX graph configuration

Before using Storm Breakers, make sure you have imported the VFX graph package from the package manager window. Then go in *edit/preference/Visual Effects* and make sure you have the highlighted settings on :



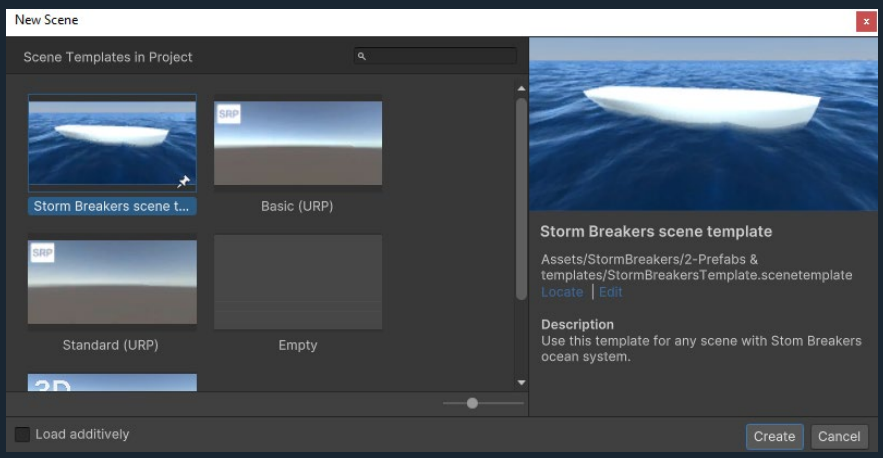
Scene configuration

To be able to see the waves movings smoothly in the scene view, enable always refresh mode :



Creating a scene with a template

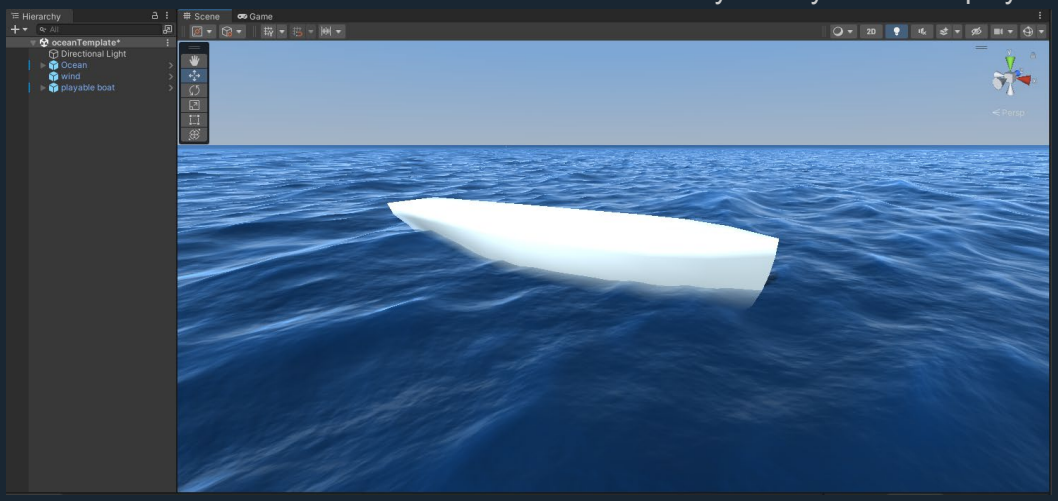
It is recommended to use Storm Breakers by using the provided scene template for a better user experience. To do so, use the new scene dialog in *File/New Scene*. Select Storm Breakers scene template and click on *Create*.



In the new scene dialog, enter the name of the scene and choose the path where you want it to be saved.

The new scene dialog will then create a new scene and a folder named the same as the scene. In this folder you will find clones of the ocean material and skybox material, you will be able to modify their properties for having changes in the new scene without having impacts on any other scene.

In the scene that has been created, you will find the basics for editing your level : an ocean, a directional light, a wind controller object and a playable boat. Note that you can immediately play the scene, you can control the camera with the mouse and the boat with the arrows of your keyboard and play in the waves.



A scene created with the scene template.

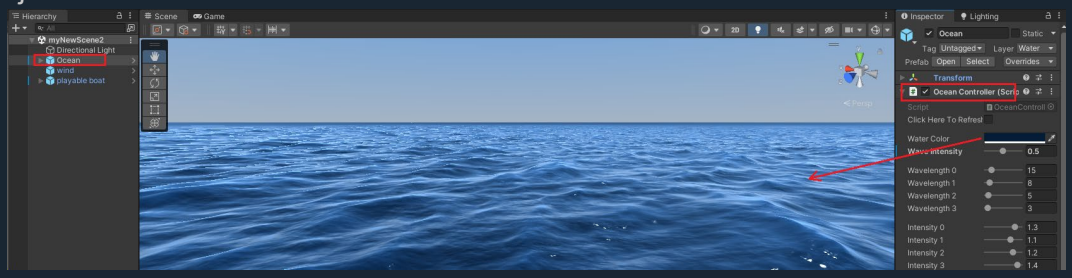
In this scene you can change the environment, mind the ambient and skybox limitation.

Setting up the ocean

Now that your scene is created, you can set up the ocean. Here are the basic steps to do so.

Water color

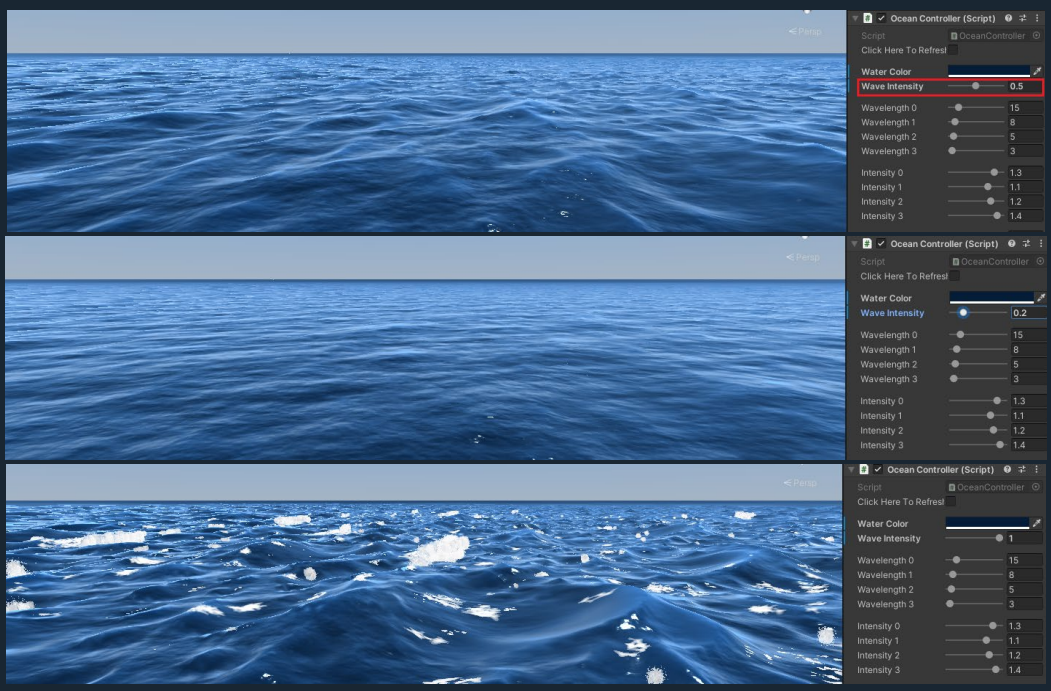
You can change the color of the water with the corresponding field of the Ocean Controller component of the game object ocean.



As a general rule, the clearer and deeper the water is, the darker its albedo is. See Ocean Controller component reference for more details. The default color is based on a real sea color.

Waves

The Ocean Controller component allows you to fully set up the waves. There are 4 wave layers that need to be in the order of the largest wavelength. Use the first slider “wave intensity” to quickly turn on and off the waves in the ocean without modifying their relative properties.



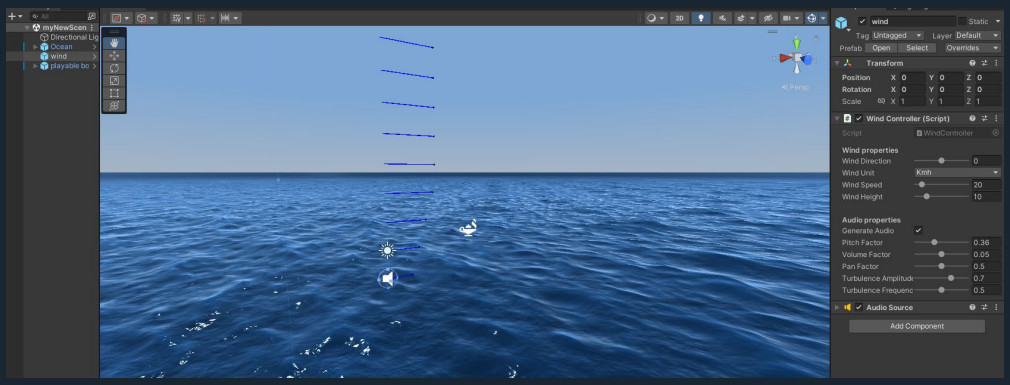
If you want to change the height of the wave, you may use the wavelength sliders and intensity sliders. The wavelength will change the scale of the wave while the intensity will change the sharpness. You can check the swell height and period in the bottom of the component (read only field).

Otherwise you can tweak the other sliders to change the wave looks and direction, see Wave model overview chapter and Ocean Controller component reference in this document for more detail on how to precisely set up the waves.

Wind

The scene template creates a game object called wind. This game object is not necessary for the simulation to work but it adds a nice audio and it also changes the looks of the water surface according to

the wind strength you provide. I can also simulate the wind caused by the velocity of the camera. In the component Wind Controller, you can set up the wind direction and speed in the unit you want. You can also set up the height at which the wind is full strength (at sea level the wind is halved). Turn on the gizmos to have a visualization of the wind vectors.



The wind is represented by the blue lines in Gizmos mode.

Adding watercraft in the scene

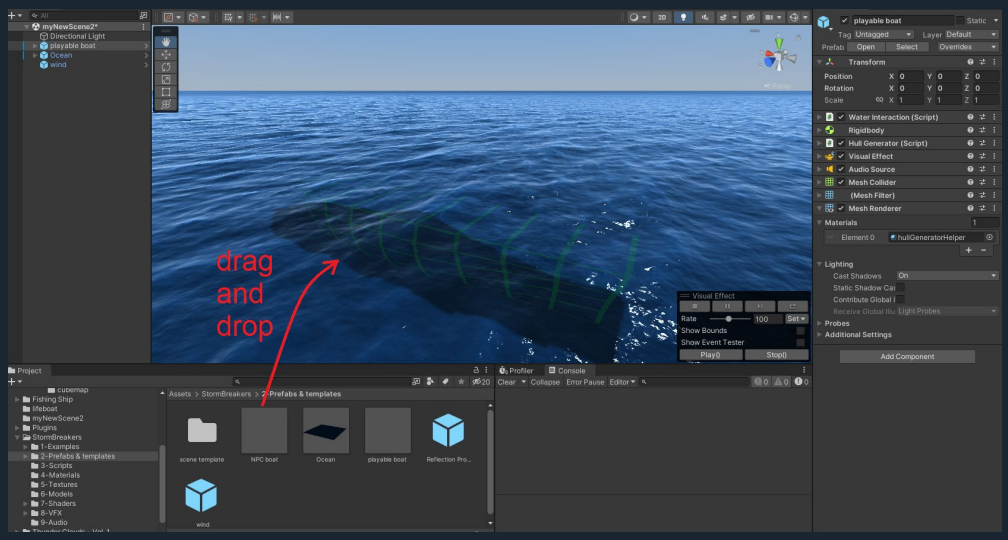
Storm Breakers includes all the effects to simulate the interaction of watercraft in the water : buoyancy, visual effect and audio.

Using a prefab

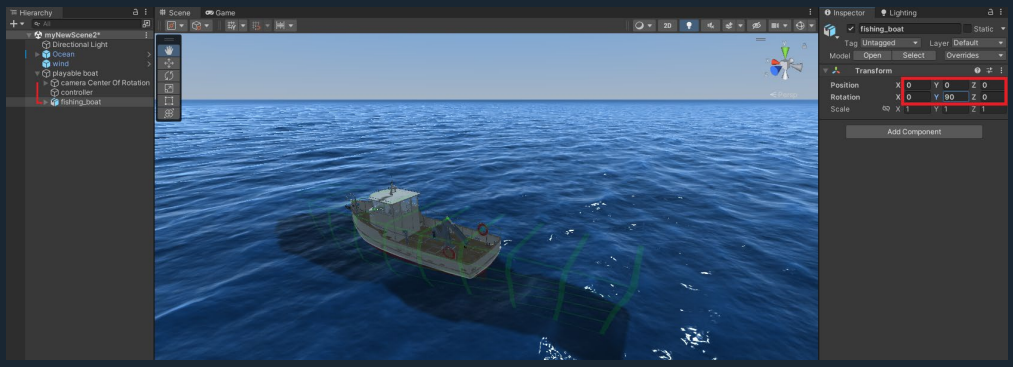
To add a boat or ship in the scene, select the playable boat prefab or NPC boat prefab that are located in the folder *2-Prefabs & template*, drag and drop it in the scene where you want it to start.

Both prefabs are quite similar, the difference is that the playable boat contains a camera setup while the NPC is set up with the automatic pilot on. (If you set 2 playable boats in the scene, there will be conflict with the camera and the input system.)

The scene created with the template already contains a playable boat with a mesh renderer set to a white lit material so you can see it in the game when playing immediately after creating the scene. The prefabs have a green grid material attached to preview the simulated mesh generated by the Hull Generator component, this material will not be rendered in the game and act as an helper.



You can unpack the prefab to have more freedom. Set the 3D model of your boat as a child of the game object, set its position to zero and modify the rotation so it is aligned with the default simulated mesh. Check the scale of your model, it should correspond to the real world object with units in meter.

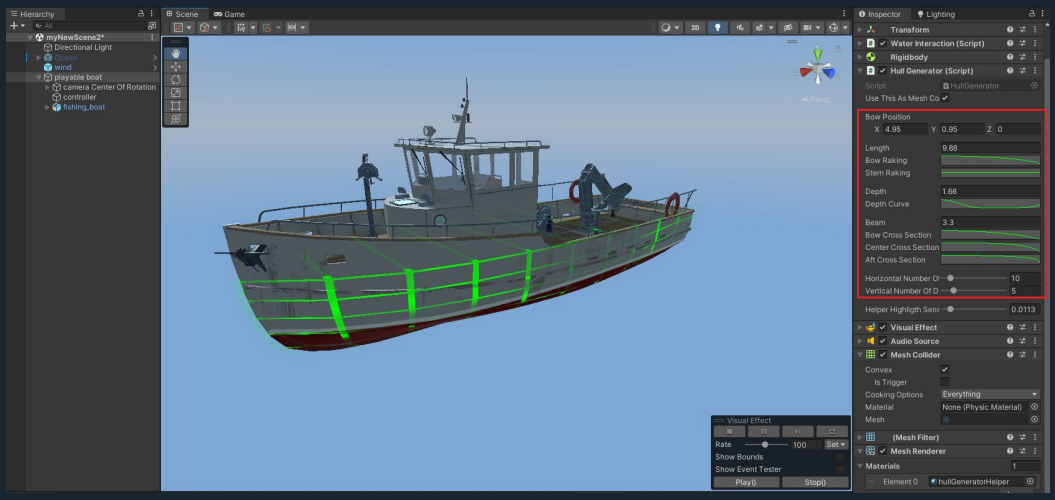


Setting up the simulated mesh

The simulation uses a specific mesh and computes all the effects per triangles on this mesh. So the simulated mesh must have a decent number of triangles or your game will run into high CPU overhead.

Storm breakers package provides 2 ways of setting the simulated mesh : either using an existing convex mesh collider if you have a suitable one, or create one procedurally using the Hull Generator component. It is recommended to use the tool when suited as it will allow you to change the shape of the hull and optimize the simulation by tweaking the number of triangles in the generated mesh. To choose between the two ways, enable the Hull Generator component if you want to build it procedurally, or deactivate it and set your mesh collider in the mesh collider component.

If you choose to build the simulated mesh procedurally, set the properties and curves in the Hul Generator component so it fits the hull of your model. See Hull Generator component reference in this document for more detail on how to use it. You can deactivate the ocean while working on the hull to have a clearer view :



It is not a big issue to have a rough approximation of the hull shape, it will barely have a visual impact on the effects. It's actually recommended to take some liberty on this shape so as to tweak the dynamic behavior of the watercraft, see the physics overview section of this document for more info on how to tweak the physics. The quad should be as square as possible.

The mesh renderer with the material HullGeneratorhelper acts as a helper by showing you the quad (2 triangles) that are generated and highlighting the edges that are close to the model. Once you are satisfied with the mesh, you can deactivate the mesh renderer if it annoys you, it will be deactivated anyway in the game if it's the helper material.

Note : you can choose whether to set the generated mesh as a collider or not by ticking the box of the Hull Generator component. If set to not, you will have to define an existing convex mesh collider, this mesh will be used by the internal unity physics engine while the generated one will be used by the Storm Breakers engine.

Setting up the buoyancy

Now that the simulated mesh is defined, you have to set up the buoyancy effect. For that you need to go in the Water Interaction component. This component is designed so that most of the effects scale with the size of the object, yet some tweakings are inevitable.

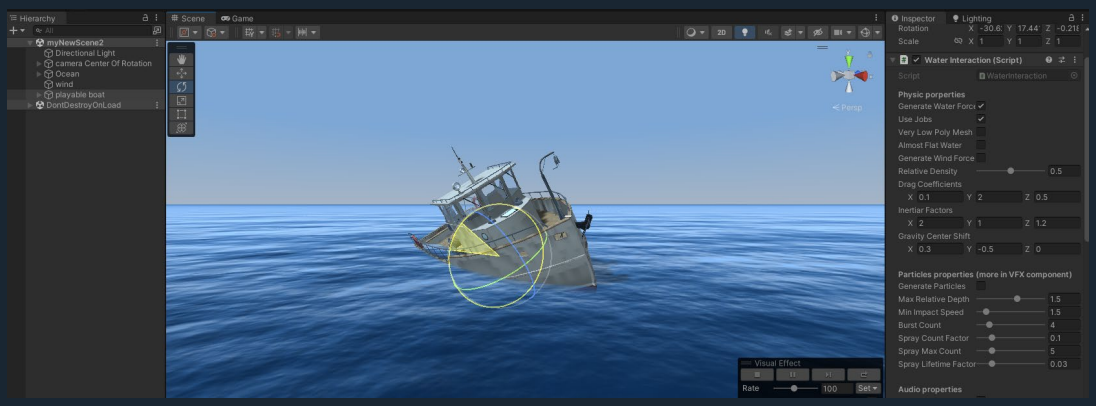
First, make sure the ocean game object is active and turn off the wave by setting their intensity to 0. This will make a flat water in which it's easier to set up the physics properties. If it's a NPC boat, you have to deactivate its controller to avoid it moving. You may also deactivate the audio and visual effect in the Water Interaction component by ticking the corresponding box so you are not bothered by those effects while setting up the physics.

Then launch the game, the boat should float somehow. What you have to do is set up the density and the gravity center's position until the boat floats correctly at its water line. See Water Interaction component reference for more detail on how to set up the physics.

- If the game object contains colliders as childs, they will be used by the rigidbody inertia computation, so check if they are consistent.
- If the boat immediately capsizes, you should lower the gravity center before entering the play mode.
- If the boat oscillates too much, you should increase the built-in drag coefficients of the rigid body.
- If the boat moves on its own permanently and the controller is not activated (this is due to some computation approximation), you may tick "Orient Force Vertically" to set the forces acting only vertically. Untick this as soon as the watercraft is in motion to avoid wrong physics.
- Before leaving the play mode, don't forget to save the modified properties by copying the component and pasting it as values when returning in edit mode.

You can activate the gizmo mode to see the position of the gravity center (yellow wire sphere) and the forces acting on the hull (red and blue lines).

Once the static stability is set up, you may tweak the dynamic stability. For that, use the transform rotation tool to heel the boat and see how it reacts.



- If the boat capsizes at a too small helling, lower the gravity center
- Tweak the gravity center as high as possible until the boat right itself at a reasonably slow speed (a second or so depending on its size). If set too low the boat will be too sensitive in the waves, see the physic overview for more details.
- You can tweak the inertia tensor factor function of the watercraft geometry, the default values being set according to a classic hull. If you have colliders as childs in the game object, the rigidbody automatically includes them in the inertia computation.
- Again don't forget to save the modification done in game mode to set and save them in edit mode.

Then you may try to make the watercraft navigate in the waves by setting back their intensity and activating the controller. If its behavior doesn't suit you, see Physics overview for a complete discussion on how to tweak the dynamic of a watercraft, and Water Interaction component reference for more details on the component setup.

Setting up the controller

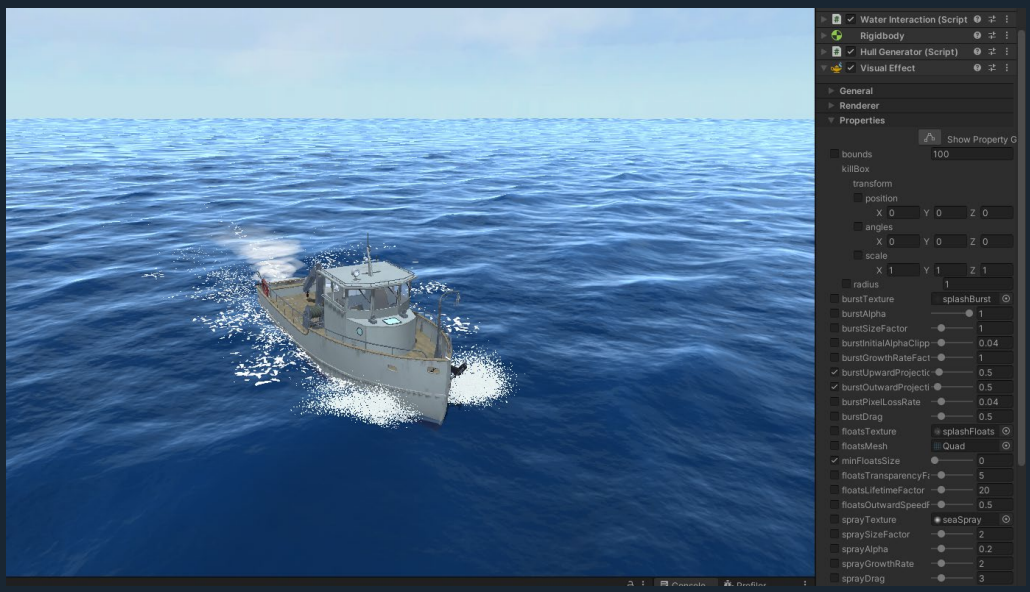
Storm Breakers provide a basic boat controller that makes a propeller and rudder force. It has an automatic pilot that can maintain the course with a constant propeller force. If the automatic pilot is set to off, the game input will pilot the boat controller (arrows or WASD by default in Unity).

To set up the controller, position it at the boat's propeller and rudder position for a realistic behavior. You can turn on the gizmos to visualize its direction. Be careful that the default values provided don't automatically scale with the size of the boat, you may drastically change them according to the weight of the boat it is attached to.

If you enable the automatic pilot, it will automatically maintain the course it has when enabled and use as much throttle you set. See Boat Controller component reference for more details on how to set it up.

Setting up the visual effects and audio

Once the physics of your boat is set up, you can go on to the visual effects and audio setup. For that the easiest is to enable the automatic pilot of the controller so the watercraft moves on its own while you tweak the values, and to set up *a bit* of waves on the ocean to make more splashes. Then enter play mode to see and hear the effects.



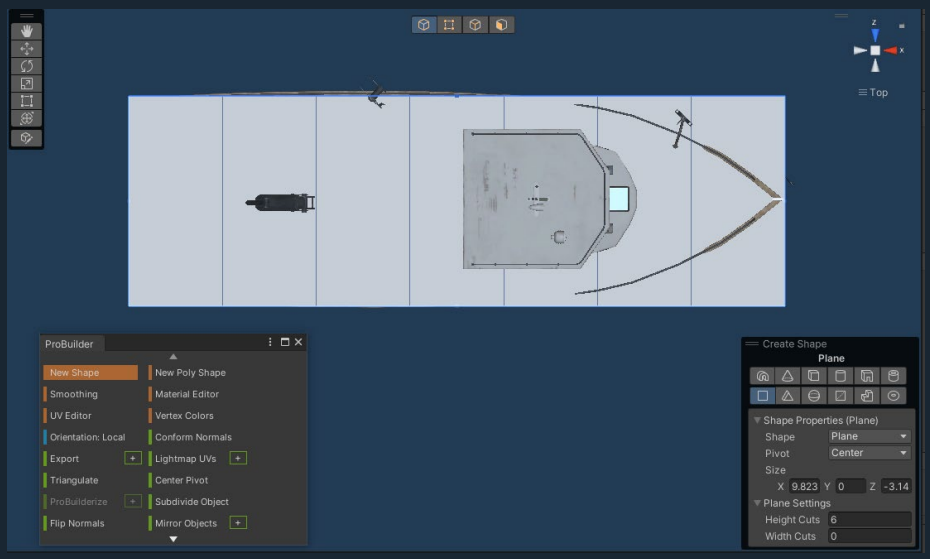
The particle system should scale well to any kind of size, but tweaking is necessary to polish the effect. See the particle overview section of this document for an in-depth knowledge of the water interaction particle system. Mind that there are some properties set in the Water Interaction component while most of the others are on the VFX inspector. Don't forget to copy the properties before leaving play mode.

For the audio you might need to tweak the volume factor and pitch factor function of the size of the watercraft. Do it in the Water Interaction component.

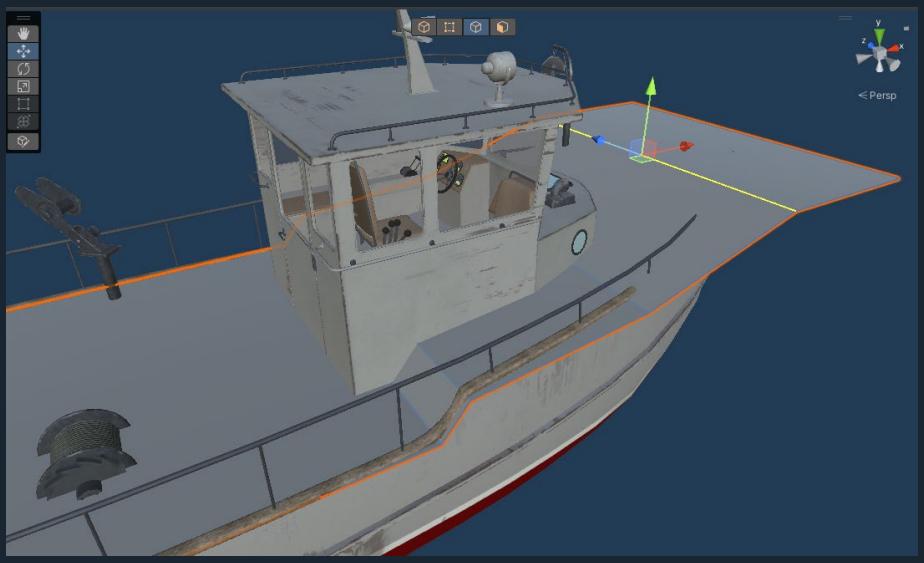
Adding a deck mask

It is recommended to create a so-called deck mask that enclose the railing of the boat to prevent the water and particles to be seen on the deck. If the railing of the boat is not watertight (with gunport for example), this mask will currently leave an unset color in the opening (this is meant to be fixed). So you may have to consider either closing the openings or not using a deck mask (this will look like the water is coming through the opening), or leave the unset color if the openings are small enough.

To build this mask, you can use the Unity Probuilder package and assign the mesh the water&ParticleMask material. The mask will become invisible but no water nor particles will be drawn behind.



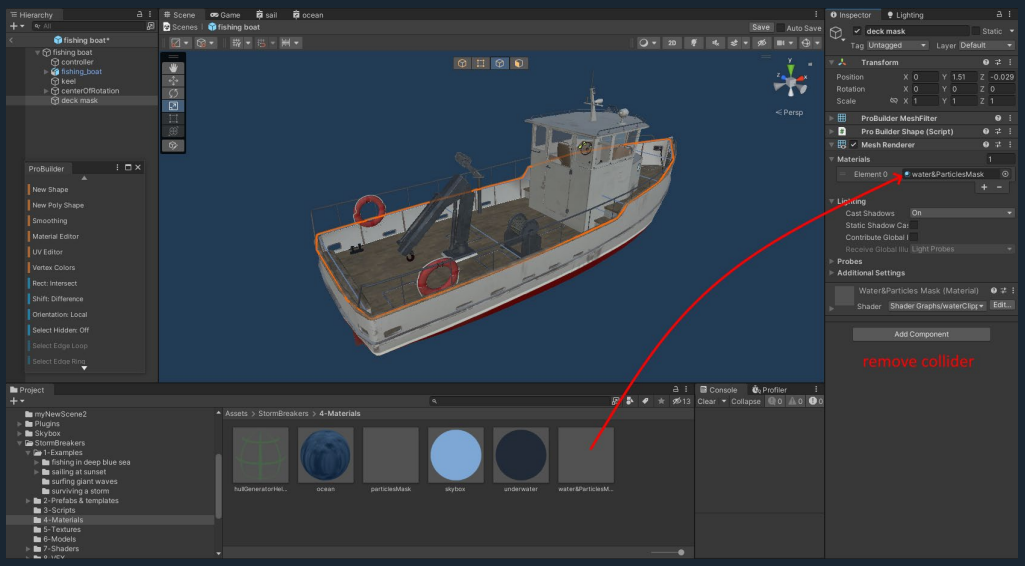
Step 1 : create as a child a plane with several cuts along the length.



Step 2 : center the new geometry and position the edges on the top of the railings. You may break the regularity of the cuts to closely follow the railings.

Step 3 : use the scale tool to adjust the width of the mask at each cut.

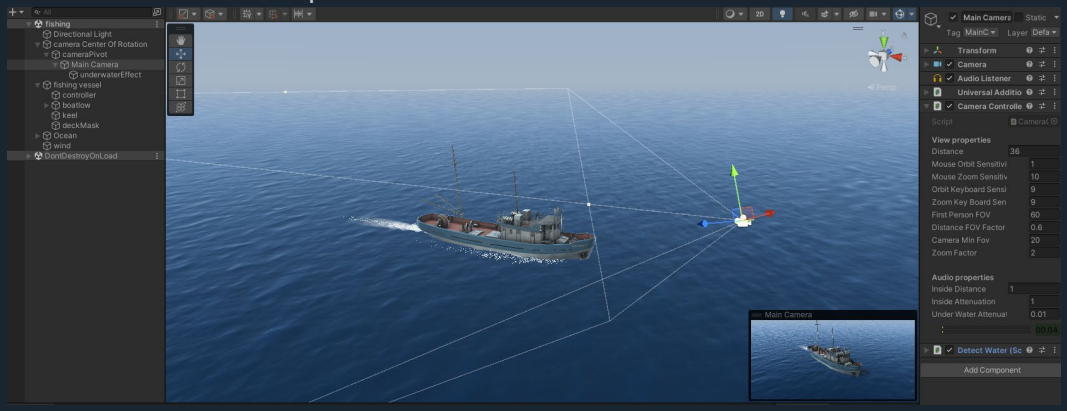
For the bow you may triangulate the first quad and remove a triangle.



Step 4 : remove the collider and attach the water&ParticleMask from the ‘4-Materials’ folder of Storm Breakers.

Setting up the camera

If you use the playable boat prefab, it includes the setup for an universal orbital camera that follows the object it is attached to without its rotation. It works both for a first person view and a third person view, you can switch from one to the other by scrolling up and down the mouse wheel. The camera setup also includes a zoom feature in the first person. See Camera Controller for more details.

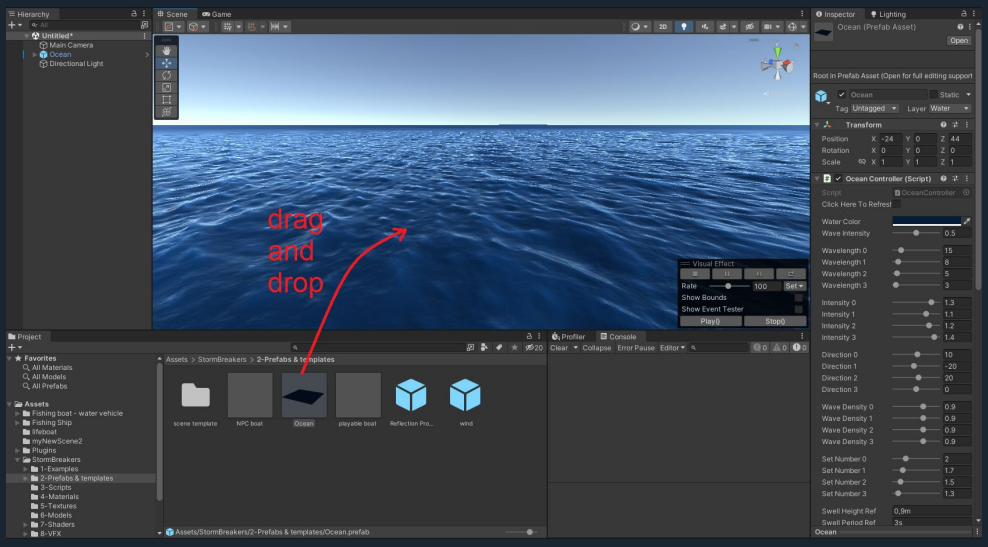


The universal camera setup provided in Storm Breakers.

You can use any other camera setup, mind that there must be one camera tagged “main camera” in the scene, it will be used to align the ocean mesh and orient the particles. Also if you want the underwater rendering to be effective, you must set a quad at the near clipping plane of the camera. See Underwater component and material reference.

Insert ocean in existing scene

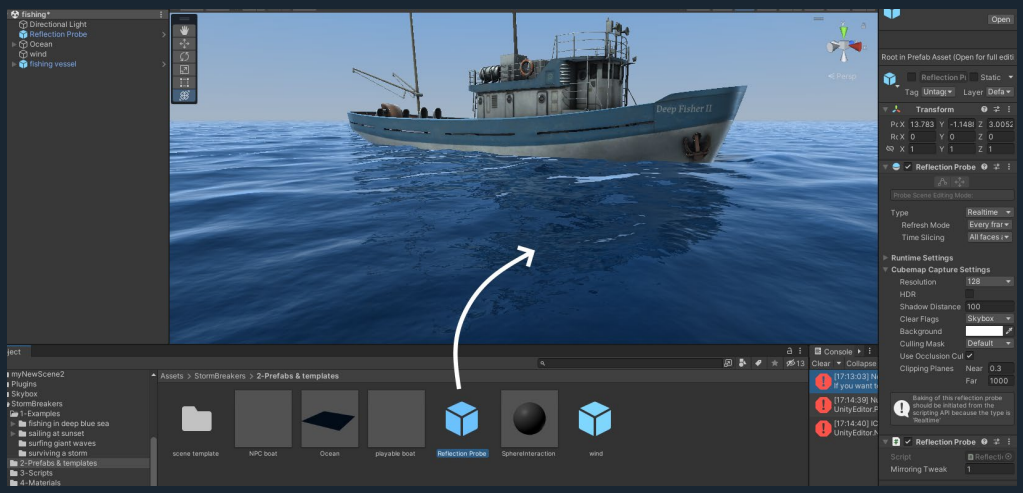
It is possible to add an ocean in an existing scene without using the scene template, simply drag and drop the ocean prefab in the scene. It will automatically be positioned in front of the camera at world y = 0. You cannot currently set it at a different height.



Adding dynamic reflection

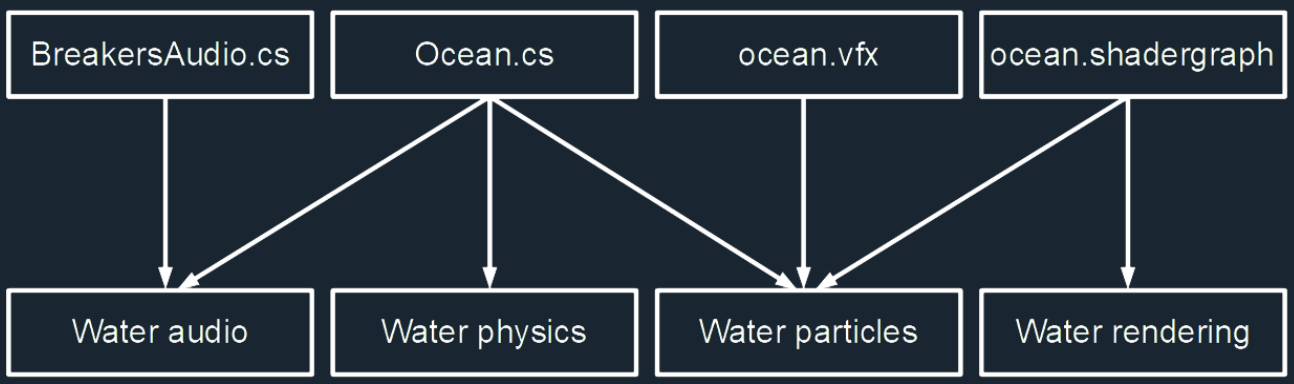
Storm Breakers provide an approximate way to make the reflection of dynamic objects in the water without using the screen space reflections that are unavailable in Unity URP. The trick is that it positions a reflection probe as a mirror to the camera relative to the sea level. It works only for good graphic hardware and on almost flat water.

Drag the Reflection Probe prefab in the scene to activate this effect. You may have to tweak the reflection probe properties to fit your project quality and hardware, the default setup being of a low quality.



Global architecture overview

Storm Breakers has been coded in a very forward way, notably the ocean model is computed many times with different codes through the CPU and GPU without any callback from any of these codes. This way of coding, although being harder to maintain, provides a very fast computation. The following diagram shows which ocean-related files participate in which effect.

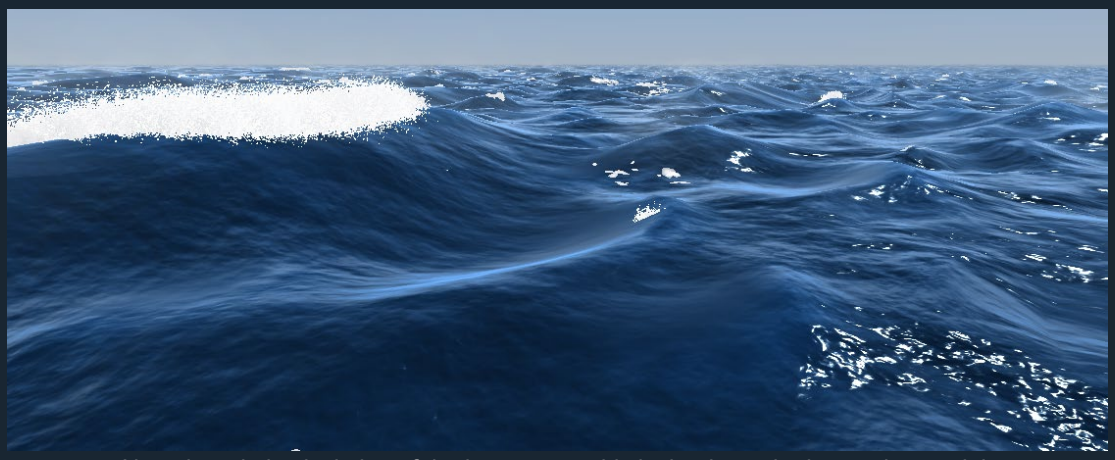


This is possible thanks to the fact that the water surface is one deterministic mathematical function. It will be difficult to add other kinds of waves like the wake waves of an object moving through the water. While it could be possible to add one mathematical function on top of the ocean model, what would be tricky is to add an unbounded number of them because GPUs don't allow much conditional computing. So for now there are no plans on adding the wake waves, a rogue wave might be added in future updates depending on user feedback though.

The determinism of the ocean model is bound to the floating arithmetic approximation. On large distances and for long times, the float values might get imprecise and lead to some artifacts. It is not known yet whether those approximations are different on GPU and CPU and when it happens on both. So if you plan on making an open world where you sail for a very long time, it would be recommended to have a fix that would bring the spatial and temporal origin back to the present time and position. This fix might be included in Storm Breakers futures updates.

Waves model overview

At the heart of Storm Breakers lies an innovative and unique wave model that makes a more realistic rendering in agitated water. Both the shape of the wave and their repartition in space were designed with a new approach. In this section it is explained how this has been coded, you may need to know about if you want to refine the setup of the waves.



Note the relative isolation of the large wave, this is thanks to the innovative model.

Deep water waves

The wave model of Storm Breakers is suitable for deep water only. Ocean, sea or large lake can be rendered in Storm Breakers.

Shoreline waves are pretty different and cannot be properly rendered with the Storm Breakers model. Currently there is no water depth effect at all in Storm Breakers and you might face wrong waves if you try using it near shoreline. Future updates might have some features for this.

Lake waves are deep water waves too, but with a smaller scale. A thinner and bounded mesh is needed to render them and is not currently supported by Storm Breakers that focus on ocean and sea.

Rivers waves resemble at some point with the deep water waves, but it would require many adjustments to be rendered through this model, so they are not supported.

Waves shape

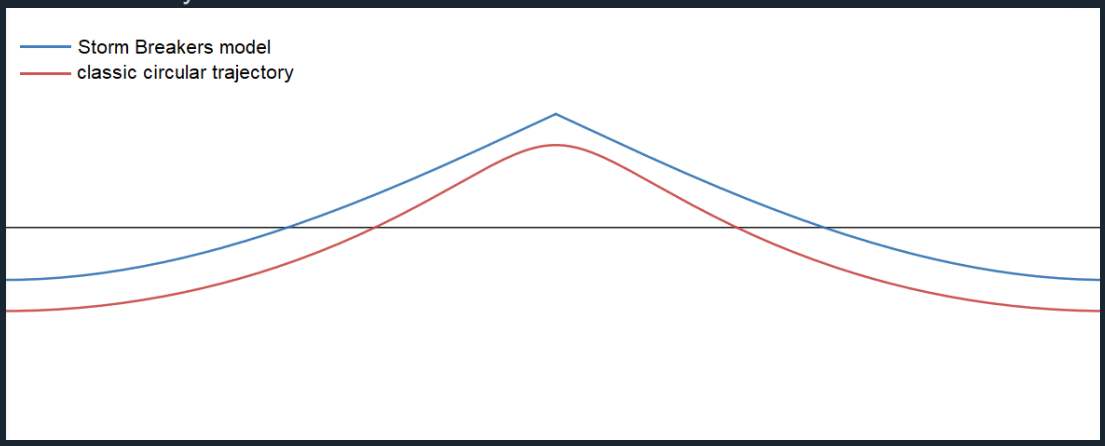
Waves are coded with a more advanced technique than with the classical water circular trajectory used in the Gerstener model or many others. Here is a description of what is new.

Sharp waves

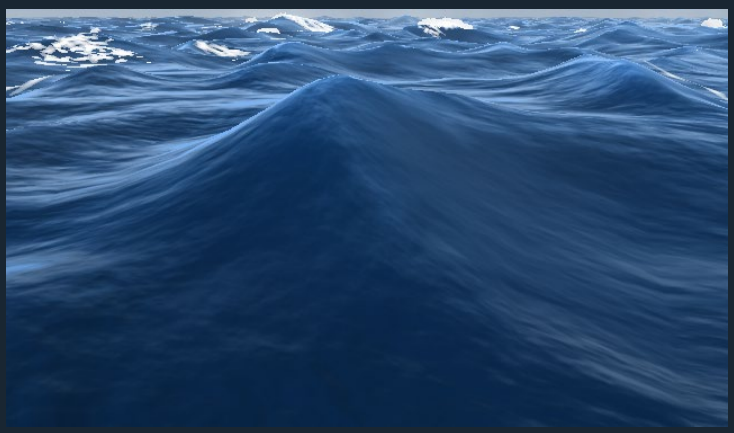
An effort has been done to make the waves have the correct shape when near breaking, which is a crest with a sharp 120° angle, its height being 0.17 times its wavelength.

Using solely a sine function like it is done in most of the other ocean models might be very handy and a very good approximation in calm water, but it can't make such a shape. It either makes a concave wave which looks too thin and brittle, or the crest is too round. Only with the sharp 120° crest a wave looks both strong and aggressive and this has been implemented in Storm Breakers by using more functions than sines.

Also the waves are not symmetrical to the sea level, there is more crest height than trough depth to simulate the non linearity of the wave.



Not only making sharp waves makes them look more dangerous, it also provides a new gameplay possibility, and not the least : with more straight slopes, we can surf the waves more easily !



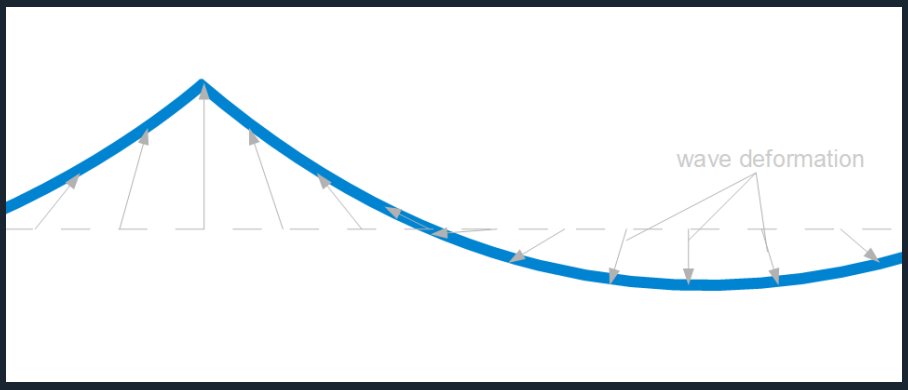
Note the straight slope of the wave, this is thanks to that we can surf them.

Vertical and horizontal deformation

A wave is not only a vertical movement of the water, there is also a horizontal displacement. In reality, at low amplitude the water moves in orbit around their rest position, at high amplitude the movement becomes an open ellipse. The water doesn't come back to its original position after a high intensity wave passes, it's called Stock drift.

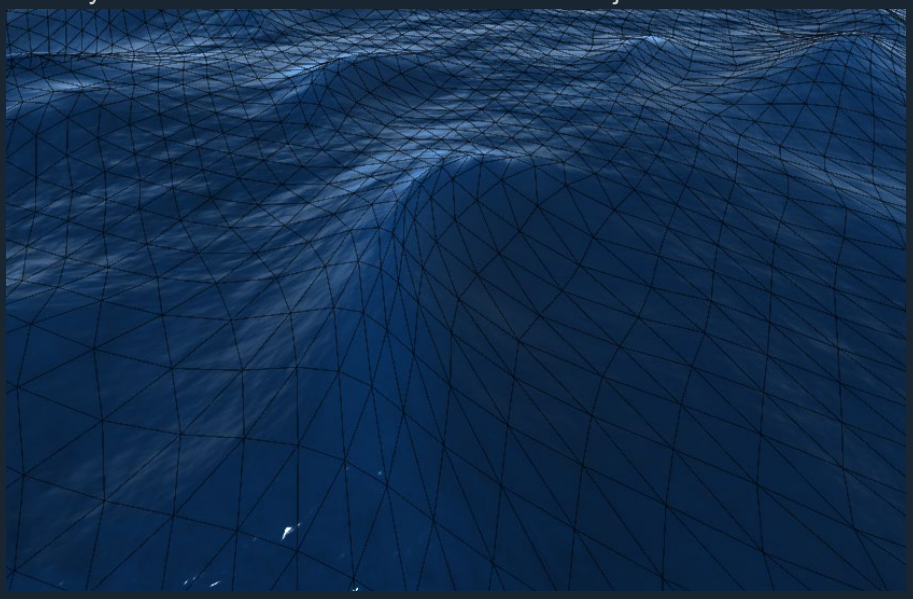
The exact trajectory of the water has not been coded in Storm Breakers. First because a drift is not possible without keeping the mesh steady, and secondly because it has been decided that the external shape of the waves matters more than the wave velocity which is less visible. The actual trajectory is bound to the sharp crest and the following aspect of the wave.

Having an horizontal deformation on the waves makes the computing of the water height non obvious. See physics overview in this document to learn how this has been solved.



Naturally tessellated waves

The trouble with a sharp crest is that it would jitter while moving due to the mesh size. A tessellation would be required near the crest to make sure there is a maximum mesh density here. This has been implemented without actually adding triangles through a shader tessellation, but using the horizontal deformation of the wave. In reality the water is compressed at the crest (which is actually why the water elevates to form a wave), in Storm Breakers this effect has been amplified up to the point that the mesh becomes almost infinitely dense at the crest so the crest doesn't jitter.

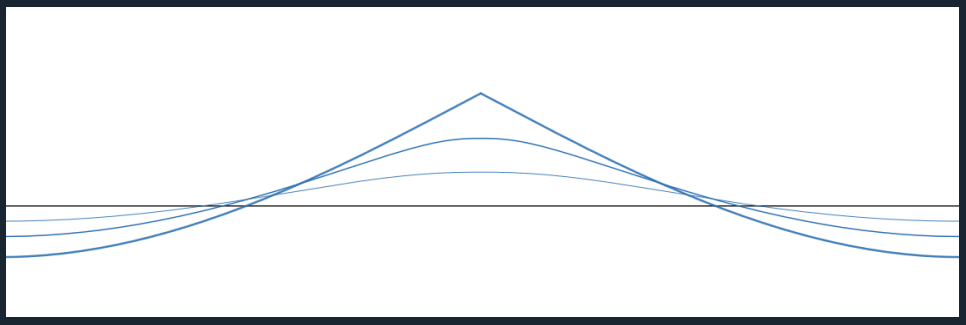


Mesh density is increased at the crests to improve their rendering.

The downside of this way of coding waves is that the water needs to move faster forward at the crest, and faster backward at the trough. Seasoned eyes might spot that unrealistic aspect.

Wave intensity

The crest is sharp only for waves near breaking. Lower amplitude of waves would lead to a classic round shape like it coded in most ocean models. The way Storm Breakers blends between calm and breaking waves is through the value called wave intensity. When this value is near 0, the amplitude is very low and the waves are round. When the intensity is near 1, the waves become sharper and higher. At 1 is the limit of breaking, the wave height is 0.17 times the wavelength like in reality, and the crest sharp. Above 1 the amplitude is clamped so as not to rise any higher, simulating roughly the fact the wave breaks.



In the order of the highest wave : intensity > = 1, intensity = 0.6, intensity = 0.3

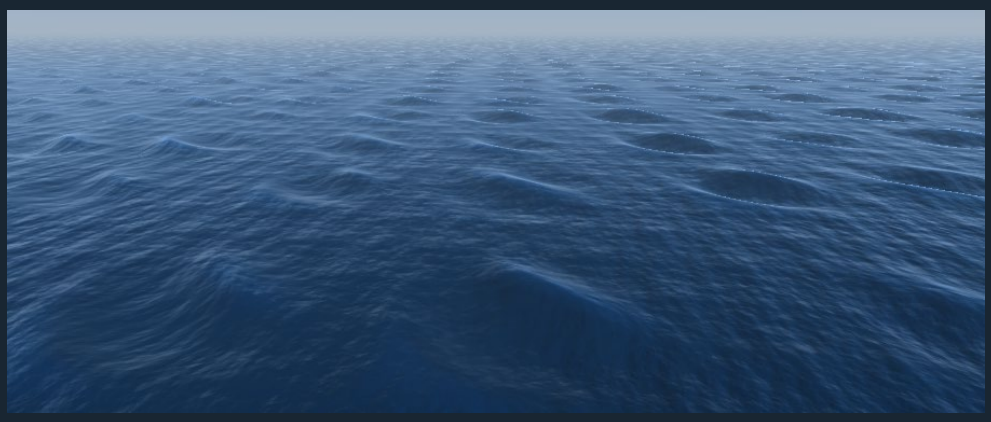
Breaking wave

The breaking waves in Storm Breakers are done through an advanced particle system and through procedural audio. They also make an extra force that can capsize small boats. See corresponding section of this document for the details on how the effects work.

A 3.5m high breaker in a storm.

Like explained before, the mesh isn't deformed to render the lip of a breaking wave. However, recent research showed that it might be possible to code plugging breaking waves using the internal group position to have access to the wave history. Future updates of Storm Breakers might include such a feature.

Groups of waves



The ocean rendered with only one layer of wave, revealing the groups.

Unlike other wave models like Gerstner or Fast Fourrier Transformed (FFT) that build waves by superposing a high number of infinite and permanent wave layers, Storm Breakers model try to get closer

to the real physics by coding waves as groups, or sets. This has many impacts on the visual and the gameplay capacity, this documentation tells about that.

Definition of groups

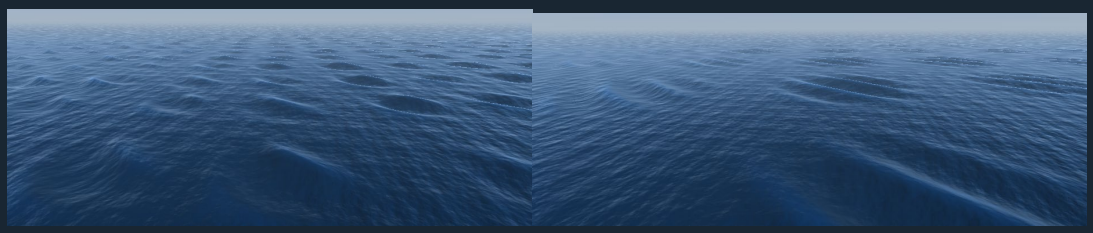
What is a group, or set of waves ? It's the fact that waves are not infinite but localized and grouped. The amplitude of the waves within the group vary, being maximal in the middle and null at the edges of the group. It is commonly known by surfers that waves arrive by a set of several waves, and it is the same phenomena.

Number of waves per group

So how many waves are there in a group ? That depends on how old they are and the answer is not a round number. An old swell that travels hundreds of kilometers in the ocean can reach a half dozen or more waves with the ones at the edge barely visible. This is what surfers are used to see and it's the origin of some sayings like the 6th wave is bigger.

What about at sea ? When the waves are younger they barely build a group of several waves. In situ observation showed that in agitated water, the breaking swell rarely reaches more than 2 or 3 waves.

In Storm Breakers you can adjust this number with the sliders in the Ocean Controller component called Set Number. Most of the time you don't need to tweak much these parameters though, low values between 1 and 2 are good to render most of the sea state. Only an old swell would require a number higher than 3 and even this would lead to some weird visuals.



On the right groups with 2 waves, on the left with 5 waves

The larger is this number, the longer the wave exists and so it helps surfing them if you wish to set up such a gameplay.

Because of the way the groups are scattered, the wave number has quite an influence on the wave density as explained further in this document.

Group speed

One important consequence of coding waves as groups is that we can then code the group speed. The group speed is the speed at which the amplitude moves, which can be different from the wave speed (the celerity).

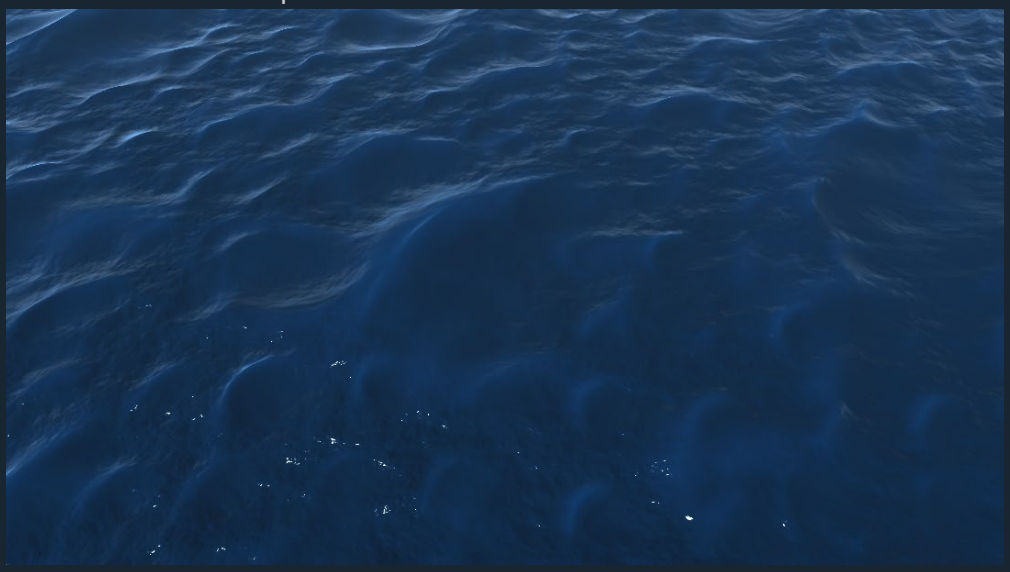
In deep water, according to the Airy theory, the group speed is half the celerity. This has a major impact on the waves and it explains why we can never track down a wave for a long distance in real sea : a wave appears at the back of the group, it builds up, sometimes up to the point of breaking, and then disappears, leaving another wave behind.. This is quite easy to see in agitated waters : when a wave breaks, another one just behind appears and breaks too because it reaches the middle of the group as well. This effect can be seen for several breakings and so we can see the group moving.

This is what has been coded in the ocean model of Storm Breakers but with a slight difference : low intensity waves see their group speed lowered so the waves appear and disappear more frequently. This helps compensate for the low number of wave layers when rendering calm sea.

Group independency

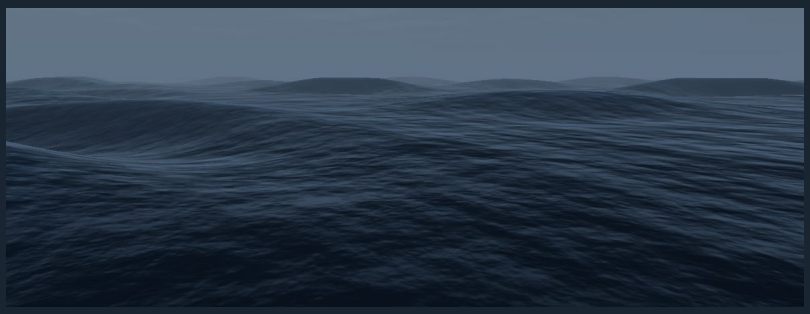
Coding waves as groups is like coding them as objects and they can behave on their own. Each group benefits from a randomized phase so their waves are not synchronized with other's, and their position can be scattered a bit (depending on the wave density) to avoid having groups being aligned. Another thing is that the intensity in the middle of the group varies with time independently of the others. So when a wave of the group breaks, the others behind might too but after a certain time the group loses its amplitude and the wave breaks no more. Further later the amplitude will build again. This makes the wave a bit less predictable.

Another thing that is coded is that the phase within the group varies function of the intensity. At high intensity the phase is advanced in the middle of the group, this makes a realistic horseshoe shape to the waves and gives them more visual power.



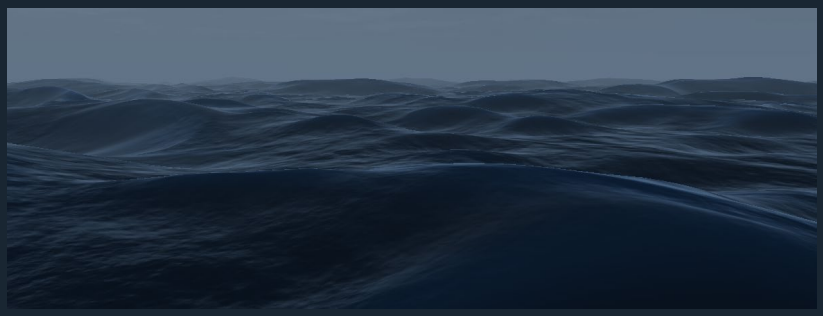
The subsurface scattering highlights the horseshoe shape of the small waves.

Wave systems



First system

Adding the second system.



Adding the third system

Adding the fourth system

Definition of system

The way Storm Breakers is coded is that it creates layers of grouped waves called systems. A system contains non-overlapping groups of waves of the same wavelength and direction. Technically each system is an amplitude map applied to waves, the map moving at a different speed than the wave to create the group speed effect. The amplitude maps separate groups in a rhombus grid, each group will benefit from a randomized phase and random position within its rhombus. This helps break any visible pattern.

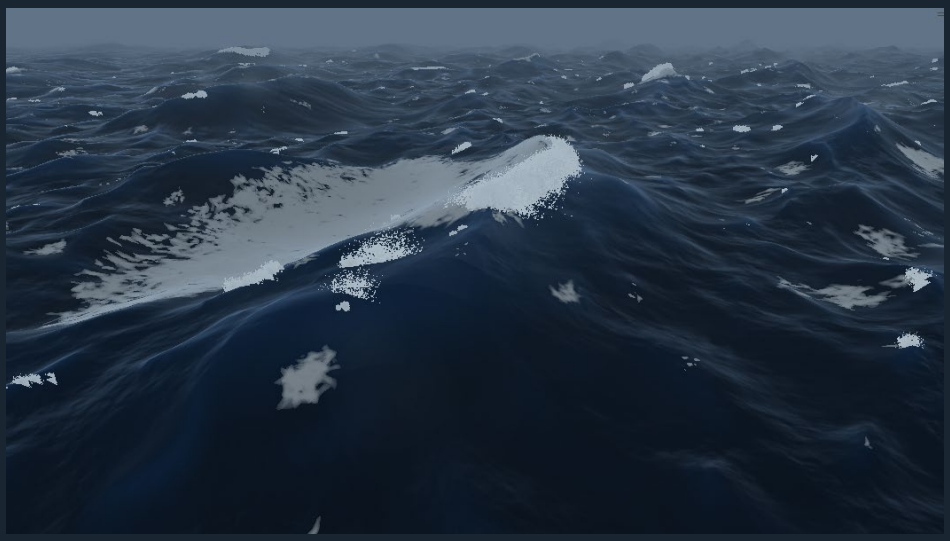
System implementation

Currently there are 4 waves systems implemented in Storm Breakers. For some special sea states it can be a bit limiting, but the aim was to reduce the number of parameters to tweaks and to maintain good performances. Indeed while other ocean models may have more layers, each layer in Storm Breakers is longer to compute being more complex. Though, thanks to the randomization of each group, visible patterns are easily removed and so 4 layers is enough for most sea states.

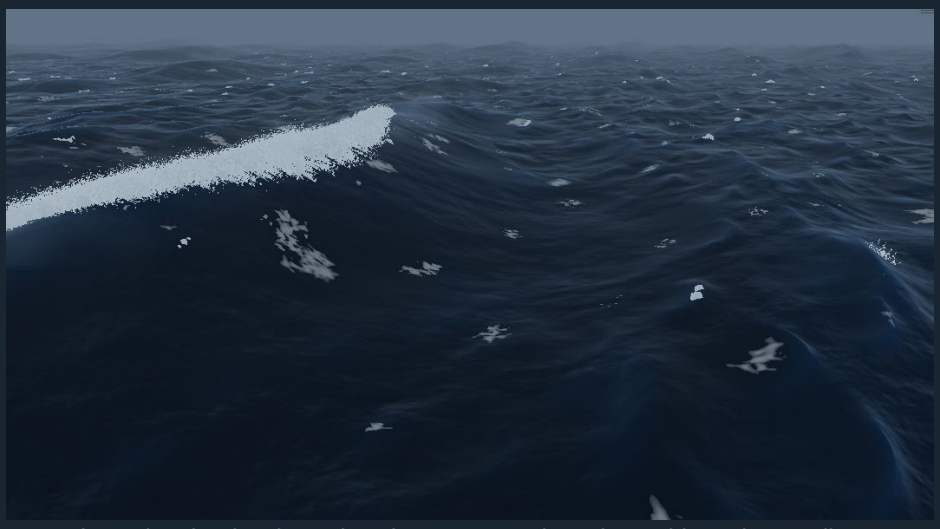
The current ocean controller component allows you to tweak every aspect of each system. See the component reference for a full description of all the properties.

Wave density

One advantage of the way the wave systems are coded is that we can tweak the wave density. This can be very important depending on your gameplay, for a lot of visuals you will need a lot of waves, but navigating through them might make you reduce their number to ease the steering.



With a high density of waves, the ocean is chaotic.



Lowering the density makes the ocean much easier to ride and to predict.

Since within a system groups cannot overlap, the density is coupled with their randomized position. With a density set too full in the Ocean Controller component, the group follows exactly the rhombus grid that is used to separate them. Then the pattern can become obvious if there is no other layered system with waves about the same size.

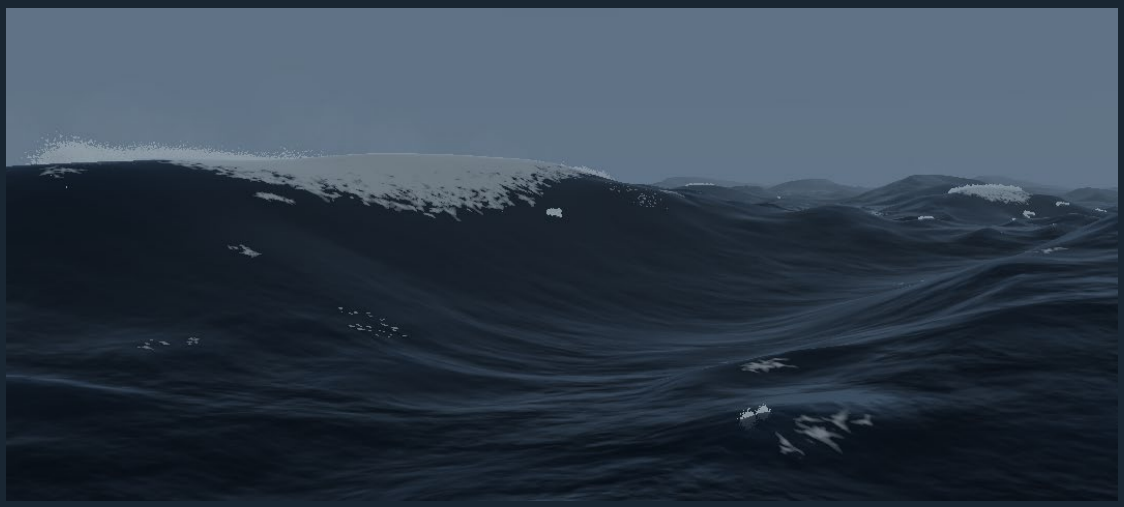
To avoid this pattern, reduce the wave density or bring the other systems to a similar wavelength and intensity.

The set number also affects the density somehow by widening the groups. This is why this value shouldn't be too high, or more systems would be required to render the sea.

Wave superiority

The way the systems overlap each other is not a linear addition of the deformation they cause, the concept of superiority is coded in the model : A large wave (long wavelength) with sufficient intensity will cause the smaller waves with the same direction to break and spill all their energy away, leaving almost nothing behind it. It's like large waves erase smaller waves with the same direction.

This effect can easily be seen in real waves. When we see a large breaking wave, it's one wave. Not much of other smaller waves on it, it's surface is almost smooth. This is because the wave of the same group that passed before erased almost everything here. Not only coding this is realistic, it also makes the large waves look even bigger and dangerous.



Note how smooth the water is within a large group.

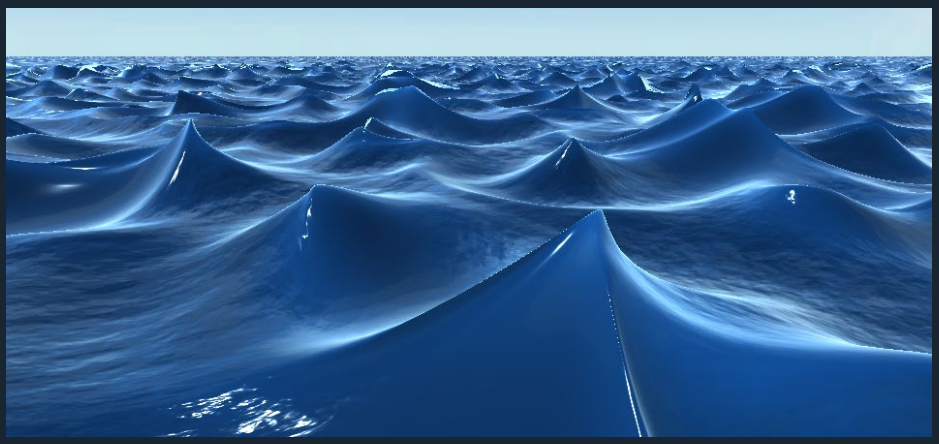
Wave superiority implementation

This effect is coded in the Storm Breaker ocean model. Since the waves and groups are not objects but merely mathematical functions, it is not possible to know if they have been erased by a large wave in the

past. What is coded is that the intensity within the group is used to clamp the amplitude of smaller wavelengths. This makes smaller waves being gradually erased inside groups of larger wavelengths, but they also build back to their original amplitude when leaving this group (which could be explained as the wind building again the waves.)

Direction-wise superiority

The direction of the overlapping group is taken into account. When exactly in the same direction, the superiority of large waves fully acts, but when the direction diverges it becomes less effective. This up to the point that there is no amplitude clamping starting when waves move with more than 90° direction difference, then the deformations of each wave add to each other, leading to a high and steep wave.



With the 4 system at 90° direction difference, it makes over-sharp waves.

When the 2 systems are moving in opposite directions, the mesh can self-intersect ! There is no fix for this except to lower the waves intensity.

Achievable sea states

With this innovative ocean model and thanks to the visual effects, it is possible to render various sea states. Here are some examples sorted with increasing agitation

Glassy sea.

Calm sea with a gentle breeze.

Freshening wind.

Cross sea

Stormy sea

Unrealistic sea state designed for gameplay with 10m high breakers.

Waves predictability

One of the outcomes of the described features is the fact we can tweak the waves predictability. You can make predictable waves if your gameplay makes the player sail through them. This could be part of the game mechanic : reading the ocean, spotting the large group of breaking waves, deducing their bearing and steering accordingly. Or you can make a chaotic ocean if you only wish to render an ocean without any visible pattern.

Those are the parameters to set up more predictability (do the opposite for less predictability)

- More difference between system’s wavelength
- More difference between system’s intensity
- Less wave density
- Less direction difference between system
- Larger set number on the predictable systems

Physic overview

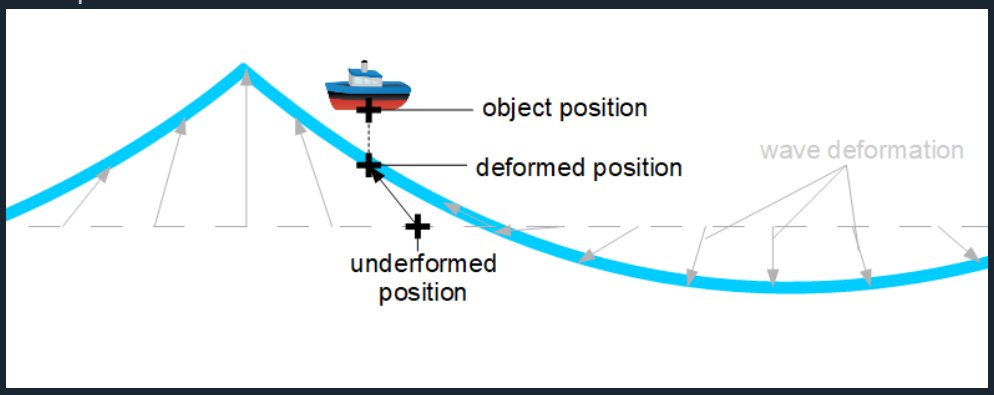
To fully benefit from the new wave model, Storm Breakers package provides all the required tools to make objects interact efficiently with the water, including buoyancy. There is also an API to get the water height, normal and velocity if you wish to use another physics system.

How water is detected

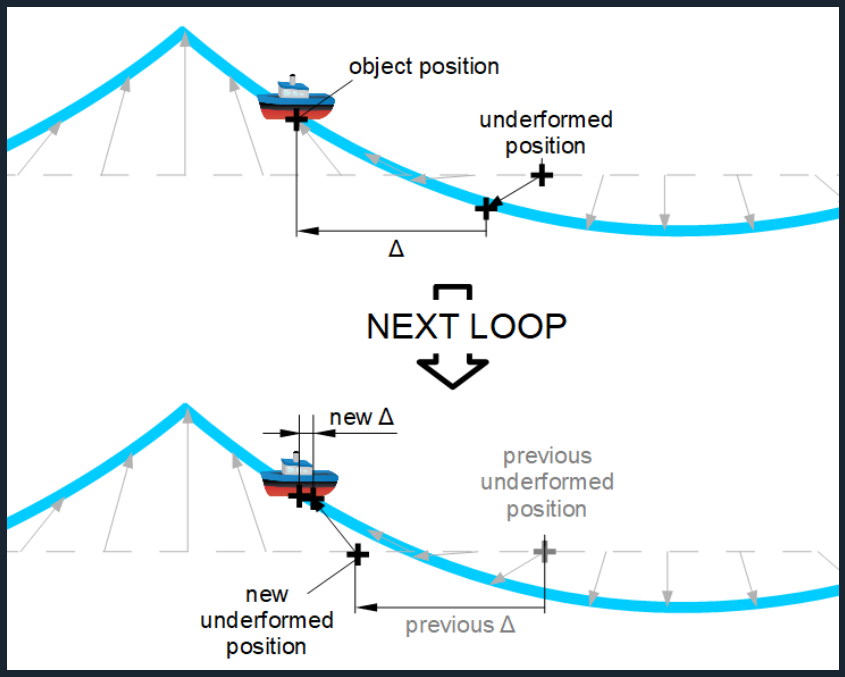
How the water is detected is non obvious since the wave generates both vertical and horizontal deformation. If we compute the wave under an object's position, it is likely to return a point that is too far from the object due to the horizontal deformation, especially for large waves where this deformation can be of several meters.

It is not required to know how Storm Breakers deals with that if you use its physics, but if you wish to use the Get Water API, you might need to understand it because it's not straightforward.

So if we compute the water deformation to an object below or above a point where we want to get the water height, it is likely to return a position that is not below or above the object. We call the "undeformed position" the water position before the wave computation, and the "deformed position" the water position after the waves apply their deformation. The undeformed position is where we want to compute the water height, speed and normal, the deformed position is where we want the water to be above or below the object for which we want to get the water height, normal and speed. The vertical alignment between the object and the deformed position must be the maximal because the gravitational pressure evolves in this direction with the depth.



What Storm Breakers do is to start with a variable that represents the undeformed position (generally initialized with the horizontal object's position), compute the deformation from this point, and get the position difference between the water deformed position and the object position. Then it adds this difference to the undeformed position variable so next time the waves are computed there, the deformed position will be closer to the object's position. This is done once or twice per frame in the game, the water data are then computed from the undeformed position. The trick is to loop continuously in the game to always get the deformed position as close as possible to the object's horizontal position without computing the waves too much per frame. It's like a real time root-finding algorithm.



When the difference between the object and the deformed position is superior to some value (currently an arbitrary value of 1 meter) then the method runs one more loop to try reducing the difference. Also to optimize the algorithm, some components (like the Water Interaction component) update the undeformed position with the velocity regardless of the wave in Fixed Update (waves are computed only in Update to reduce overhead).

This algorithm is implemented in C# for the physic and also in VFX graph to compute when the particles precisely hit the water. You can reuse this algorithm for your own VFX graph, for example rain particles that generate splashes when hitting the water.

So if you use the API to get the water height, velocity and normal, you need an internal Vector3 variable to save the undeformed position and pass it as reference to the method Ocean.GetWaterHeight, this method will then update the variable so the next frame it will be closer to the actual water height. To initialize the undeformed position, set it at start to the world position of the object.

Also when using the Ocean API, you need to save some of the results that are shared between GetWaterHeight, GetWaterVelocity and GetWaterNormal. This makes the use of the API not intuitive, but it is required to get enough performance. Computing the waves on the CPU can take quite some time as it is done a lot of time per frame.

Water velocity and normal

Computing the water velocity is very important as many effects depend on the relative velocity of the object onto the water (dynamic pressure, particles generation and audio), and waves create water velocity. To compute it, the wave deformation is computed once more at the undeformed position, but at a different time. The difference is divided by the delta time and results in the water velocity. This velocity is then attenuated with the depth to try simulating the real world wave attenuation (water doesn't move much under two times the wavelength in depth). Because the velocity is computed not knowing which wavelength is present at the undeformed position, an equivalent wavelength is deduced using the water velocity itself and the dispersion relation (the formula that defines the wave speed function of the wavelength). The depth is compared with this equivalent wavelength to make the attenuation factor.

To compute the water normal, which can be needed in some special case where a low number of water sampling if performed (Sphere Water Interaction component for example use the water normal), the wave deformation is computed 2 time more at different location, then the the normal vector is computed with the 3 know water position.

Computing the wave deformation being quite resourceful, getting the water velocity and especially the water normal lead to an CPU overhead. See the performance section of this document to know what features are planned to be added to optimize the computation of these vectors.

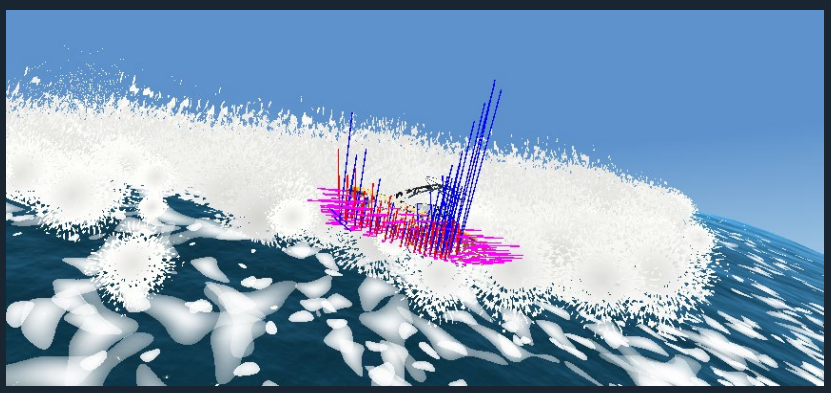
Breaking waves force

Not only Storm Breakers includes an innovant particle system and procedural audio for the breaking waves, it also computes the additional speed of the water being thrown at the crest and the torque it creates.

The breaking additional speed is computed using a similar value to the one triggering the particle generation and audio (compression), but a bit different to try propagating the extra speed further than the crest. This speed is proportional to the wave celerity, each system adds its own breaking speed at a given point. Its value is zero when there is no breaking at this point. This additional breaking speed can push small watercraft in the breaking wave, or create larger sea foam bursts when a wave hits a ship.

Then there is a breaking wave torque that is computed using the additional breaking speed. The direction of the torque is the one of the vortex that would be generated by the lip of the breaker, meaning perpendicular to the wave and rolling in the wave. This torque makes small watercrafts roll in the breakers, making them more dangerous.

Currently the breaking wave torque is only implemented in the Water Interaction component. The torque is computed when reading the water speed for each triangle, then it is multiplied by the triangle area and added to the rigidbody. It is activated a bit above water (arbitrary 0.02 times the breaking wavelength in height) to simulate the thickness of the sea foam turmoils. The torque is attenuated with the depth by the same factor that attenuates the water velocity.



The breaking torque can be visualized in magenta in the Gizmo mode.

You can change the intensity of the torque force and additional breaking speed in the Ocean Controller component. So for a storm survival gameplay you would make strong waves that really need to be avoided, while for a more casual game or any simulation not focused on the waves you may reduce it or even remove it by setting their value to zero.

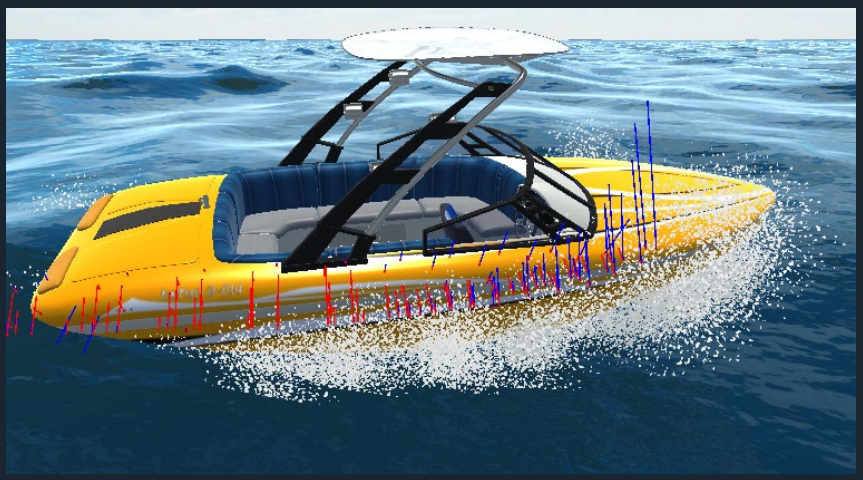
Buoyancy physics

Storm Breakers provides the buoyancy effect so you can make objects float in the waves. Here is a description on how it works.

Physics computation

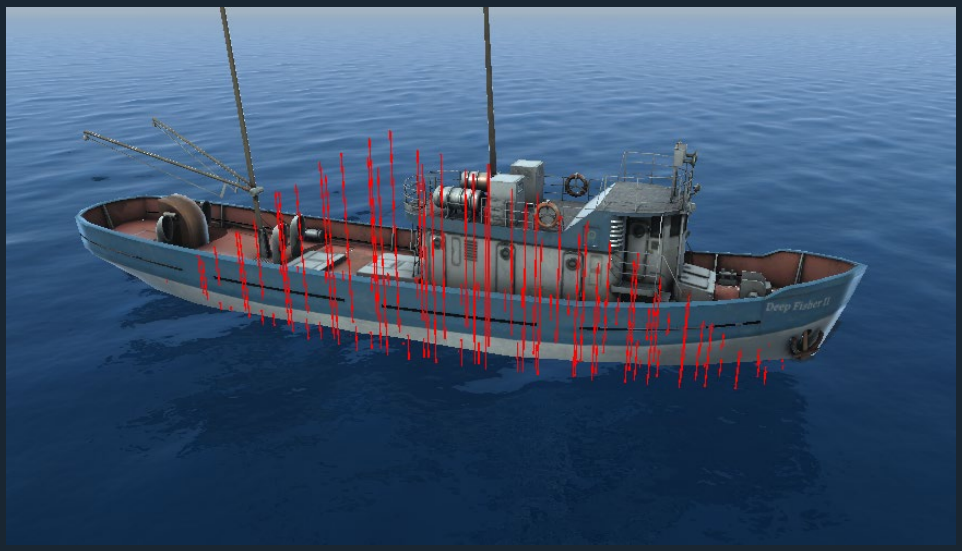
The way the buoyancy is computed is by calculating the water pressure acting on each triangles of a simulated mesh. The pressure comes from both the gravitation (called here static pressure) and the relative velocity (called here dynamic pressure). The pressure on each triangle makes an Unity force that acts on the rigidbody, the forces are normal with their triangle and not vertical.

The numerical values of the force are in the International System unit (meter, kilogram, second) following scientific formulas. The static pressure is proportional to the water density (1000kg/m3) depth and gravity (10m/s²), while the dynamic pressure is proportional to the square of the normal velocity of the triangle relative to the water, the half water density and the drag coefficient. The pressures are summed by keeping only positive values (fluid cannot make negative pressure on a surface), and then transformed to a force by multiplying with the triangle area and directed in its normal. The force applies in the middle of the triangle.



In red the static pressure forces, in blue the dynamic pressure forces.

The fact that the forces are not vertical can be non intuitive, but it's the most realistic way to code buoyancy. This is also how surfing physics happens. Yet on some very calm water it can lead to an issue : since some triangles can be nearly inside water while others are nearly outside, there might be a small asymmetry in the forces, making the object moving by itself. To solve this, the water interaction component provides a way to keep only the vertical components of the forces.



Orient the force vertically only in flat waters and non moving watercraft.

The forces ares computed in fixed update as it is recommended by Unity, but the water computation is done in update being more expansive (Fixed update can be called many time per frame if the previous frame was long, leading to an even longer frame and so the next fixed update will be called again many time. It is better to leave the minimum in this function to avoid having spikes of CPU overhead).

Wake wave

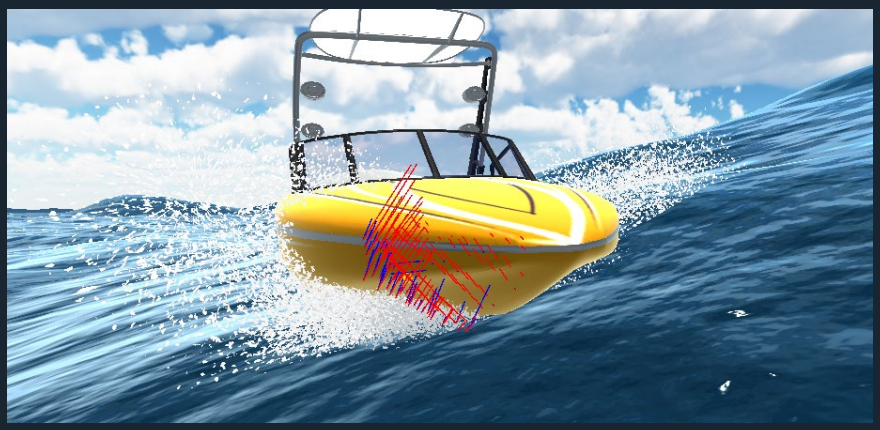
Storm Breakers physics provide a simplified yet very useful approximation of the wake wave, that is the wave generated by the object moving in the water. This wake wave is virtual and cannot be seen on the water surface because of the way Storm Breakers is coded. Its forces on the object are computed though, and it provides an advanced simulation feature to simulate cornering and surfing attitude.

The way it is coded is by increasing or decreasing the water depth to the height of a wave moving at the relative speed of the triangle to the water. When the relative speed moves toward the exterior or the triangle, depth is increased and the triangle might get virtually submerged where it shouldn't. Conversely the depth is diminished when the relative speed moves away from the triangle exterior and the triangle might emerge where it shouldn’t. The increase or decrease of depth also changes the static pressure acting on the triangle which in turn changes the maximal limit of the dynamic pressure, making this effect even more impactful.

In the flowing screenshot the wake wave is disabled, more static pressure acts on the upward side of the hull due to the slope of the wave. This will make the speedboat align to the slope of the wave and then slide down.

No wake wave here.

In this same screenshot, the wake wave simulation is activated. Note how the forces on the upper side decrease while the ones on the lower side increase. This is because sliding down a bit the slope will generate a wave on the lower side and a trough on the upper side.This will lean the speedboat in the slope and generate a side force strong enough to maintain it in its trajectory : this is realistic surfing mechanics !



Wake wave activated.

The same thing happens when cornering. In this first screenshot the wake wave is deactivated and the speedboat overturns :

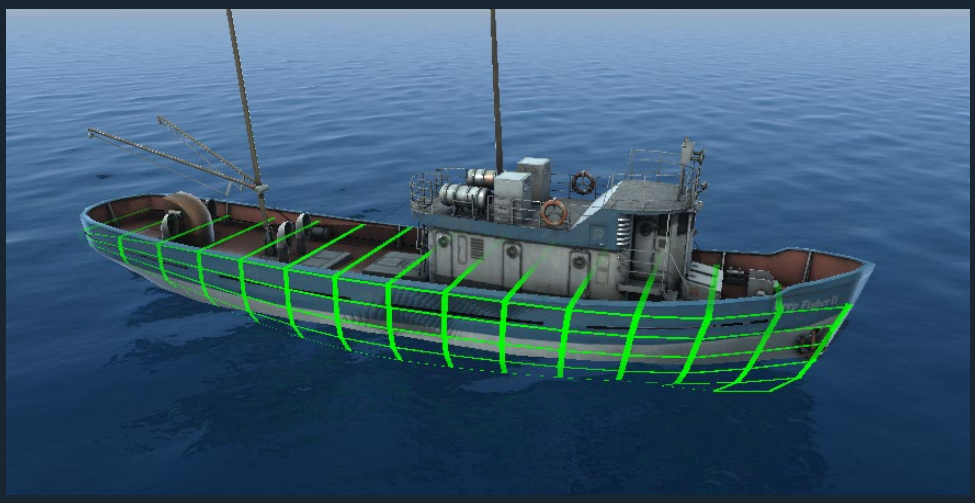
No wake wave.

In this screenshot the wake wave is activated and the speedboat makes a better curve :

Wake wave activated.

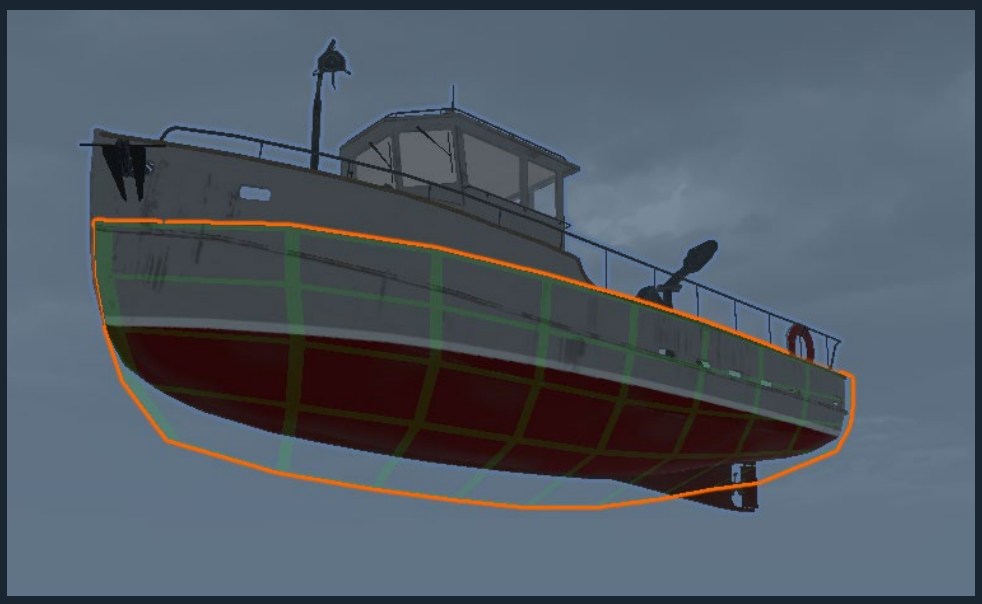
Simulated mesh

The buoyancy requires the use of a simplified mesh to be simulated. Having too many triangles in the simulated mesh will lead to an overhead by excessive computing of the ocean and applying a lot of Unity forces. This simplified mesh can be the one used with the mesh collider. For boats and ships, Storm Breakers provides a tool to procedurally generate the simulated mesh so you can tweak the number of triangles as well as other things.



A visualization of the simulated mesh with the hull generator helper.
Note how it envelops the railing to generate the particles when it hits the water here.

The simulated mesh doesn't need to accurately shape the objects it simulates. It is actually recommended to take a bit of liberty to have more control on the buoyancy dynamic (see next section). There is a limit to this, the particles are also generated according to the triangle position, so you cannot make too much change.



A simulated mesh with more depth is a way to reduce rocking while keeping good particle positioning.

For the Water Interaction component, all the triangles of the simulated mesh will generate static and dynamic pressure while underwater. You cannot currently segregate some triangles not to generate static pressure like you would for the railing of a boat. For this reason it is recommended to envelop the railing with the simulated mesh to generate the particles and dynamic pressure when hitting a wave. The issue is when capsizing, because the mesh needs to be convex there must be triangles joining the simulated railing, and those triangles will make a buoyancy force where there are no actual surfaces to do so, so the

capsized boat will float too high. There are currently no fixes for this, but might be improved in future updates.

In the Water Interaction component, the simulated mesh shouldn't need to be convex for the Storm Breakers engine to work. Yet because it is currently linked to the collider mesh that needs to be convex so the rigidbody can compute the inertia and use it as a collider, it is eventually quite mandatory to have a convex mesh. This is much a constraint as *every* ray from vertex to vertex should be *within* the mesh, limiting many kinds of shape.

Special buoyancy

Storm Breakers does not yet provide an efficient way to simulate very low poly meshes (like the built-in cube with its 12 triangles), nor flat meshes (like a surfboard) nor linear objects (which could be useful to make the buoyancy of a character's rig). If you have such a need in your game, you should currently use a complex mesh that closely shapes the object. Such features might be added in future updates though.

Watercraft dynamic

The physics in Storm Breakers is coded to be as realistic as possible and you may wonder how to tweak the behavior of your watercraft. There is no explicit tweak but rather a possibility to use real world phenomena to adjust the waterfract behavior to anything you want.

Watercraft stability and sensitivity

When speaking about stability, there are in reality 2 different things to consider :

- The stability limit is the maximal tilt angle from which the watercraft can recover its initial position.
- The sensitivity is the response time to which the watercraft will recover its initial position after being tilted.

Because a watercraft will always tend to align itself to the slope caused by the waves, the sensitivity plays an important role : it defines how fast the watercraft will align to the wave. If it is slow enough it might not even be impacted by small waves, but with high sensitivity the watercraft will rock a lot into high stepped waves. So in this document we speak about the wave sensitivity as a flaw.

Note : In naval terms we rather speak about primary stability and secondary stability, which are respectively the stability at low and high list angle. In this document we speak differently by talking about the wave sensitivity because for a game where we want to see impressive waves, too much realism in such a sea state would lead to too much seasickness for the player. So it's important to know how to tweak the watercraft wave sensitivity.

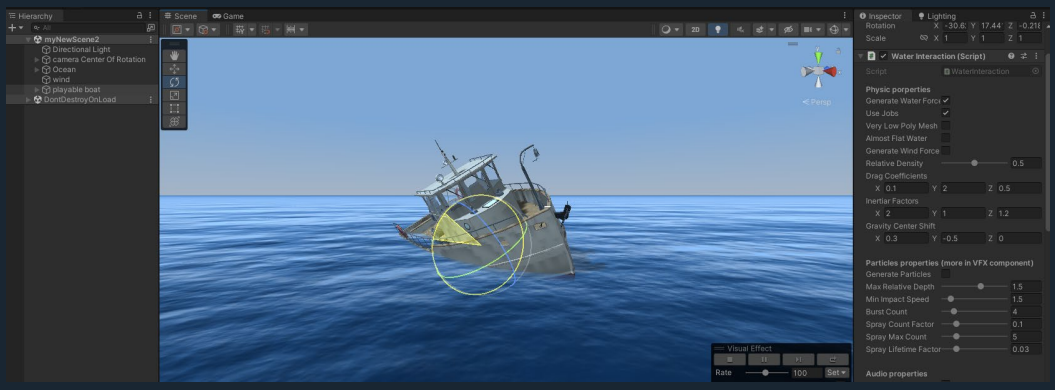
Roll dynamic

To adjust both the roll stability limit and the roll sensitivity, you can play on the mass repartition, the hull cross section geometry and add some components. Here is a description on what each variable can change the behavior of your watercraft.

Center of gravity

The first variable you might think about is the center of gravity. You might think that lowering it will increase the roll stability but lowering too much the gravity center also increases the roll wave sensitivity which makes your watercraft rocks more under the wave ! (and so looking less stable).

The position of the center of gravity has a lot of impact and shouldn't be set too far high or down. As a general rule, what matters in terms of stability and wave sensitivity is the speed at which the watercraft returns to its original angle : set the gravity center height so the watercraft returns reasonably slowly to its initial position (a second or so depending on the watercraft size).



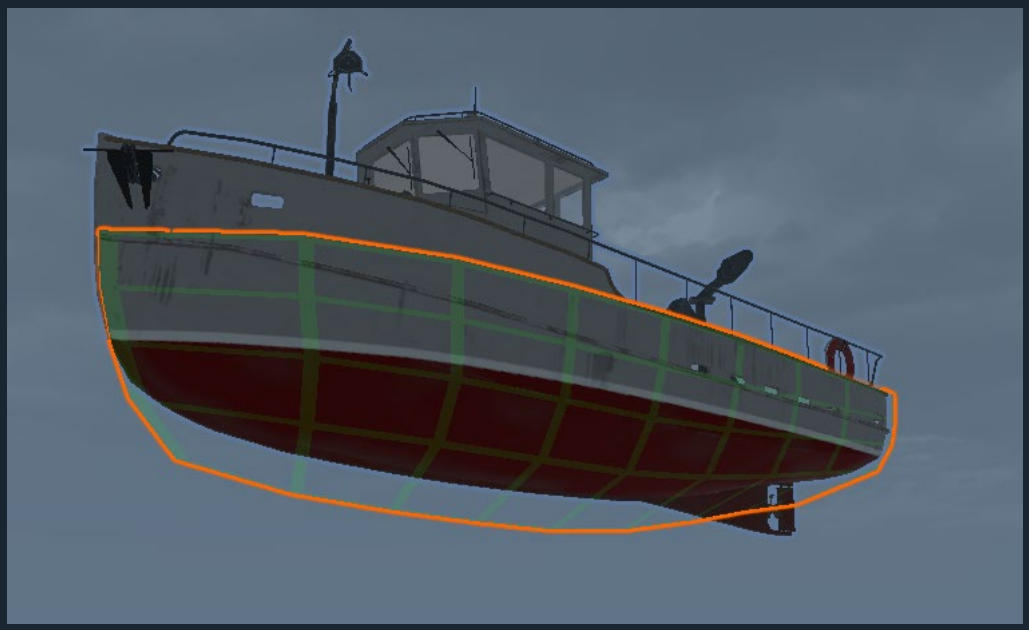
Tilt the watercraft and increase the gravity center until it right itself reasonably slow.

Though, this rule cannot be applied to sailboats. With the external force coming from the wind on the sails, the center of gravity needs to be set very low to avoid too much list. But sailboats have a keel, which is another way to solve the issue described below.

Inertia and density

Inertia plays an important role as well. By increasing the rotational inertia or the density, the watercraft will return more slowly to its original angle and so the inertia center can be set lower to increase the stability limit without impacting the wave sensitivity.

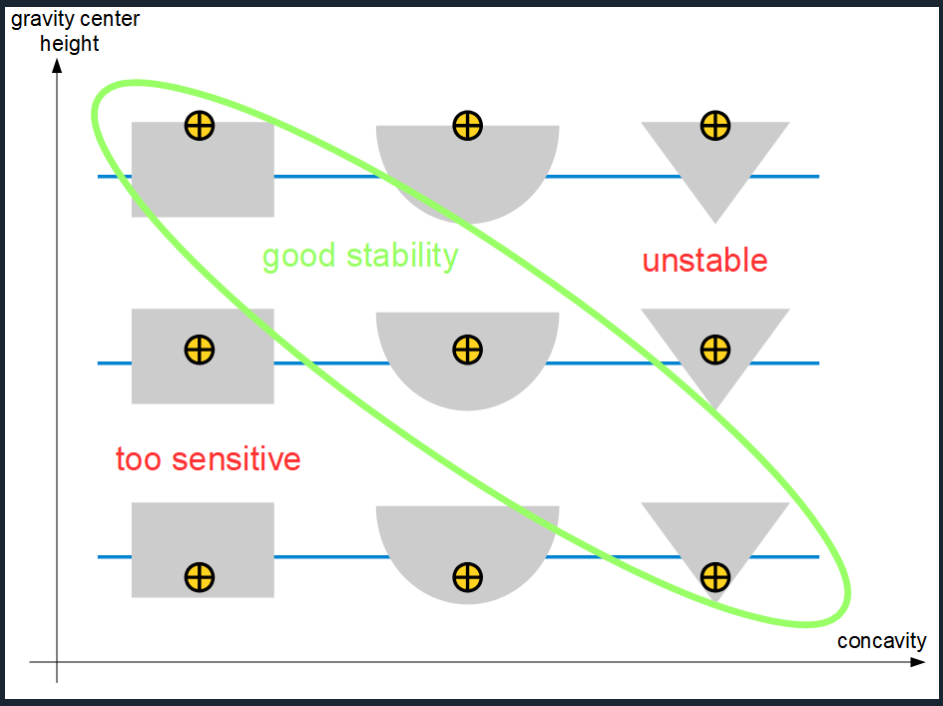
In the Water Interaction component you can tweak the roll inertia tensor to use this tweak. This can work until some point where it becomes visually unrealistic in some situation, so use this with moderation. One other way is to increase the hull depth, you'll need to increase the density too (and the controller forces and perhaps the gravity center height) and you'll have a less wave-sensitive watercraft.


Increasing the depth of the hull reduces the wave sensitivity a lot.

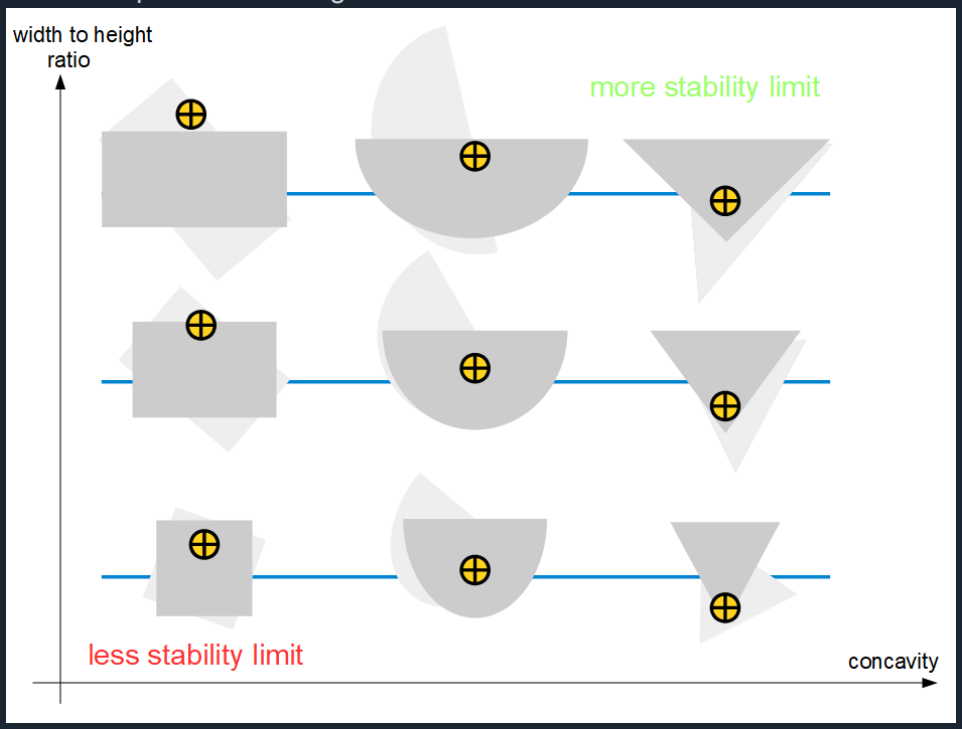
Hull geometry

The hull geometry, in combination with the gravity center height, plays a very important role in the watercraft stability.

If we consider the the sensitivity at low list angle, then this diagram shows how to setup the gravity center depending on the hull geometry :



If we consider the stability limit, then this other diagram shows how to choose the hull shape whether you want your watercraft to capsize at low angle or not.



Note : As you can see, the V shaped hull would be in theory the most seaworthy one. While this is indeed the case in Storm Breakers, in reality the V shaped hull has more drag than the round one, and also it requires a lower gravity center which in return needs to increase the watercraft weight to offset the hull weight. So in reality the V shaped hull is much less performant, but in Storm Breakers you don’t need to be limited to such things and the V shaped hull is a good choice.

If the mechanics of your game makes the watercraft capsize on a player mistake (for example when trying to survive in a storm), then you should try to narrow the hull and choose the correct shape : a square shape will make the watercraft abruptly capsize which leave few opportunity for the player to recover a large heeling, a more round shape will progressively lose in sensitivity so the player can anticipate before it's too late to recover.

Water fins

One very useful way to reduce the roll of a watercraft is to add stabilizers on the hulls, just like many real watercraft do (sailboats keels act also as rool stabilizer in addition to prevent drifting) :

Credit : wikipedia

They make drag forces that oppose the roll movement, thus reducing the effective sensitivity without any impact on the stability limit.

You can easily simulate them by adding a quad on the edge or bottom of the hull with the Storm Breakers Foil component attached too. The lift coefficient will greatly increase the effectiveness of those features with the speed increasing.

In the sailing example, an invisible triangle with the Foil component acts as roll stabilizer in addition to reducing drift.

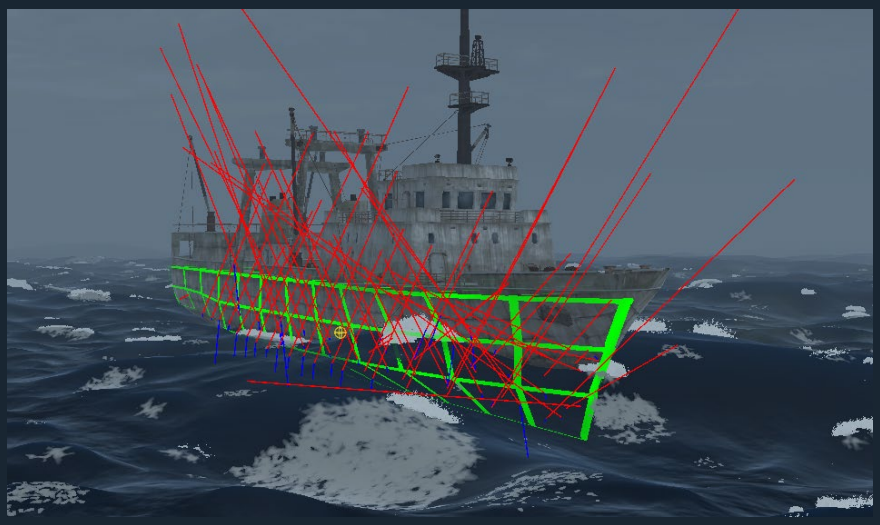
Pitch dynamic

When speaking about a ship or a boat, the length in general greatly exceeds the width and the hull depth is almost constant along the length. And while a hull can reduce its roll sensitivity with its cross section shape, the pitch stability cannot be much modified and lead to a high pitch sensitivity into the waves. This

is why pitching of boat and ship is in general more visible than roll and it is the same in Storm Breakers. But with a long length the pitch inertia is high and the pitching is in general slow enough not to reach high-frequency causing seasickness (see Gameplay overview section of this document for more info about seasickness).

Still, there are things to consider for the pitch dynamic. First the inertia factor in this axis can be adjusted in the Water Interaction component, but you shouldn't increase this much higher to 1 (which results in the inertia of a full body in its length) or you will see non-natural pitching. Even with masts or high buildings not computed in the inertia by the rigidbody, it's rare that such features actually increase the whole inertia because they are quite light compared to the pitch inertia of the hull itself.

The bow shape plays an important role when hitting big waves. An enlarging bow will create a lot of reserve buoyancy and makes the ship bounce on the wave, leading to high pitching. To be noted that high pitching in the waves slows down the watercraft considerably, and sometimes with insufficient speed the watercraft can surf backward in a big wave so it's better to have a narrow bow to prevent this.



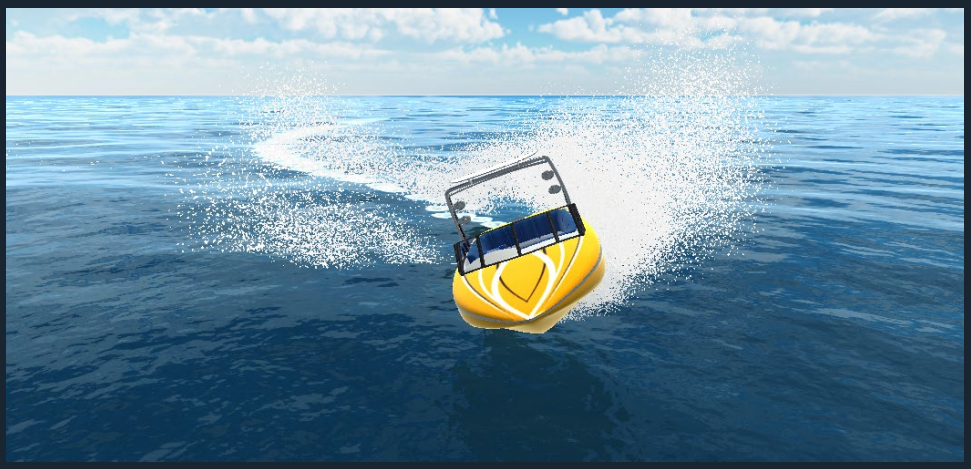
With an enlarging bow, a lot of buoyancy is generated forward when hitting large waves.

With a narrower bow there is less reserve buoyancy and the ship pitches less.

Note : in real world hull design, the bow is often shaped with enlargements so as to reduce the water coming on the deck when hitting a wave, but this isn't much of an issue in a game. So again you rather change the simulated mesh to make a narrow bow that prevents surfing backward in a wave.

High speed dynamic

Storm Breakers allows you to simulate high speed watercraft. The Epic Surfing scene example shows you how this can be achieved.



Planing

At high speed the dynamic pressure becomes more important than the static pressure and phenomena happen. First the hull will have a tendency to generate a lift with the increasing dynamic pressure when the equilibrium of forces make the bottom of the hull have a positive incidence angle. This is called planing and all the real-world speedboats rely on it to move faster. This also happens in Storm Breakers though this doesn't affect the speed much because the drag forces are already lower than they should be at low speed (the wake wave is not completely simulated).

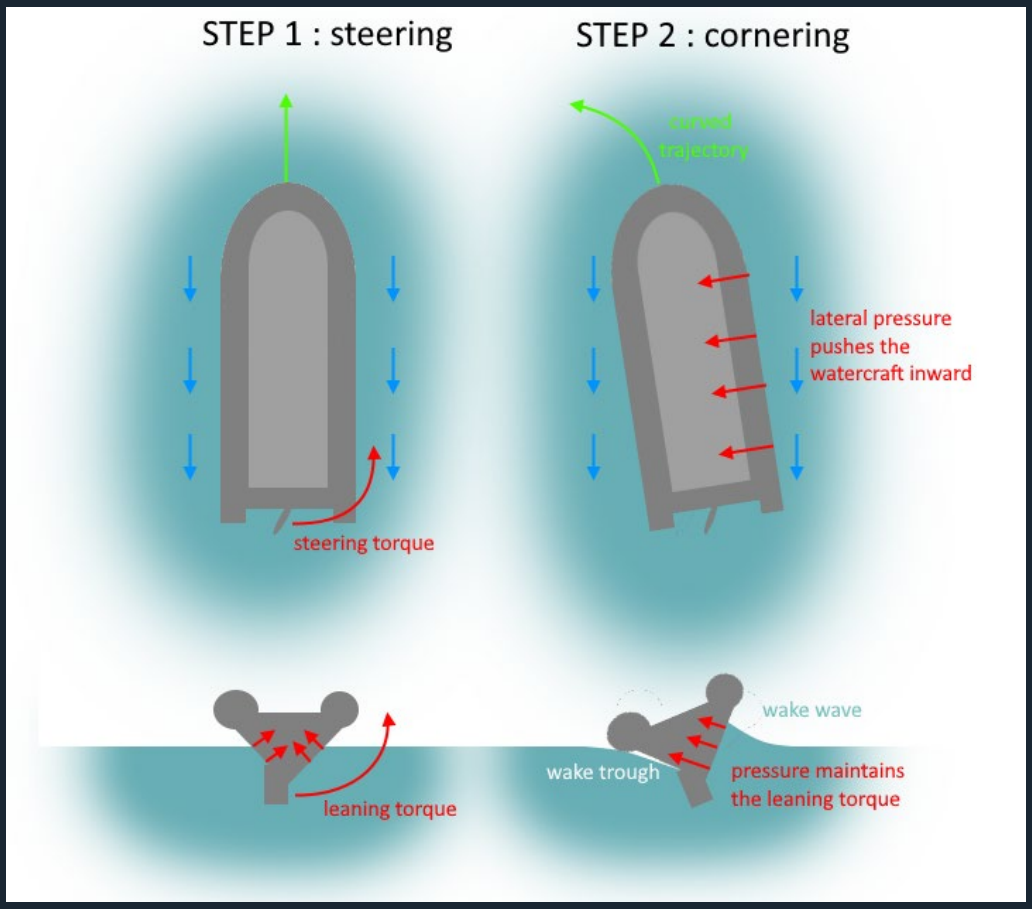
Note how the dynamic pressure (blue lines) rise up the speedboat.

Cornering and surfing

The difficulty in cornering is to push the watercraft inward the curve. If there is no such force, then the watercraft will overturn and lose all its speed, making cornering impossible.

With the rudder set at the bottom aft, the force it will generate will make a steering moment and a leaning moment in the correct way, but this force won't help to create the required inward force (It actually goes against it, this is why in a car the steering wheels are in the front and not the back). So how do real

speedboats corner ? They rely on the wake wave they generate when setting the hull across the trajectory. See the following diagram that explains this physics.

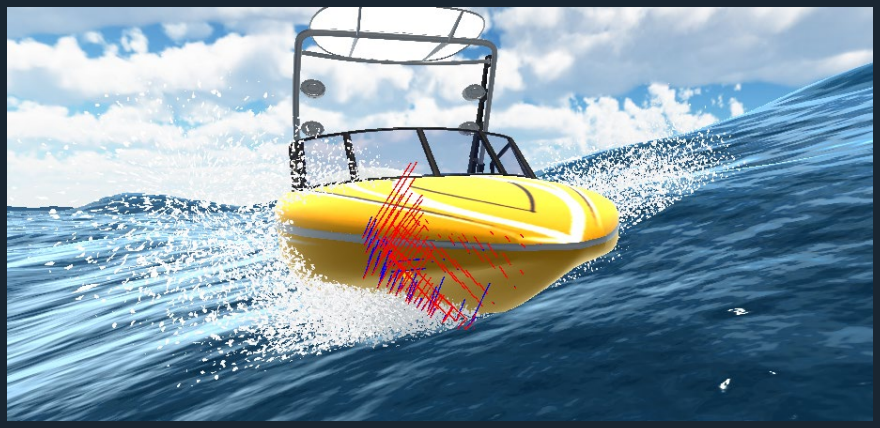


Note how the V shaped hull matches with the wake trough and the wake wave : on one side there is almost no more pressure while on the other the pressure builds up a lot in an almost horizontal direction. The large dissymmetry in lateral pressure makes the net force that pushes the watercraft inward the curve, making a curved trajectory without losing much speed. The wake elevation and trough also create a leaning torque that maintains the watercraft leaning in the curve. If there wasn't the wake wave the waterfract would tend to lean outside the curve because the pressure acting lower than the gravity center, which would reduce the cornering attitude.

For the surfing mechanic the same effect happens and allows one to ride across a wave. The hull (or board) is set across the trajectory either by a controlled force (rudder, surfer) or naturally with the slope making the waterfact sliding down the slope. Then the wake wave builds up on the lower side while a trough is created on the upper side, making a net force that prevents sliding down too fast and staying longer in the wave.

The effects of the wake wave can be simulated in Storm Breakers allowing for cornering and surfing mechanics.

The shape of the hull is very important for cornering and surfing and only a V-shaped hull will be performant enough to do so. The narrower or deeper the hull is, the more efficient it will be at cornering and surfing (this requires a lower gravity center but it can easily be modified). A flat hull like a surfing board can corner and surf only with a high additional leaning torque that maintains the lean, more than a classic rudder force would do. Indeed such a hull is not naturally stable in a curve, for a surfboard this comes from the weight of the surfer leaning on the board.

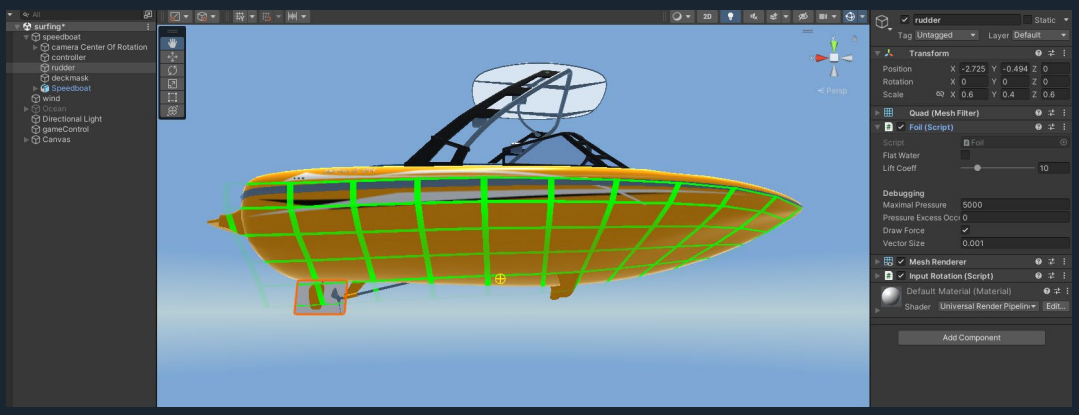


The pressure acting on the speedboat while surfing. Note the lateral asymmetry.

The pressure acting on the speedboat while cornering. Note the lateral asymmetry.

Trajectory stability

When navigating fast, we like to keep the course steady and not overturning at any mistake. For that we can do that with a fin set at the aft (a simple mesh with the foil component attached to), with a gravity center set more forward (and not aft as we may expect) or/and with a hull that is deeper aft than forward. All those effects will make sure there is more dynamic pressure behind the gravity center when the hull is across its trajectory, making an aligning torque.



A simulated hull deeper aft, an aft fin with the foil component (also acting as a rudder) and a gravity center set a bit forward, all to improve trajectory stability.

Recap

Now you know all the parameters you can play with to adjust all the way you want a watercraft dynamic. Here is a brief recap by using possible desired watercraft dynamics as example :

- Low seasickness, no capsizing :
  - Prefer round shaped or V shaped hull for high stability limit
  - Moderately low gravity center for some stability limit and not too much sensitivity.
  - A large keel fin to reduce roll
  - Inertia increased a bit to slow down roll and pitch
  - Narrow bow to reduce pitching.

- Fast watercraft :
  - V shaped hull required
  - A foil/rudder behind to stabilize the trajectory
  - Low density to corner better
  - Small size to keep some response time

- Capsizing under big waves only
  - Round shaped hull to avoid brutal capsize.
  - Low width to height ratio to foster capsizing.
  - Low sensitivity with a fairly high gravity center, just enough to have a bit of stability on flat water.
  - Enlarging bow to push backward the hull and foster capsize.

Now you see how important the hull shape is, this is why it is recommended to generate the simulated mesh using the hull generator. With this component you will easily change the shape hull even if it doesn't fit much with the 3D model, and you can tweak many dynamic behaviors of your watercraft. The speedboat in the Surfing Giant Wave example scene is a good example on how the simulated hull has been modified against the 3D model to improve cornering and surfing attitude. Also keep in mind that some ship or boat 3D models don’t have a correctly shaped hull and they might not be naturally seaworthy.

Wind

In Storm Breakers, the wind is part of the simulation. It affects mainly the visual effects and the water surface, but it can also affect the physics : the Foil component can act as a sail with sufficient mesh area, the Water Interaction component can also make the object drift with the wind force if asked so.

In the sailing example the boat is actually powered and heeled by the wind (blue lines), note how the keel prevents drifting.

Wind implementation

Currently the wind evolves with the height to simulate the boundary layer effect. Some simple turbulences are also implemented, future updates might make the turbulences affect the water surface to improve the sea rendering.

The wind can be set up with the wind controller object, if not present the default values will be taken.

Currently the wind doesn't affect the waves, but only the ripples. Depending on the user feedback, future updates might include an advanced wind/ocean controller that will automatically set the waves properties realistically according to the wind and its evolution.

Sailing

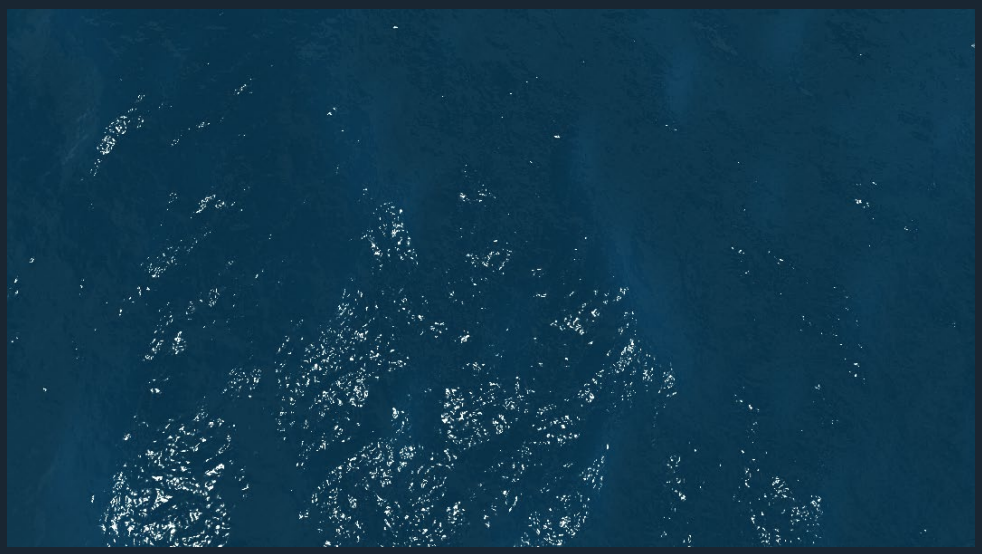
As explained it is possible to simulate sailing using the Storm Breakers package feature. Yet, because this is still under development, this document doesn’t provide yet more info on how to correctly set up a good sailing simulation and you may have to wait for more features before making a sailing game using solely Storm Breakers.

Rendering overview

The rendering in Storm Breakers is designed to be as realistic as possible while keeping good performance. Future updates might increase the global definition to better render the ocean and the effects.

Water rendering

Here is briefly described how the water is rendered.

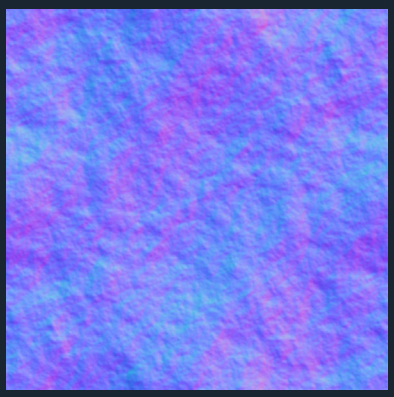


Waves

The rendering of the waves is 100% done in the graphic process unit (GPU) through a vertex shader, this vertex shader having the exact same math as the one used in the CPU to compute the physics. The normals and many effects are computed per vertex to reduce the overhead in the pixel stage.

Ripples

To improve the water surface, a ripple system is rendered in pixel stage using 2 normal maps moving in opposite directions at different speeds. The normal map used come from this link :
https://blenderartists.org/t/animated-water-normal-map-tileable-looped/673140



The ripples are erased on the back of a breaking wave to simulate the wave superiority as explained in the wave model overview section of this document, but this effect is not direction-dependent (future update

might try to include this feature). This makes a smooth water surface like we can see behind real breakers, this also helps break the monotony of the water surface.

Relying on a normal map to render strong ripples isn't good because we see as much the front of the ripples and their back that should be occluded by the front. To try to reduce this issue, the current ocean shader attenuates the ripples' normals when they get at too much grazing angle.

Future updates might include a thinner water mesh and more small wave layer to improve the rendering of the water details. Also parallax mapping of the normal texture might be considered.

Reflections

The water reflections are done using the built-in Fresnel effect of the shader graph PBR master.

By default the water reflects the skybox only, but Storm Breakers provide an approximate way to reflect dynamic objects without using the Screen Space Reflection (not available in URP). To do this, a dynamic reflection probe is set at a mirror position of the camera relative to the sea level. This reflection probe captures most of the scene (except the water) at runtime and generates a cubemap texture that is automatically used to compute the reflection.

Using a reflection probe instead of SSR can make a bit of improvement on the water reflection because some geometries become visible, but mainly some defaults. First it's quite expensive for the GPU as it acts as a second camera rendering most of the scene, to solve this you should reduce its resolution and how frequently it is updated, but you'll have much less quality. Secondly it's not accurate enough in the waves, this is why there is a tweak factor to reduce the reflections angle. Though, on a glassy water and with a good hardware it is possible to render nice reflections :

When there are more waves, the approximation of this technique makes reflection *behind* the object (the ones we can see between the sails in the following screenshot), which is of course wrong. To offset this a bit, the reflection probe automatically increases its depth to reduce the reflection behind, but this also reduces the correct reflection so it's not perfect.

Future updates of Storm Breakers might upgrade the package to the High Definition Render Pipeline, then the screen space reflection will be available and most of the issues using a reflection probe will be solved.

Horizon

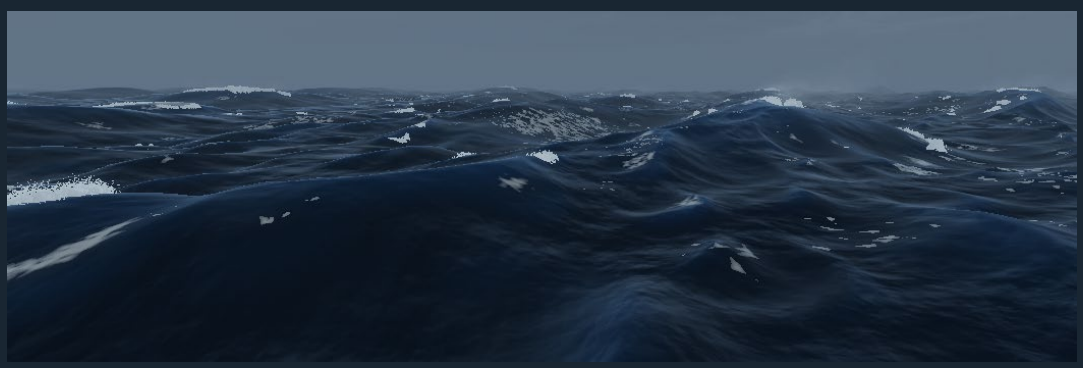
The provided mesh that renders the ocean has a reduced triangle density at the horizon to reduce the number of vertices computed. To offset the lower density, Storm Breakers replaces at the horizon the wave from its model by a different wave's normals computed in pixel stage. Those waves are called fake horizon waves because they don't precisely match with the real ones of the model.

Currently the fake waves are computed by a single non-moving perlin noise. The trick is using the pixel's rapid world position change as a way to animate the fake wave. With high screen resolution and slow camera movement, this effect can be insufficient though. Future updates will try to improve this effect based on user experience.

The intensity of the fake horizon wave is automatically set to the maximum intensity of the wave systems, but sometime this is not relevant (for example in the scene surfing giant waves, the maximum intensity of the wave systems is very high because of the large breakers, but at the horizon we want to see the background waves with low intensity), so you can tweak their intensity with a value in the ocean material inspector.

SubSurface scattering

Subsurface scattering is the effect of the water being lit by transparency. Storm Breakers provides an approximation of this effect in the waves : when the wave gets sharp, at the crest the material emits a bit of light corresponding to the water being lit.



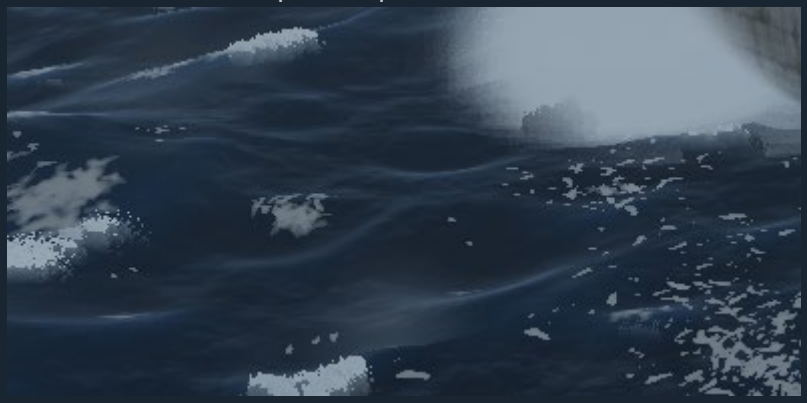
You can tweak the intensity of this effect through the ocean material inspector. High value can lead to surrealist rendering.

Transparency

In deep water it's hard to see through the water because almost no light comes from below to lit underwater surfaces. Still, the Storm Breakers package provides transparency for the water to make smoother water rendering.

There are big issues with water transparency and particle transparency though :

- Since VFX graph particles have trouble being ordered, breakers' sea foam burst can’t be transparent and so they must be drawn before the water in the opaque queue, and so we can see them through a transparent water. To prevent this you must set the water to opaque by setting its minimal transparency to 1 in the material inspector when there are breakers VFX in the scene.
- The soft particles used to render sea spray don’t work on the water because its scene depth cannot be read being rendered in the transparent queue. There is no fix for this.



The issues with water transparency

Underwater rendering

Storm Breakers provide a very basic underwater effect in case the camera would go underwater (works only in play mode). You shouldn't rely much on it to render realistic underwater scenes.

This effect does not segregate precisely the pixels in the screen that are actually underwater and the ones that are not, it's a global fog effect that activates as soon as the camera is near the water surface or below. This will be improved over future updates but precise segregation is still out of reach.

The rendering of the water surface from below is currently not physically based, it's rather a trick using the reflection with a mirrored skybox. With this trick we see the sky color at grazing angle and the water color at normal angle while it should be the opposite. So the reflections are inverted, still it provides somehow the right colors. So currently the bottom of the skybox affects underwater rendering. This will be soon improved in future updates when finding an efficient way to get the sky color that we see by transparency, the underwater normal will then be inverted and the water color will be rendered below so the surface reflects it.

Also the particles are not culled underwater. Again this is a feature that is to be expected in future updates.

Water clipping

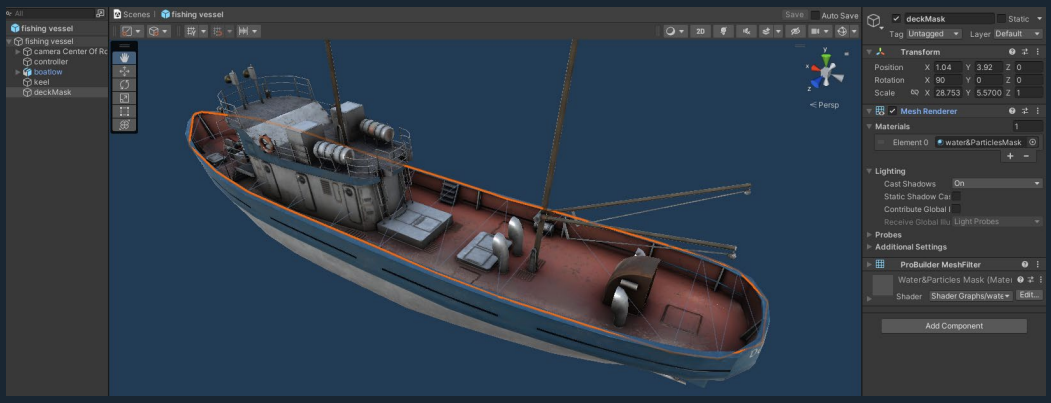
Clip the water inside a boat using one of the mask materials on a quad.

Storm Breakers package provides a material to mask the water (water&ParticlesMask.mat) This is very handy for keeping open boats exempt of water. Though, high crest waves might go above this mask from time to time and there is currently no fix for this.

Currently the implementation of this mask is a simple fully transparent shader graph writing in the depth buffer and rendered before the water. The problem is when there are gunports on the hull, nothing will be drawn afterwhile within those openings (not even the skybox), leaving an unset color. Future updates will try to fix this issue by writing a more advanced shader that checks the scene depth before writing its own.

The mask must be fully inside the object you want the water to be clipped (meaning not above the railing for a boat) so as not to have the water wrongly clipped when seeing from a side view.

The mask material that clips the water also clips the particles. So as a general rule it is good to make a flat water/particle mask that enclose the railing of a boat. To build the mesh quickly you can use Probuilder.



This deck mask was built with Probuilder.

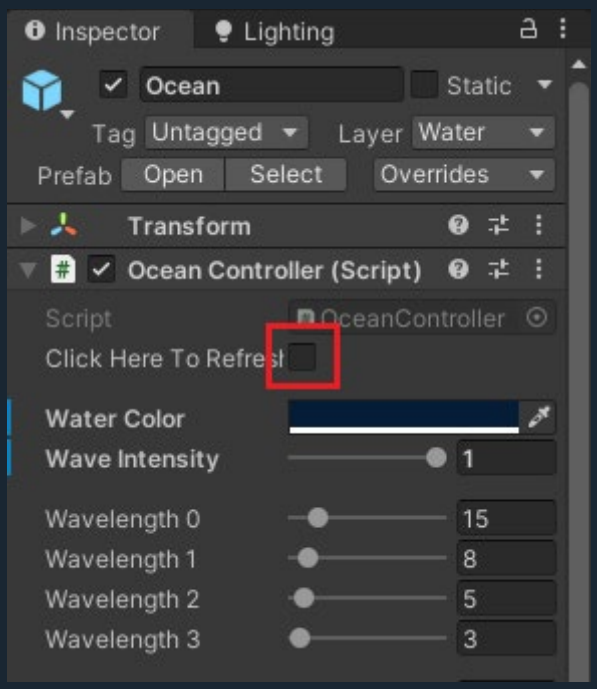
Visual effect rendering

Fake lighting

Most of the visual effects on the scene are fake lit by multiplying their color to the sum of the ambient color and the halved main light color (Halving the directional light color is the result of a tweaking). The reason is twofold : since those particles render small drops of water, the texture’s normal doesn't fit to this geometry and we wouldn’t want their color on screen to be affected from where we look at them. Secondly this greatly reduces the pixel stage GPU overhead when drawing large amounts of particles on the screen, which turned out to be a major bottleneck. Using fake lighting makes the particle not affected by scene shadows though.

In this screenshot with exaggerated lighting, the sea foam bursts are fake lit while the sea foam floats are not.

Making the sum of the halved directional light and ambient light is an approximation that can lead to different brightness with the floating particles in dark scenes. Also attempts on getting the ambient light color using skybox ambient mode failed, this is why the ambient mode must be currently set to either color or gradient. This addition is done in the Ocean Controller component (it acts as a general controller). To update the fake lighting in the scene you should trigger its start function by clicking on Click Here To Refresh :



Another drawback of the fake lighting is that it's more difficult to simulate light occlusion within a large burst of sea foam. Currently the color of the sea foam burst is darkened a bit at spawn and quickly returns to white in an attempt to fake the light occlusion. This effect will try to be improved over the future updates.

Transparency and sorting

Using many transparent particles lead to a problem of sorting them at draw to correctly have the nearest one on top of the farest one. The VFX graph package includes a way to sort them but it turned out not to be working properly. So it has been decided to set the sea foam burst from the breakers to the opaque queue and have them properly sorted. The drawback is that the water is rendered after (being transparent) and makes transparency on those particles.

Floating particles

Large amounts of various floating particles coming from the waves, the splashes and the propeller.

The floating sea foam and swirls are not rendered through the ocean surface but through particles that use the same vertex shader as the ocean. Here is the main reason why it is almost impossible to render wake waves on the ocean: while we could deform the ocean mesh by CPU script, these particles would not be affected by such modification, and it's impossible to code an unbounded number of additional wave in GPU because of its limitations (no loop, no conditional computing, etc).

Future updates might improve the rendering of floating particles using normal mapping.

Particles clipping

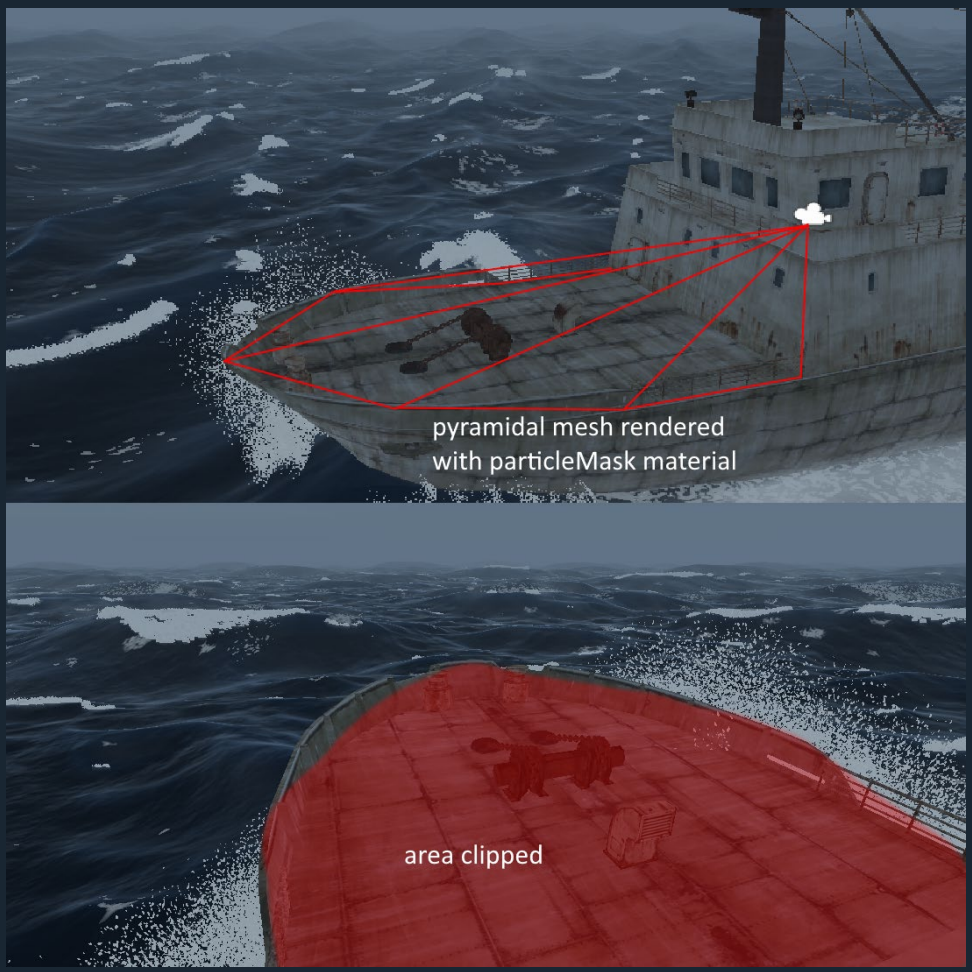
Storm breakers package provides a way to clip particles the same way it is done with the water clipping using a depth mask material (particlesMask.mat).

A quad set in front of the scene to clip the splashes particles.
The cliping of the breaker's sea foam burst is not currently possible.

Because of the way it is currently implemented, the water interaction component makes splashes particles appear sometimes through the deck of a boat, which can be very annoying. When this happens, you should first try to tweak the particle system (more outward speed) as explained in the visual effect overview section of this document. If the spot where this happens is localized, you may use the kill box provided by the particle system, but this is sometimes not enough to clip large particles (with the kill box, the entire particle is removed only when its origin penetrates the box). Then you may rely on masks, meaning invisible mesh with the masking material.

When this happens, try tweaking particles properties or use a mask.

The difficulty with masks is that they don’t act as volume and they clip everything behind. The trick for this issue is to make a pyramidal mesh that links the edges of the volume we want the particles to be clipped to the camera. This clips all the particles below this pyramid.



The pyramidal particles mask concept explained.

Overdraw

Overdraw visualized in a stormy scene.

Because there can be many transparent particles rendered in the scene (especially in stormy scenes), overdraw is known to be a main bottleneck in the Storm Breakers package. The GPU overhead happens for the pixel stage when drawing particles, so the solution is to have fewer pixels affected by particles on the screen : lowering the resolution, reducing the particles quantity or simply bringing the camera further back.

Future updates will try to reduce this issue by splitting the water interaction bursts to smaller particles and reduce their amount of pixels drawn.

Post processing effects

Attempts on setting up post processing effects (such as HDR tone mapping) during developments were unfruitful. Notably what happened in HDR mode is the transparent material having a darker tint that they should have. So if you want to use post processing effects in your project, you may have to use your own graphic skills to make them work properly.

Mobile platform

Storm Breakers is not designed to run on mobile platforms. The main issue is the VFX being computed by the GPU, this requires a compute capable graphic processing unit and very few mobile phones have this.

High Definition Render Pipeline (HDRP)

Currently Storm Breakers is developed for the Unity Universal Render Pipeline (URP) for a better versatility. Though there aren’t many pipeline-specific features used and the upgrade is planned to be done in future updates.

With the HDRP some features will be available and will increase the package quality : screen space reflection, floating origin, and perhaps more.

When upgrading to HRDP, it is likely that there will be a new package sold separately from the URP package. The philosophy might be different by increasing the number of wave systems rendered,thinner mesh density and more particles so as to match the high definition expectation and high end hardware capacities.

Visual effect overview

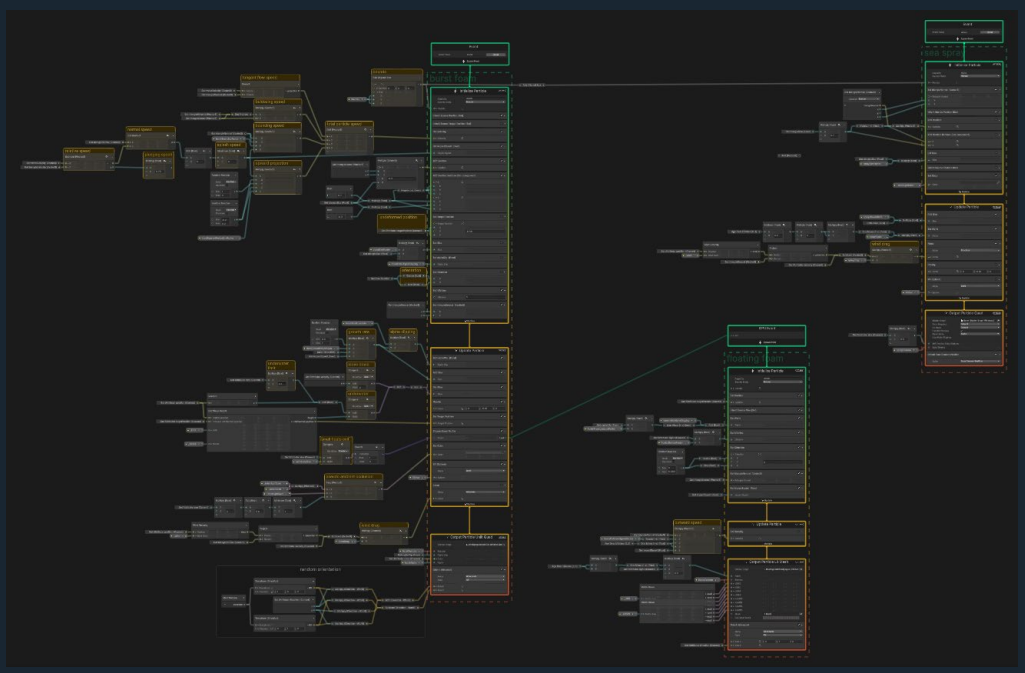
Storm Breakers package includes a whole water visual effect stack for improved visuals.

A large ship hitting a big wave and making a lot of visual effects.

VFX graph

In Storm Breakers, all the visual effects were developed with the Unity VFX graph package. This package makes the computation of the particles run on the GPU instead of the CPU, which greatly improves their performances and allows scenes with high visual effects. Though, this works only on GPU compute capable hardware, and so most mobiles won't be able to render the particles.

The VFX graph package is also a graphic coding way which greatly eased the visual effects development. The graphs are accessible and commented, so you can benefit from this easiness by customizing yourself the graphs and making them suit your project (personal use only).



The water interaction visual effect graph.

Many properties are exposed in the inspector of each particle system, yet there could be even more and some already-tweaked value lies in the graph. Don’t hesitate to take a look at these graphs (after reading this section to understand how they work) and modify the graph according to your project’s needs.

Realistic and scalable particles

All the water particles systems were developed considering physics phenomena. So their speed, size, rendering, and many other attributes evolve in the most realistic way. But most importantly, the particle's behavior scales well to any size they simulate. So without changing much the properties, you can simulate with a single VFX template both large scale objects as well as small ones.

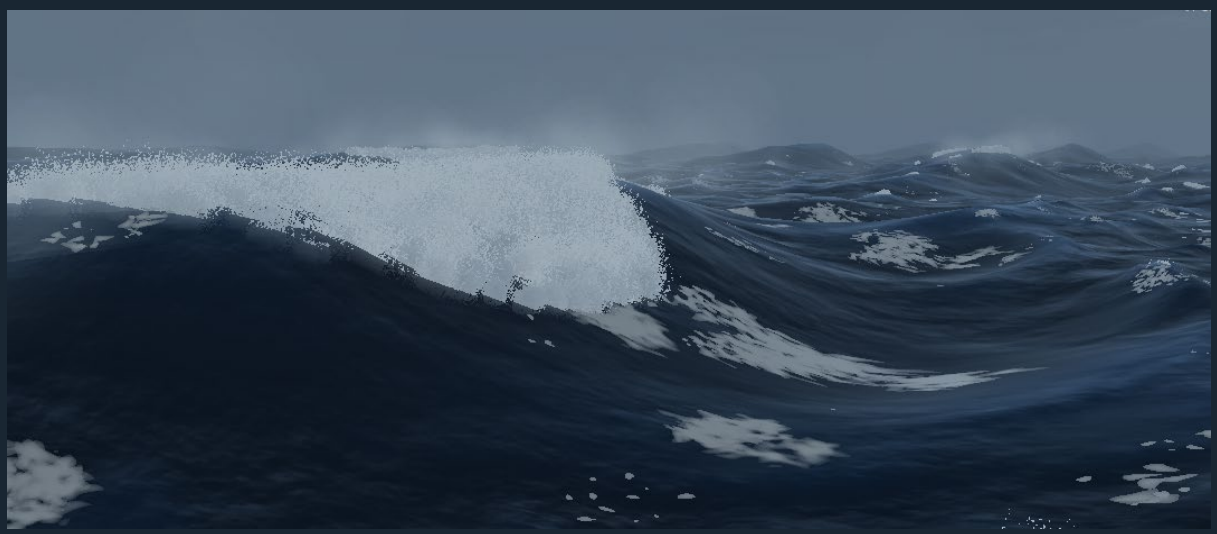
Type of particles

In Storm Breakers there are 3 major kinds of particles :

- Sea foam burst : They represent the sea foam projected in the air because of water hitting a surface. They happen when a wave breaks and when an object hits the water with sufficient velocity. The wind takes them away a bit. Physically they are drops of water foam of large size.
- Sea foam floats : They represent the sea foam floating on the water, generally a result of the airborne sea foam falling down the surface.
- Sea spray : They represent loose water being atomized by the wind blowing. They happen when the wind is strong enough and when the water is considered loose : at the crest of the waves or during a splash. The wind takes them away. Physically they are very thin drops of water

The boat controller VFX also includes the rendering of swirls that would be generated by the propeller spinning; this kind of particle is quite similar to the sea foam floats.

Ocean visual effects



In this screenshot we can see the sea foam bursts, the sea foam floats and the sea sprays.

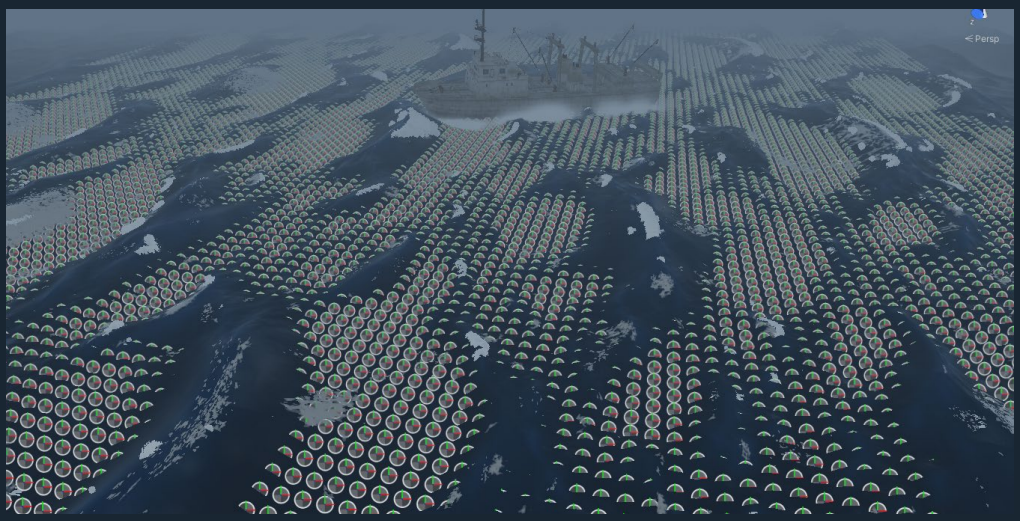
In Storm Breakers, the way the breakers are simulated is mainly through the visual effect, here is briefly described how the particle system works.

Note : this particle system is not synchronized with the ocean material in the editor. This will be fixed as soon as Unity provides a way to synchronize both. If it’s too annoying in the editor, you may deactivate them temporarily.

Primary and secondary grid

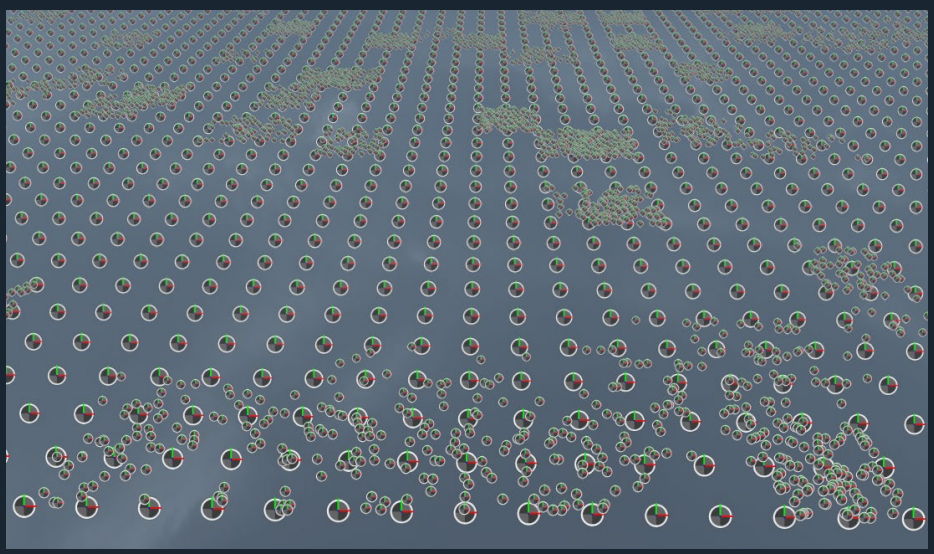
To generate many particles on the ocean surface, this particle system relies on a two level grid. On the so-called primary grid, invisible particles read the sea state at their position. When a primary trigger is reached, the invisible particles generate a number of other particles around them, these new particles in turn read the sea state at their position and become alive or dead function of a secondary trigger, the alive ones will then render either seam foam burst or sea spray. This way many particles can be precisely generated by keeping a good GPU process time. Though, sometimes the primary grid is too rough to detect small waves and there can be moments with a lack of particles on these waves.

In the primary grid, the invisible particles are regularly disposed inside the VFX bounds. They move with the ocean gameobject to stay in the camera focus. You can change the bounds of this grid to optimize the performance or conversely increase the width of the visual effects (in that case you might need to increase the system capacity). The number of particles in this grid is a function of the primary step as defined in the VFX inspector.



The primary grid particles visualized with textures.

The secondary grid is generated within a square shape that matches with the primary grid, the particles are randomly positioned within this space. The number of particles in this grid is a function of the primary step, the secondary step and also the size of the breaking wave causing the trigger (the wave size is traced back with the breaking speed that is a by-result of the trigger computation).

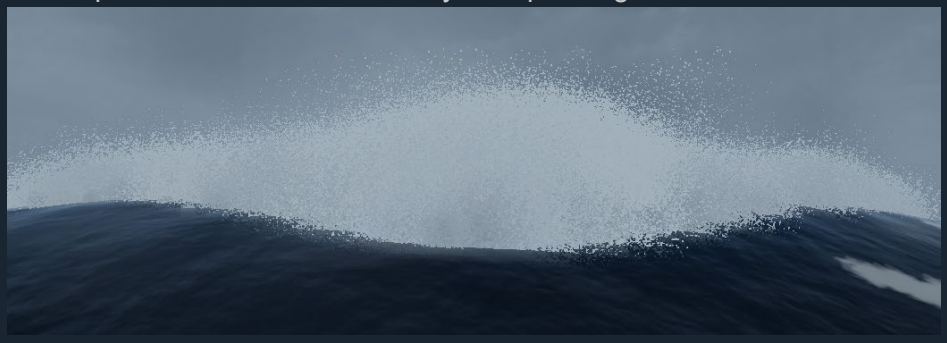


The particles of the secondary grid before the secondary trigger is read.

The setup of the grids works with many kinds of sea state. If it doesn't work well because you are making a special kind of sea state, then you may have to modify their properties but before a solid understanding on how the whole system works is required.

Sea foam bursts

The sea foam bursts represent the burst caused by the lip hitting the wave surface.



Compression trigger

The sea foam bursts are triggered by a physical value that is believed to be the real world trigger of the wave breaking : the compression (Some would says that the slope is the trigger: above 30° the wave breaks the same ways a sand pile would collapse when too steep, but further observation showed that a wave breaks regardless of the slope it lies on.) The compression is how much the water is horizontally squeezed, the derivative of the horizontal deformation in more mathematical terms. Note : in reality the horizontal compression is actually the cause of the waves rising up and down because of water's incompressibility.

The compression can be visualized with the mesh density seen from above.
2 vertices brought to the same point means a compression of 1.

To be realistic, the compression should be computed by adding the derivatives of each system's horizontal deformation. This would make the large waves break the smaller one on their crest like it does in reality. But issues arised doing such an addition, so currently there is no addition of each wave system's compression, they break independently of each other. This makes quite some wrong breaking, especially in the trough of a wave no other waves could break because of the negative compression lying here. Future updates might try to improve this, but only seasoned eyes should spot the wrong breakers.

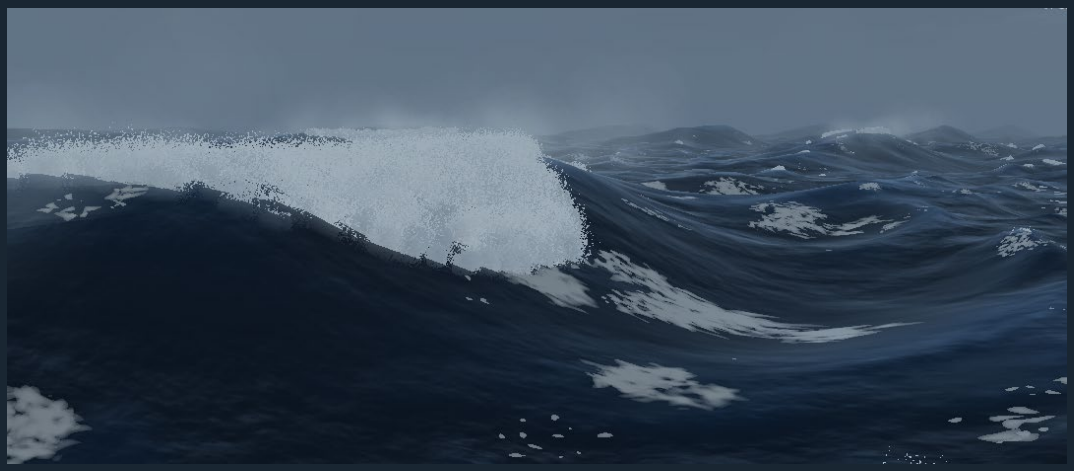
The default compression limit at which the sea foam burst appears - the secondary trigger - is 0.85. (An horizontal compression of 1 would lead in physics to an infinite wave height so this value is way higher than the one that happens in reality, this is because the horizontal deformation of the wave has been

increased to make a natural tessellation as explained in the Wave model overview in this document.) This limit happens as soon a wave reaches the same intensity. So this happens a bit before the wave height is clamped so as to simulate the ripples triggering the breaking and the wind blowing off the crest. (This effect might be improved in future updates so as to actually take account the wind strength.) Increasing this value makes the waves break less longer, which can be an effective way to reduce overdraw but making a less rough sea. In the primary grid a very low value is taken so as to avoid missing small breaking waves that could lie between the grid reads.

Hit point

When the secondary trigger is reached, a particle from the secondary grid is set alive and its position is changed. This new position is not exactly the one the waves would lead to, the hit point is taken in account.

The hit point is the position from the crest at which the virtual lip would hit the surface. It is computed somewhat to a parabolic trajectory versus a plane equation. The speed of this parabolic trajectory is linked to the size of the wave and the compression so as to be realistic. Thanks to that, large breaking waves make foam over a distance that simulates real breakers nicely.



When several waves are breaking at the same spot, the hit point positions are summed.

Speed

Making just a positioning with the hit point wouldn't be sufficient for the particles to stay along with the wave during its lifetime, a speed must be printed to it. In the graph you will find this speed as break speed (although it’s not exactly the speed at which the lip would be spilled) and it is computed as proportional to the wave speed with a bit of vertical component.

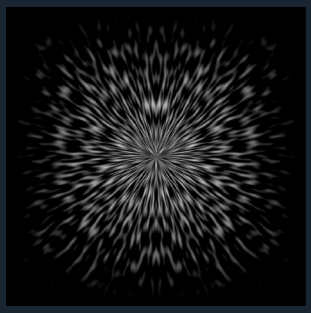
Like the hit point, the wave system's break speed adds to each other when several waves break at the same position. This adds naturally some realistic randomness in the burst.

Size

The size of the particles when spawned is linked to the density of the secondary grid so as to avoid lack of particles. Some randomness is added to break visible patterns.
Then the particles grow as a function of the wave size. When growing, the texture is stretched.

Rendering

The sea foam bursts are rendered with a texture that has been procedurally generated.



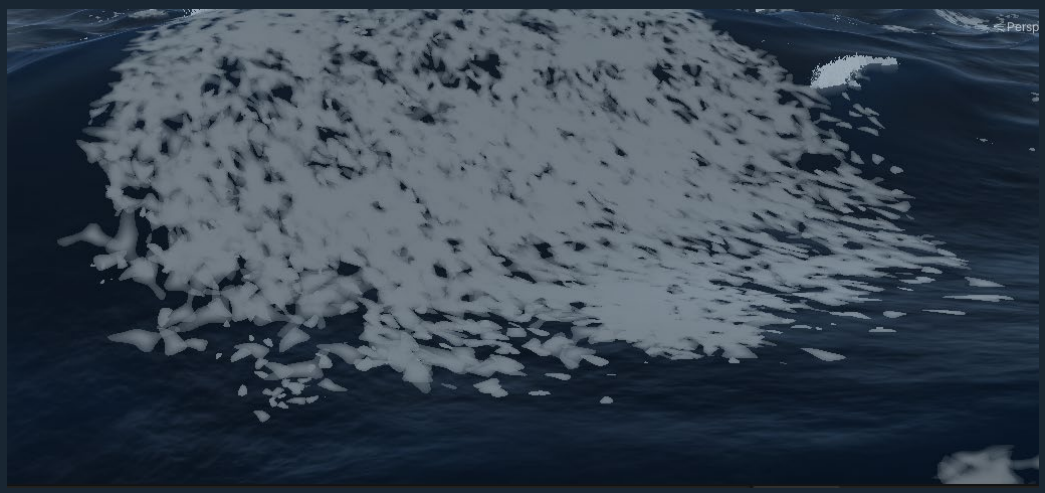
A darkening has been drawn in the middle of the texture to try to simulate the light occlusion happening in the middle of the burst. This effect is further simulated with a full texture darkening function of the lifetime of the particles. So when a new particle is spawned, it is assumed to be within the bulks of particles so it should be occluded from the light, and when growing it is assumed to get on top of the particles bulks so its color become the one from the lighting (sum of the halved directional light and ambient light as explained in the VFX rendering section this section).

Lifetime

The lifetime is computed as a function of the speed and vertical acceleration in an attempt to predict when the particle will go under the surface without computing the collision. When the particles die, it creates the sea foam floats.

Sea foam floats

The sea foam floats particles represent the sea foam that floats on the water in the turmoils of a breaking wave.



Lifetime

These particles are spawned by the sea foam bursts when they die. They are being given the undeformed position that has been computed roughly by the burst and a size proportional to their. Then the lifetime is a function of the size so as to make the floats of large waves lasting longer than the smaller ones.

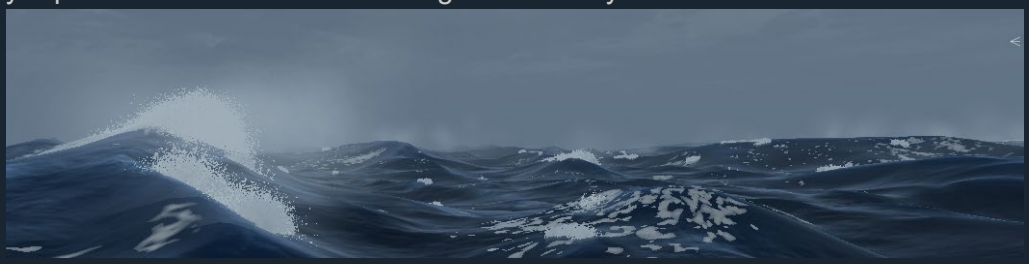
Rendering

The particles are randomly oriented. This plus the random size and random position coming from the burst, this makes a highly randomized pattern. The particles are rendered using the same vertex shader as the ocean and a procedurally generated texture with an alpha clip function of the lifetime to disappear by leaving a thinner float

It is to be noted that something important is missing in rendering the breaker's floats : there is no directioning. In reality a wave about to break should leave long strings of floats. Future updates will try to render this effect.

Sea spray

The sea spray represents the loose water being atomized by the wind.



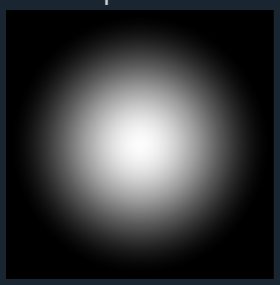
Lifetime

Sea sprays are spawned using the primary and secondary grid system. The trigger is a function of the local wind compared to a minimal wind trigger, and how sharp the wave is (computed somehow like the compression but also including the wave effective height). Because the secondary grid is too dense, a sort of probability act to reduce the number of particles generated.

The velocity of the particles is the local wind inclined to a 30° slope so as to simulate the wind acceleration at the crest of the waves. Then all the others properties are user defined (lifetime, size and growth rate, transparency)

Rendering

These particles are rendered using a simple transparent texture colored by the lighting.



Water interaction visual effects

Storm Breakers package includes a physically based particle system to render the splashes and floats of any object moving through the water.

The splashes, the sprays and the floats are generated by the Water interaction VFX.

Their generation is computed along with the physics using the same simulated mesh. They scale pretty well to any size of simulated object without changing their properties much.

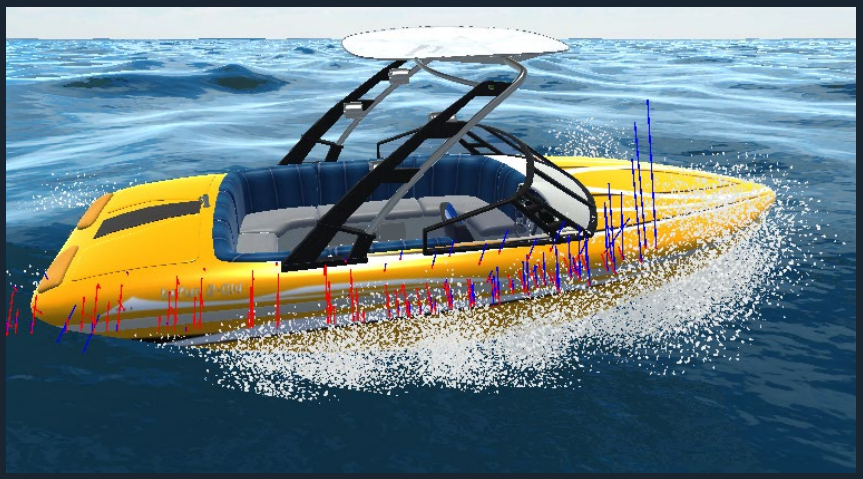
Sea foam bursts

The sea foam burst represents the water drops projected by the water hitting a solid surface.

Large bursts caused by high speed hits.

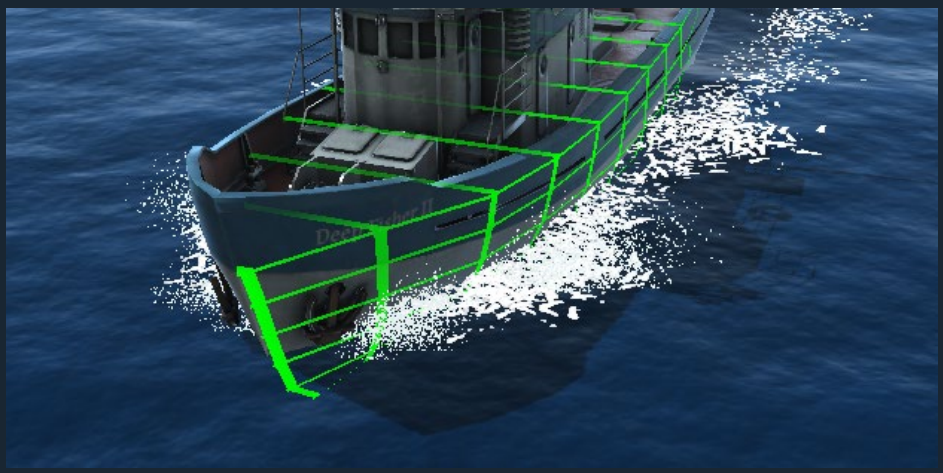
Spawn

The sea foam bursts are spawned per triangle of the simulated mesh when a velocity based trigger is reached. One component of this trigger is the normal relative speed (the speed of the triangle relative to the water projected in triangle normal) This speed can be visualized with the dynamic pressure in the gizmos mode (blue lines).



Dynamic pressure (blue lines) is linked to the burst generation.

This means the particle generation is bound to the shape of the simulated mesh. Notably a blunt bow will generate more burst than a sharp bow. This is because when moving straight forward in the water, the triangles normal of a blunt bow will be more aligned with the incoming water velocity. It is better to flatten the bow of the simulated hull a bit to emphasize the burst when moving in calm water than to increase its generation trigger, or you may face too much burst generation under special situations.



The blunt bow of this vessel makes enough particles burst at moderate speed because triangles face more the incoming water stream.

The burst particle generation is also linked to the plunging speed, that is the vertical component of the relative velocity. This helps render the wake wave that is generated when a body is immersed fast in the water. The sum of the normal relative speed and the plunging speed, clamped to positive value, is the trigger and is called the impact speed and defines how strong the collision is.

The number of particles generated is currently constant per fixed delta time frame as soon the impact speed exceeds its trigger value. So if you change the fixed delta time you may have to change the number of particles created by frame as well.

Lifetime

When spawned, the particles are positioned at the triangle center and then their position is scattered around it. They are also outwarded and lowered using the triangle normal and size so they appear in the water and as far as possible to the simulated mesh to avoid seeing them through the hull.

Their final velocity is the addition of many velocity :

- The bulldozing speed is the *triangle velocity* in the triangle normal direction, this simulates the hull pushing the water.

- The tangent flow speed is the *water velocity* projected in the triangle plane, this simulates the water not being slowed down in the tangential direction.
- The outward speed is the *impact speed* oriented in the triangle normal and factored by a property. This simulates the water bouncing off the hull.
- The upward projection is the *impact* speed reoriented and randomized in the vertical axis and a bit in the tangent axis. This simulates the water spilling preferably upward due to the excessive compression of the water underneath.

The outward and upward velocities are very important to tweak : increasing the outward speed against the upward speed reduces the chances of having the particles going through the hull.

Then the velocity is affected by gravity and a bit of drag with the wind. To try simulating the wind moving around the hull and not through it, the wind is projected on the triangle surface (whose normal is set horizontally to avoid the wind moving up and down).

The particles grow as a function of the impact speed to simulate the burst. When growing the alpha clip is modified to make thinner water drops.

All along its lifetime, the particles detect the water surface to see if they hit it. When this happens (and always happens with the gravity on, it's kind of a "while true" loop but the lifetime makes sure they eventually disappear), then the particles die and generate floating particles precisely at the hit location.

Rendering

During development, the rendering choices for the sea foam burst of the water interaction were based on the breaker's sea foam burst, meaning using stretching textures with many drops drawn on it. While this was necessary for the breakers visual effect to avoid having countless numbers of particles to compute, this choice turned out to be non optimal for the water interaction : excessive overdraw with highly clipped texture, and a hard way to make them collide and clip with geometry, thus making particles seen sometimes through decks. The solution for these problems would be to split the particles to smaller ones with a different texture representing much less drops, thus rendering way more particles but only were required. First tests using this approach were conclusive enough performance wise, still much development is required to make a nice rendering. So a major update is to be expected with the water interaction sea foam burst : more smaller particles instead of large stretching ones.

Another issue that could be solved with this new approach is that the texture doesn't scale well with the size of the object interaction on the water. For a large ship hitting large waves, we would expect some completely atomized water that could be mistaken for sea spray. To offset this issue you can draw your own texture and replace it in the VFX inspector (though it is quite difficult to break the regular shape of the particles). The provided texture, like most of the textures provided, was drawn procedurally using a shader graph and a third party tool to bake the texture in a file.

During its lifetime, the alpha clip of the texture varies function of the growth to simulate the drops losing size in the air. You can control this effect using the very sensitive “Burst Pixel Loss Rate” property .

Note how the drops shrink in the air.

Sea foam floats

These particles represent the foam floating on the water after the bursts fall down on the surface. They make a natural and dynamic wake.

The wake is naturally done with the burst falling down the surface.

Spawn

The sea foam floats particles are spawned by the sea foam burst falling down the water surface. The position is precisely the one of the burst even in the waves, this is thanks to the water detection algorithm explained in the physics overview of this document. Yet because there is quite a mismatch between the burst and floats textures (this is to improve the visual quality of the floats), the precision of the burst falling down the water and making floats there is lessened. This might be improved with the planned update of making smaller burst particles though, it would reduce any chance of having particles spawning and disappearing abruptly.

The alpha clip used to shrink the particle with time is inherited from the burst alpha clip so as to try to match the foam density.

Lifetime

After being spawned, the particles can move outward of the hull remembering the normal of the triangles that generated the burst. This helps greatly to simulate the wake wave and make a V shaped wake.

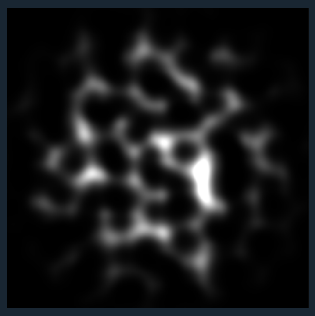
The alpha clip is increasing with the age of the particles so they shrink and stay opaque instead of fading out with a basic transparency.

A wake made last longer, note the shrink of the floats with time.

Beware when increasing the floats lifetime too much, you may exceed the particle system capacity and no more particles will be spawned until some die. Though you can easily change this in the VFX graph, the current capacity has been optimized.

Rendering

The floats are floating using the same vertex shader as the ocean. This shader is opaque lit with alpha clip but doesn't currently support a normal mapping. The texture provided is procedurally generated and can be changed to another of your choice to match your project style.



Because the floats can have a large size, the default 5 vertex quad they use to be rendered is frequently insufficient to accurately follow the ocean waves, leading to some artifacts. To solve this you can change the mesh in the VFX inspector by using one of the subdivided quad provided in the *6-Models* folder of the Storm Breakers package. Future updates will permanently solve this issue by using submeshes function of the particle size.

Sea sprays

Sea sprays particles represent the water being completely atomized by the wind blowing.

Some sea spray near the surface.

Spawn

Sea sprays particles are spawned pretty much the same way than the burst, but also taking in account the local wind. This makes them appear at the same time as the burst when the wind is blowing hard enough. Their number also depends on the product of the local wind per the impact speed as an attempt to increase the spray density when large hits happen under high wind, though this effect still fails to render correctly real sea sprays and will be more developed in the future.

Lifetime

Like the sea foam burst, the sprays are affected by the wind and gravity but with a larger drag coefficient. The wind is deviated around the hull the same way as the burst particles do so as to prevent the particles going through it. The wind is randomized a bit to scatter the spray particles. It would have made a better rendering to have the particles closely moving above the hull using a fluid simulation of the air, but this is not possible without sacrificing the frame rate that can already fall down in stormy scenes. So currently the sea spray stays close to the surface and we don't see them much.

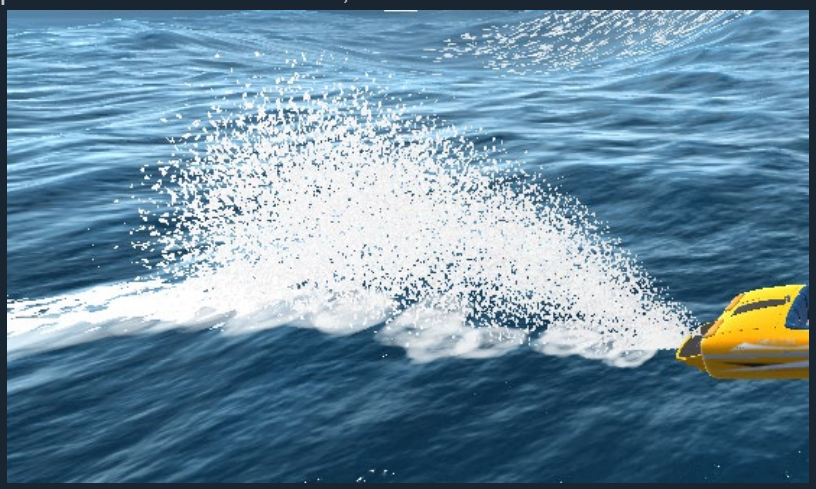
The lifetime, size, growth and transparency are mostly property driven.

Rendering

The sea sprays are rendered using a fake lit texture with the alpha looping from 0 to 1 during lifetime. They can be the cause of overdraw. They use the soft particle shader of the VFX graph to simulate their volume when close to an opaque surface. Though this effect doesn't work on the water because it is drawn in the transparent queue and its depth cannot be accessed because of this.

Boat controller visual effects

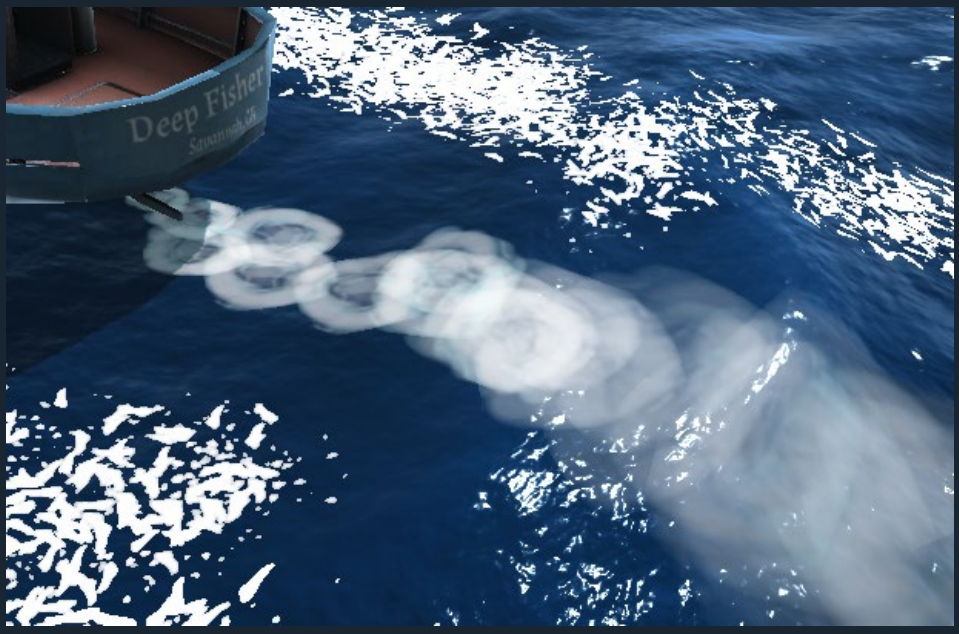
The Boat Controller component included in Storm Breakers works along with its own visual effect that generates 3 kinds of particles : sea foam bursts, sea foam floats and the swirls.



The boat controller can generate 3 types of particles : sea foam burst and floats, swirls.

Swirls

Swirl particles represent the turmoils left by the propeller in the water.



Spawn

The swirls particles are spawned as a function of the propeller usage at the undeformed position of the controller gameobject by the controller component. This component tries to spawn them at a fixed distance rate when moving so as to have a constant density of this effect no matter the speed (but at high speed the frame rate might not be enough to maintain the fixed distance rate). When the controller is not moving it's a time rate function of the lifetime that spawns the particles. Then in the visual effect the position is randomized to scatter the particles and remove visible patterns.

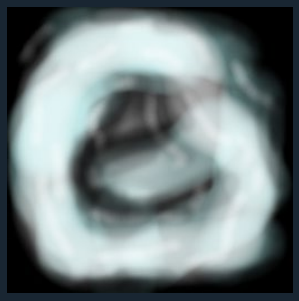
The transparency is a function of the propeller usage too so as to simulate more intense swirls when the propeller spins.

Lifetime

Nothing much happens during the lifetime of the swirling particles : their size grows to simulate the swirl spreading, and the transparency diminishes to make the particles fading out before disappearing.

Rendering

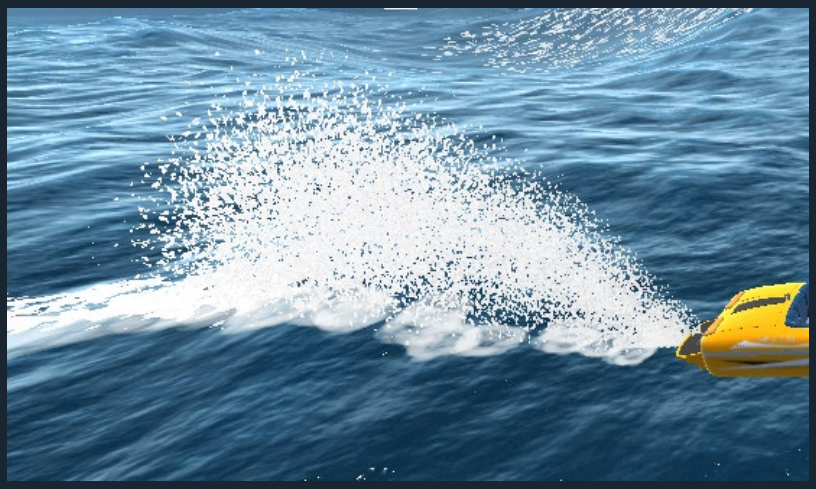
The swirls particles are rendered using the same vertex shader as the ocean. A hand drawn texture is provided and you can replace it with another texture that better fits your project.



The default mesh used to render the swirls particle is a 5 vertex quad. When their size exceeds the ocean waves, you may pick another mesh with more poly from the model folder of the Storm Breaker package so they better float with the waves.

Sea foam bursts

Sea foam burst particles represent the burst of water projected upward caused by the propeller when near the surface.



Large sea foam bursts emphasize the power of the boat.

The sea foam bursts are spawned at a local position you can set up in the Boat Controller component. The speed they are being given is a function of the propeller usage and also to the depth to simulate the propeller spinning in loose water.

The sea foam burst's lifetime and rendering is similar to the ones of the water interaction. They also leave a sea foam float when falling down the water surface.

Sea foam floats

Sea foam floats particles represent the foam floating on the surface when the bursts fall down on the surface. They are similar to the ones of the water interaction.

Audio overview

Storm Breakers package includes all the water and wind audio effects as well as a simple engine sound modulation.

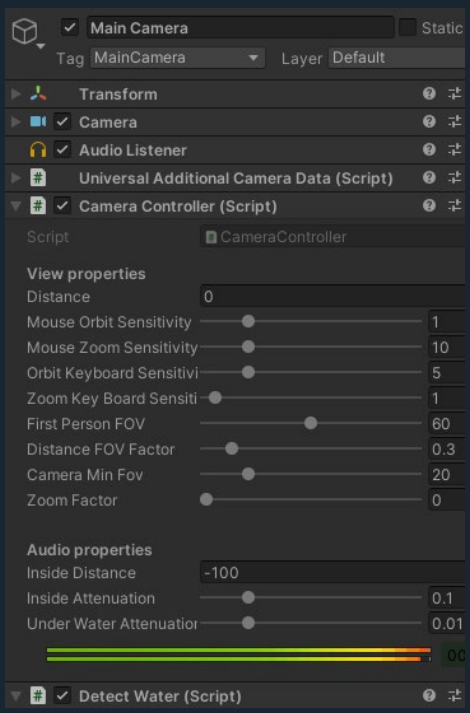
Sound power and saturation

When a large wave breaks or when a large ship hits the water fast, we expect loud and powerful noise, but this is not exactly possible without turning the speakers very high and lowering the overall volume. Doing so would make ear injury risks with other audio and so this approach has not been chosen. Instead the loudness of some audios were trimmed down by setting a higher pitch than expected. Here is why.

Both internally in the audio stream and in the speaker construction, the values generating the noise are bounded. Indeed the audio stream is a series of floats values ranging from -1 to 1, they roughly define the position of the membrane of the speaker as function of the time, and so the membrane vibrates according to the stream of data and produces audio waves in the air. This makes the low frequency need to be lowered to keep within those bounds and so they cannot be powerful enough.

If we try to exceed the bounds, the values of the audio stream are clamped to [-1,1], this is called saturation and it doesn't make more sound. The human ear is quite sensible to saturation, it makes a special kind of noise that is disturbing. So it's not good to make the audio stream saturates too much.

Still, it has been observed that a bit of saturation makes a noise looking more powerful by adding some more high frequencies. You can visualize the audio saturation in the inspector window of the Storm Breakers Camera Controller and you should try to keep it moderate :



The red indicates the audio saturating, the 2 bar are for each audio channels (left and right)

That being said, it is recommended to turn the speaker high to fully benefit from the audio quality of Storm Breakers : loud breaking wave's roar, shivering wind and satisfying splashes.

Waves audio effect

Not only the Storm Breaker's waves break with an exclusive visual effect, they also make realistic noises. This effect is done with the Breakers Audio component.

A single audio source

All the procedural audio is encapsulated in one script that feeds one audio stream. This is because in a stormy ocean, there is likely more wave breaking at the same time than there are audio channels in your hardware, which would cause audio preemption when using many gameobjects.

How breaking waves are detected

The breaking waves are not detected using the particle system (running on the GPU it's hard to get events from it). The algorithm is actually quite different to the particle system's grid approach that would lead to too much CPU computation time. It works thanks to the waves being coded as groups.

Virtual emitters are coded in C#, they virtually move to keep in the middle of a group. When they reach too far from the camera, they switch to another group within the bounds (this is actually coded in a simpler way using the rhombus grid coordinates). Every frame, these virtual emitters check if the phase in the center of their group and the current group intensity make enough compression for the wave to break. When this happens, they compute their audio volume based on the wave size and the camera distance and orientation, they also compute the pan and pitch.

This particular way of detecting the breakers and its difference with the particle system approach can lead to some difference, especially for large waves because they break longer. Yet it allows to keep the forward way of coding that is the backbone of Storm Breakers : no callback delay, only fast forward computation.

The fact that currently in the VFX the waves break independently from each other helps a lot with the synchronization between the particles and the audio. If the breaking VFX is updated to use the real compression addition like explained in the VFX overview section of this document, then this algorithm for generating the breaker's audio will need to be reworked and it might be more CPU expansive.

How the audio is generated

The audio is generated in a OnAudioFilterRead method. It requires an empty audio source to play so it can write custom data on the audio stream.

There are no audio clips for this effect, the noise is generated by modulating and filtering a white noise produced by a series of random numbers. This reduce a bit the possibility of increasing the low frequency and reducing the high ones, but it is necessary to prevent the different virtual stream to interfere to with each other (with the same audio clip for several breakers playing simultaneously, the addition of each virtual stream value make a permanent saturation on the final audio stream).

So for each virtual emitter, the virtual stream consists of a white noise that receives first a low pass filter to make the noise a bit slow pitched, and another low pass filter to simulate the audio fading off with the distance. Then the volume and pan that have been computed by the virtual emitters in Update are applied to this filtered noise. Finally all the virtual streams are added on the audio stream that is sent to the audio listener.

It is to be noted that the spatialization of the noise is hand coded : the volume fade off with the distance using a low pass filter (https://en.wikipedia.org/wiki/Low-pass_filter) instead of a basic volume variation, this makes the high frequency disappear faster with the distance that the low frequency. Thanks to that, if

you turn your speaker’s volume very high in a stormy scene you’ll be able to hear the low roar of distant breakers. Yet the spatialization might lack a bit of precision if you try to focus on it, future updates will try to improve it.

Splash audio effect

Storm Breakers package includes a procedural audio effect of the object splashes in the water. The audio is computed at the same time as the particles and using the same physical value for a better fit between visuals and audio effects.

To generate the splashes audio, a 3D audio source playing permanently along with either the Water Interaction component or the Sphere Water Interaction component should be attached. Storm Breakers provides a suitable audio clip for this effect (splash.wav) that has been generated by filtering a pink noise.

The audio is generated by modulating the pitch and volume through a hand-coded filter. Both depend on the total dynamic pressure acting on the mesh : the more dynamic pressure, the louder and lower the audio will be. Then the audio pitch and volume fade off at independent rates to simulate the drops falling down the water.

This effect can become oversaturated for large ships, and lowering the volume only makes the effect less audible. As explained in the beginning of the section, this is because of the bounds of the audio stream. To solve this you may either lower the volume of this effect (so it's less audible) or increase the pitch to make more powerful noise (but too high pitched for the size of the effect).

Wind audio effect

Storm Breakers package provides an audio effect for the wind.

To use it you should attach a 2D audio source playing permanently along with the Wind Controller component. Storm Breakers provides a suitable audio clip for this effect (wind.wav) that has been generated by filtering a Brownian noise.

The Wind Controller component modulates the audio source’s volume and pitch function of the wind strength and camera position and orientation. When facing the wind the audio is louder to simulate the wind blowing harder in both ears. Currently there is only one audio clip modulated, this audio clip is rather suited for the sound of the wind around, but not directly the noise that would be generated in the ears, so the effect can be a bit unrealistic. Future updates might segregate both sounds to improve the simulation.

It is possible to include the camera velocity when computing the wind, this makes better high speed feelings when reaching so. But as explained above the effect might be a bit wrong because it does not accurately simulate the noise happening in the virtual ears.

Turbulences are included in the generation of the audio to make it look more realistic. In future updates the turbulences will be generalized and synchronized to the physics and water surface rendering so as to have an improved simulation of the wind.

Engine audio effect

The Boat Controller component includes the possibility to modulate an audio source’s volume and pitch to simulate in a simple way engine sounds.

To use it you should attach a 3D audio source playing permanently along with the Boat Controller component. Storm Breakers does not provide any engine audio clip.

The Boat Controller manages the audio volume and pitch function of the propeller usage to simulate the engine noise function of its load. The pitch is also managed with the propeller depth to simulate the engine revving higher when the load comes off.

Future updates will try to implement the forces and audio varying with the speed to improve the engine load simulation.

Stress cracking

It is planned in future updates to include a fully procedural cracking noise in the water interaction component to simulate the structure stress audio. This noise will make cracking with the volume function of the bending stress and pitch function of the time variation of the bending stress.

Gameplay overview

Storm Breakers is not only meant to be a realistic simulation, it has been coded with gameplay in mind. In this section are explained some gameplay considerations.

Simulation vs arcade

Storm Breakers is coded in between a realistic simulation and an arcade game. Science has been used massively to produce convincingly enough visuals and to make the engine scale well to any kind of object sizes as long as they represent real world objects. Yet many simplifications were done in modeling physical phenomena, reasons for this are various : to maintain high performance in game and make this engine suitable for many hardware, and to simplify the developer interface by lowering the number of properties to tweak.

So Storm Breakers might be a bit imprecise if you want to make an accurate naval simulation. You can still add your own scripts to improve it.

Possible game mechanics

Gameplay has always been kept in mind while developing, the idea of Storm Breakers is to provide realistic game mechanics based on real world phenomena. The demos and examples provided show off achievable game mechanics. Here are some explanations of what game mechanics you can rely on with Storm Breakers (apart from the classic gameplays of naval games).

Dangerous waves gameplay

Thanks to the way the waves are coded with a customizable predictability and an adjustable break force, they can be intrinsically part of an epic gameplay.

Capsizing under a large breaker.

With a correctly adjusted dynamic, a waterfact can capsize under the slope of the waves. The breaking force can further increase the dangerousness of the waves. So a possible gameplay would be to navigate in a stormy water, to predict the large waves by spotting the groups (having learned their behavior), figuring out where the breakers will hit and try to steer accordingly to avoid them. Some more game mechanics could be added on top of that : water entries that make you need to activate the pump, reducing the power on the propeller and thus your ability to steer in the waves, managing the boat and its crew during the time between large waves, etc.

More casually, steering a boat that doesn't keep its course under the waves is a simple yet effective game mechanic to get hooked on. In that case you don’t need much to be able to read the waves and predict them, but simply learn how to anticipate the boat's changes of bearing with the waves. At high speed you may have to better anticipate the waves to avoid bouncing off them.

Surfable waves gameplay

With its advanced physics and customizable wave, the Storm Breakers package provides the ability to surf !

Starting to ride a 10m high breaking wave.

This game mechanic is implemented in the surfing demo with a sea state set voluntarily unrealistic to have highly predictable and easy to ride waves. With the groups’ amplitude varying over time, you need to constantly search for new groups to ride on to, leading to some high speed navigation between them that is also interesting for gameplay. With the waves hard to catch because of their speed, and with the break force of the waves that can throw hardly a small speedboat in the air, surfing can become tricky and making it even more satisfying to succeed !

Satisfying effects

With the whole particle system stack synchronized with procedural audio, Storm Breakers can render satisfying water experiences.

Sailing can be satisfying to watch.

Like in reality with the popular ship-in-storm footages, many water visual effects can be satisfying to watch. This comes from the fluidity and size of the visuals, and also how a watercraft reverts effortlessly to a clean state after receiving large blasts.

Also the way the audio is implemented using white noise, pink noise and brownian noise is quite similar to the one used for some ASMR noises, providing satisfying audio experiences for the player.

Seasickness

When making a game mechanic based on the waves, seasickness management is very important. Here are several things described so you can manage it well in your game.

The seasickness we can feel while playing a boat game is a bit different from the one we would have in the same condition in the real world. In game, what makes us feel sick is mainly the oscillating movement spread on the screen plus the immersion feeling. In real condition, the slow movement under 1Hz also makes us feel sick through the inner ear, but such frequencies don’t have much impact on game since the inner ear doesn't move while seated. This difference is to be noted, in real world we would aim for a vehicle moving at frequencies near 1.2Hz to reduce seasickness, while in game we want frequencies to be as lower as possible.

So there are two things that foster seasickness : oscillating movements and how much these movements are present on the screen.

If you want to reduce seasickness, you may have to tweak the physical behavior of the watercraft the camera is attached to for a lower wave sensitivity, as explained in the physics overview section of this document (higher gravity center, larger inertia, more density…).

You may also have to consider the camera setup. Storm Breakers provide an orbital camera setup that doesn't follow the watercraft rotation, this to reduce seasickness when seeing from a distant third person point of view. However, when seeing from a first person point of view, the waterfact geometry takes more space on the screen and can lead to seasickness because of it oscillating too much. Then tweaking the dynamic behavior of the watercraft is necessary.

On the other hand, if you want to make some hardcore gaming where seasickness is part of the gameplay, then you can make the camera follow to some extent the watercraft rotation, a tweak allowing for more or less of this effect function of the player. (This was tried to be implemented in Storm Breakers but the attempts failed because of the quaternion interpolation limitations.)

Unity editor interface overview

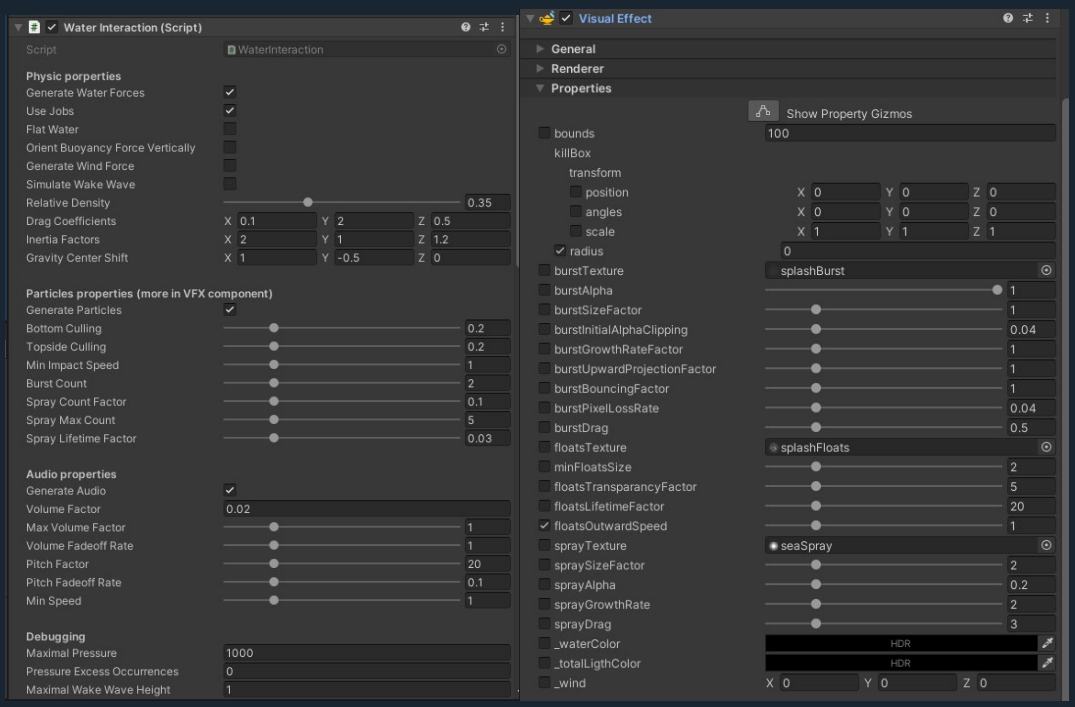
A few words about the Storm Breaker's interface in the Unity editor.

Custom editor

Currently there are no custom editors at all, but this will be quickly changed in future updates to improve user experience based on your feedback.

Sliders

Most of the components use almost exclusively sliders for the floats properties. The reason for this is to improve the user experience by providing a better mouse variation than it is with the classical float fields. For aesthetic and functional purposes, most of the sliders range from 0 to 5 times the default value (it's also a way to remember the default value). Doing this is possible thanks to the fact that in Storm Breakers, most value in the components scales well with the size of the object you are simulating and you rarely need to change their magnitude.



Sliders in the Water Interaction component and in its VFX

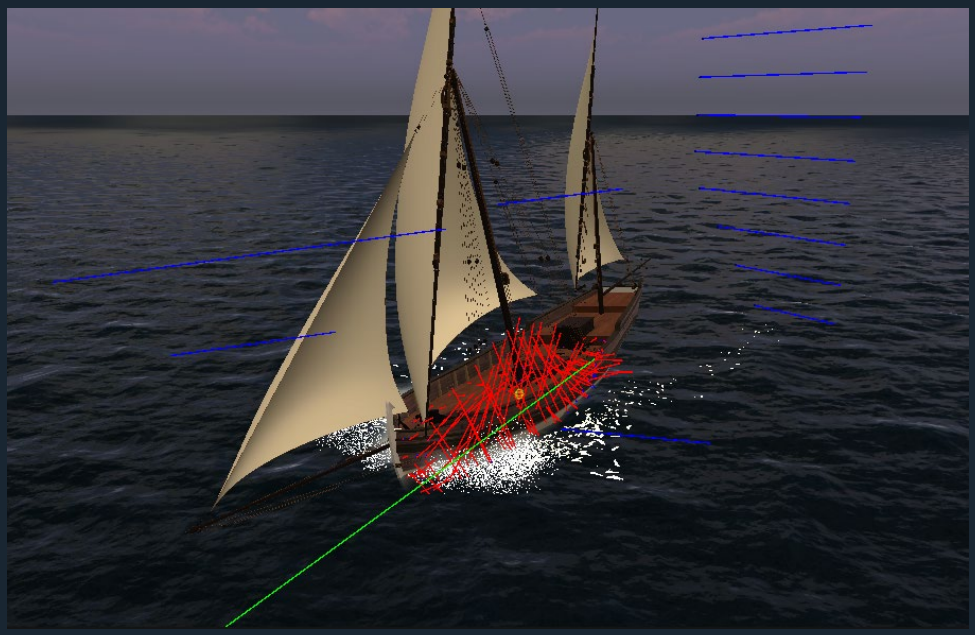
The counter side of this is that in rare situations it gives too small a range and you might want to exceed the limits. Currently the only way to solve this is raising such an issue to the developper so it can be fixed in the next update, and modifying the files that cause the trouble.

Depending on the user feedback, the sliders will be improved or simply removed in the future updates.

Units

Storm Breaker's engine uses the metric system. It's important that your 3D model scales to the real world object with the size in meters, otherwise it will move too slowly or too fast.

Gizmos mode



A screenshot with many gizmos drawn : hull forces, sails forces, automatic pilot course, wind velocities, gravity center.

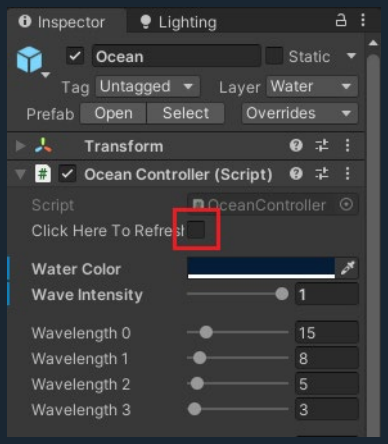
Storm Breakers rely a lot on the gizmo mode to show many helpers like some special positions or vectors. Turn it off when you want to have a clear view. All the component's gizmos can be independently turned off or on to show only specific helpers. Do this either from the inspector for the runtime force lines (they are not drawn in OnGizmos but on update using Debug.DrawLine), or from the Gizmos list of visuals.

As a general rule blue lines are for velocity related vectors (dynamic pressure, velocities), red line for static forces (static pressure, boat controller), green for trajectory (automatic pilot course), yellow for mass related items (inertia center).

In-editor script runs

To update the scene when modifying global sim data like the waves, Storm Breakers rely a lot on the OnValidate message to run some scripts. The most important one is with the Ocean Controller. Whenever you change something on its inspector it will run the code that it executes when launching the game : updating the waves, the rendering properties, the VFX, the lighting… There are some expensive codes inside (like searching gameobjects) so it can slow down the editor a bit while making the modification, but only then.

Sometimes this code should be called to update the scene but is not. This can happen after a scene save or an asset refresh, what goes wrong most of the time is the waves on the ocean (This comes from a serialization issue with the wave matrices of ocean.mat). To trigger the OnValidate method without actually changing something on the inspector, you can click on Click Here To Refresh tick box.



To help prevent this issue while a correct fix is found, there is one piece of code running permanently while in the editor (and only one). This piece of code permanently sets the matrices of ocean material with the one from the static class. It's not very clean, but without this the water would actually disappear frequently from the screen !

All the scripts running in the Unity editor are concealed within regions that are not compiled on build, so they don't have any impact on the builds.

Files protection

It has been chosen to publish the Storm Breakers package without physical protection on its files (They are legally protected by the Unity EULA though). The reason is to provide the users the ability to fully understand and customize the package for personal uses. Then with all its VFX and shaders done with easy-reading graphs, fully commented scripts and this documentation, you can adapt the whole package to your needs : changing the art style, modifying the waves model, using other physics engines…

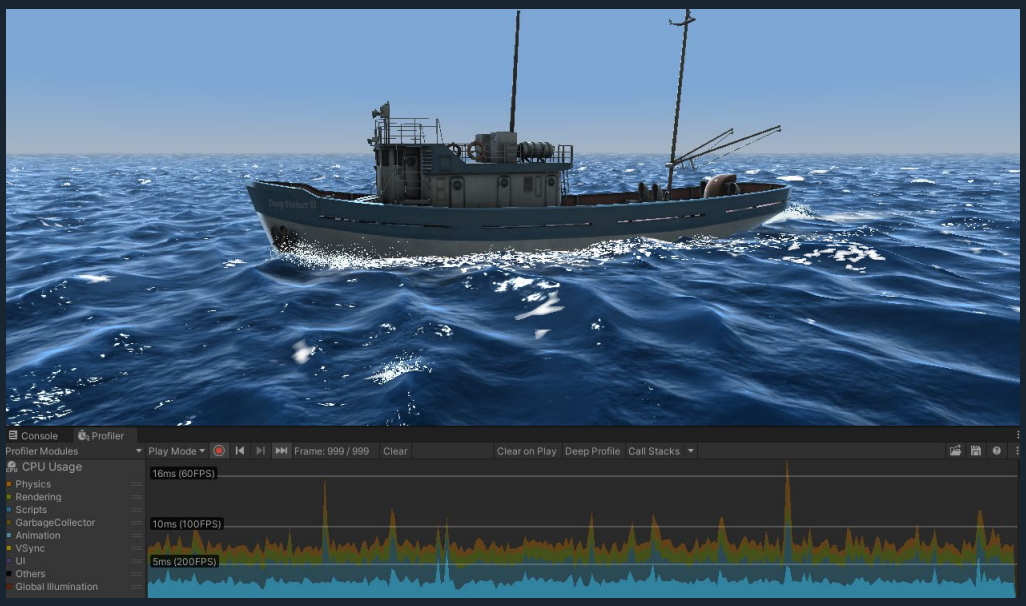
```
// oriented losange grid
Vector3 worldAxisPosition = undeformedPosition - directionVector[w]*groupPosition;
float Ai = Vector3.Dot(worldAxisPosition, iVector[w]);
float Aj = Vector3.Dot(worldAxisPosition, jVector[w]);

// calculating the coordiante of the center of the losange M
float im = Mathf.Floor(Ai) + 0.5f;
float jm = Mathf.Floor(Aj) + 0.5f;

// calculating the randomized variation of the center of the group C relative to the center of the losange
// doing in several steps to save values for others computation
float randomimjm = Mathf.Cos(33f*im+53f*jm)*Mathf.Cos(20f*im+48f*jm);
float deltai = randomization[w]*randomimjm;
float randomjmim = Mathf.Cos(33f*jm+53f*im)*Mathf.Cos(20f*jm+48f*im);
float deltaj = randomization[w]*randomjmim; ;

// calculating the minimal distance of the center of the group C to the edge of the losange
// non optimized : float distance = Mathf.Min(0.5f-Mathf.Abs(deltai), 0.5f-Mathf.Abs(deltaj));
float distance = Mathf.Min( 0.5f-(deltai<0f ? -deltai : deltai), 0.5f-(deltaj<0f ? -deltaj : deltaj) );

// coordinate in randomized losange
float ic = (im - Ai + deltai)/distance;
float jc = (jm - Aj+ deltaj)/distance;

// calculating the amplitude in i and j direction by making sure the value is clamped between 0 and 1
// non optimized : float relativeDistancei = Mathf.Min(1f, Mathf.Abs(ic));
float relativeDistancei = ic<0f ? -ic : ic; if(relativeDistancei > 1f) { relativeDistancei = 1f; }
float ai = 1f - relativeDistancei*relativeDistancei;
float relativeDistancej = jc<0f ? -jc : jc; if (relativeDistancej > 1f) { relativeDistancej = 1f; }
float aj = 1f - relativeDistancej*relativeDistancej;

// define wether is in front of the group or not
isFrontOfTheGroup = (ic-jc) > 0f;

// calculating the group amplitude variation over time
float variation = 0.8f + 0.2f*Mathf.Cos(10f*randomjmim + 0.1f*pulsation[w]*time);
```

A part of the wave model.

All of this is possible for personal use only. It's forbidden to sell any code containing pieces of code from Storm Breakers unless otherwise mentioned as an example. Selling a compiled game using the Storm Breakers package is allowed though. Check the EULA terms for more info on what you can do or not with Storm Breakers files.

Performance overview

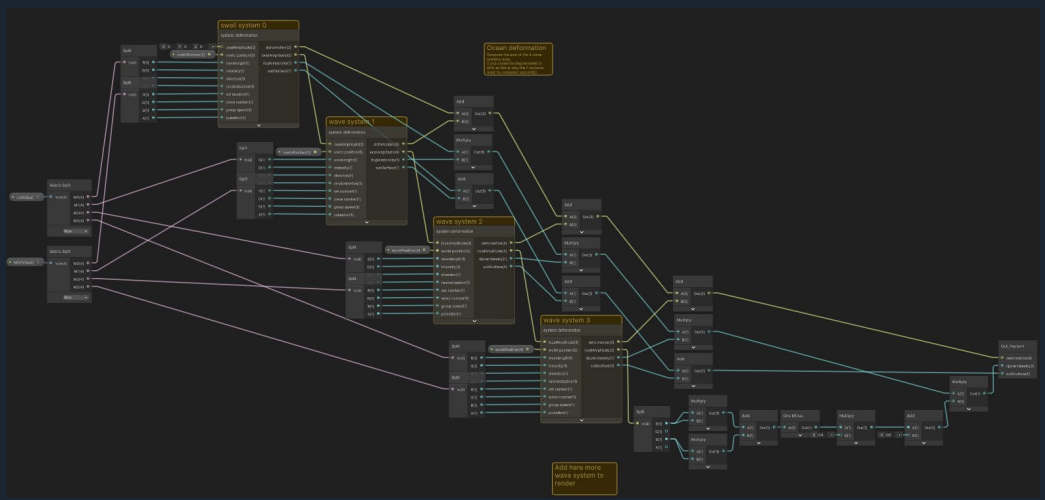
The Storm Breakers package has been developed by trying to keep good performances. It was tested on an average laptop to ensure its accessibility for most developers and players. It is optimized for what it is designed for : one or few watercrafts seen from an external camera. Still many optimisations can be implemented for more versatility, some of them are explained in this section.



Profiling on an average laptop.

High definition water rendering

As said the Storm Breakers package is optimized and tested to run on average hardware. With a higher end hardware you might want a better definition of the water rendering. This is possible without the need of more advanced shading, what misses mainly is a more detailed mesh with more waves systems computed than the 4 implemented. With some skills you can already do that on your side by changing the mesh used to render the ocean and modifying both the shader graph and visual effect graph “ocean deformation”. There you can add yourself more wave systems, the problem then is how to define their properties.



You can modify the shaders to increase the number of wave systems rendered.

Future updates of Storm Breakers might include a more detailed rendering of the water, it is likely that this feature will be sold in a separate package designed for high definition (including perhaps the Unity high definition render pipeline.)

Large scale simulation with numerous floating objects

Currently the Storm Breakers engine is not optimized to simulate a large amount of floating objects. Still, you can increase the number of simulated objects by trimming down the number of triangles on each simulated mesh, and furthermore by disabling the wave computation if the water is flat enough as the following screenshot demonstrates.

Seven ships of the line in a line of battle. Disabling first the wave computation half the CPU overhead, then decreasing the number of triangles reduces the force computation overhead to a reasonable framerate on an average laptop.

To increase the simulation scale capacity, futures updates might includes these features :

- Less precise water detection (approx -20% in wave computation)
- Possibility not to compute water speed (approx -20% in wave computation)
- Compute a limited number of wave system function of triangle size and distance to camera (approx -25% per system removed)
- Full Unity DOTS implementation with ECS + jobs + burst compile (unknown gain)
- Force computation brought to the jobs system (unknown gain)

Character point of view camera

Surfing POV

If your game involves a first person point of view of a character in a ship, then the rendering is already quite optimized as less water and particles are drawn to the screen. The trouble then is how to clip the particles that would go through the objects. There are some solutions as explained in the particle clipping section of this document, but you might have to wait for the possible major update in the water interaction particles for having smaller particles easier to cull separately. Also a close point of view on large sea foam bursts lead to more GPU overhead.

Overdraw

Overdrawing is known to be a major bottleneck, especially in scenes with a lot of visuals. What happens is that the GPU spends too much time drawing many objects on the same pixels. With the current sea foam burst particles implementation of the water interaction, the fully transparent areas of the texture are drawn and require a bit of GPU overhead in the fragment stage, which can add up when many particles are overlapping.

Except for the possible major update in the burst particles implementation of the water interaction, there are no updates planned to fix overdrawing. This is what to be expected in a scene with many visual effects, only good hardwares can render such detailed scenes.

Example scenes

The Storm Breakers package includes example scenes. These are the ones used to compile the demo of Storm Breakers but they don't include the asset for which there is no suitable licensing, that is the 3D models, the engine audio clips and the textured skyboxes. In this section you can find the links where to download them, most of these assets are for free.

These examples show you how to make various kinds of ocean scenes, take a look at them to learn quickly how to use the Storm Breakers package. You can also use them as a starting point to build a game.

Surviving a storm

This example scene aims to show off the stormy sea rendering capacity of Storm Breakers. The scene contains a lifeboat that you can play with, the sea is stormy with large breakers and the wind is blowing hard. The scene also contains a large ship rolling dangerously in the waves, an example script create an increasing mass to make it sink.

The scene has been optimized for fast rendering, you can increase its realism with an improved lighting and more particles.

Assets links :

- Ship 3D model (this asset is not anymore available on the store)
- Boat 3D model
- Skybox texture
- Boat engine audio clip

Surfing giant waves

This example scene aims to show off high speed watercraft and realistic surfing mechanics achievable with Storm Breakers. It contains a quite unrealistic ocean to foster this game mechanic with highly predictable surfable waves, and a speedboat that you control.

Assets links :

- Speedboat 3D model
- Skybox texture
- Engine audio clip

Fishing in deep blue sea

This example scene aims to show off what the Storm Breakers package is optimized for : a rough sea with a boat satisfying splashing on.

Assets links :

- Boat 3D model
- Boat engine audio clip

Sailing at the sunset

This example scene aims at showing off the achievable beauty of the water in Storm Breakers. It also shows off the sailing mechanics (still in preview). It contains a sail ship sailing with the wind in a calm sea with a sunset sky. Post processing effects are not implemented.

Assets links :

- Sail ship 3D model
- Skybox texture

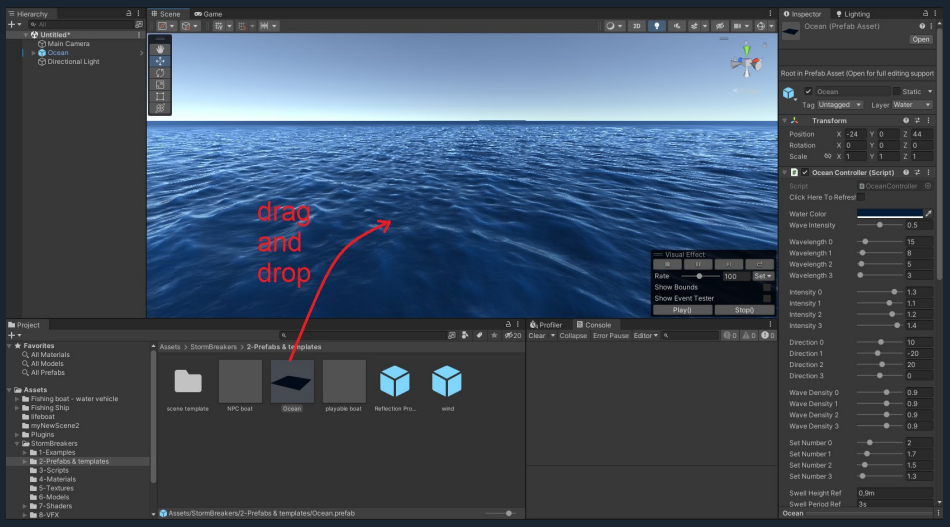
Prefabs references

In this section are described the prefab that you can use as templates to create various game objects. Unpacking the prefabs is recommended to have more freedom when modifying them.

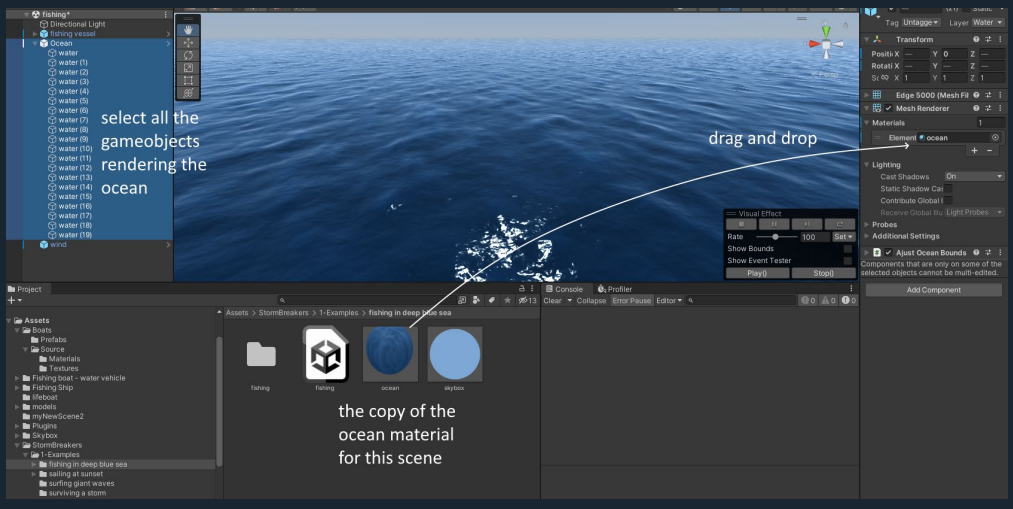
Ocean prefab

The ocean prefab should be used when you want to add an ocean in a scene that wasn't made with the scene template.

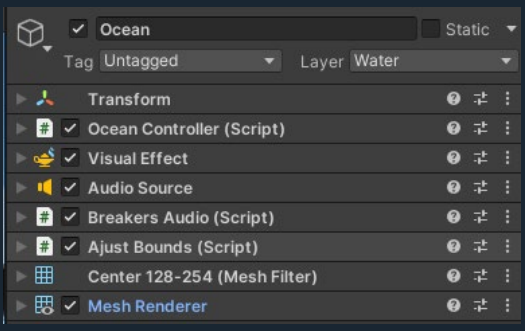
Simply drag and drop the prefab in the scene and the object will automatically set itself in front of the camera at y=0.



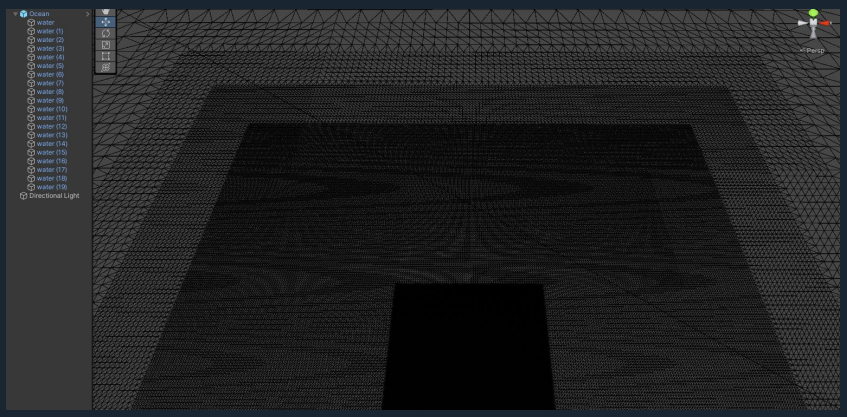
Then it is good to make a copy of the ocean material so you can make independent modifications on the water across different scenes. Assign it to this prefab and all the children as well.



This game object contains the required components to generate the ocean effects : waves, rendering, audio, visual effects. See their references in this document for more details.



The water meshes are separated by the childs so the frustum culling can prevent non visible meshes from being rendered. The splits are axis aligned to simplify the mesh shapes. The distant meshes have much less detail than the one close from the camera focus to optimize the rendering. If this setup doesn't suit your project you can change it with other meshes.



The ocean meshes seen from a far distance

The closet meshes have triangles of 50cm. This is enough for most sea states, but if you wish to render small waves (wavelength below 2m) on a close look, you can scale down the gameobject. This will increase the mesh density without changing the number of triangles in the scene.

Scaling down the ocean to have more detailed waves on a close look.

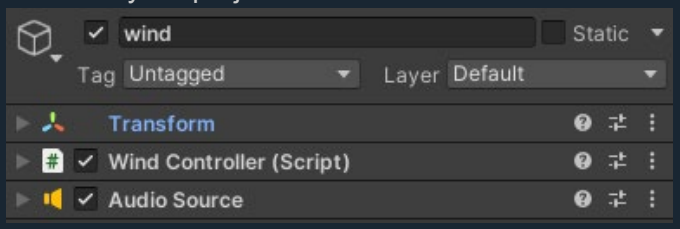
Note that doing this also brings back closer the fake horizon waves, so it works only when your camera focuses on near regions.

You can deactivate the VFX component when you don't need it (no breaking wave nor sea spray) to save on performance.

Wind prefab

The wind game object is not mandatory but allows to change the wind properties and generate its audio. If there isn't this object in the scene, the simulation will use the default value of 20km/h oriented in the opposite world red axis. What are impacted by the wind are the water surface smoothness and some physic components.

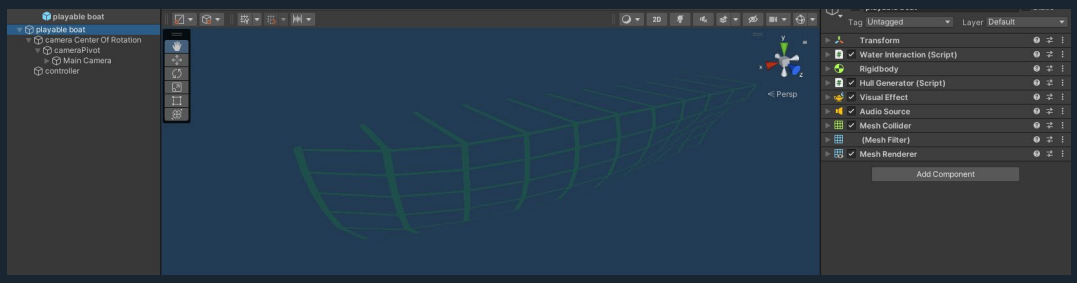
This game object contains the wind controller component and the audio source. You can change the audio clip for another one that better suits your project.



Its position within the scene has no impact whatsoever.

Playable boat prefab

This prefab is very handy to make a boat or ship interact with the water. It contains all you'll need to have a playable boat : the water interaction component, a rigidbody with a collider, the hull generator and its helper material, the visual effects, an audio source, a boat controller and the setup for an orbital camera.

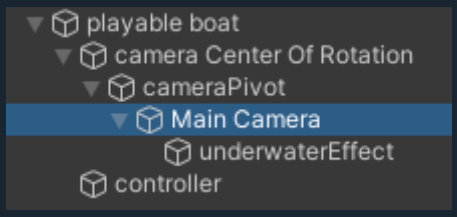


Set your 3D model child of this game object and then adjust all the properties like it is described in the getting started section and in the physics overview section of this document.

The 3D model set as a child of the prefab.

If you want to add colliders to the one used by the Water Interaction component (for example to make obstacles for a physical player walking on the deck), you can add them as childs. All colliders must be convex (all Unity built-in colliders are convex).

This prefab contains an orbital camera setup that works both for third person like or first person like, scrolling the mouse wheel makes it possible to change from one view to another. For this it use 2 empty gameobject, one serving as an anchor to track the boat without its rotation, and another child of this gameobject to make the camera rotation pivot. So you will find the camera (tagged as main) by expanding 3 times the hierarchy. This setup is provided as example.



Along with the camera is the setup of the underwater effect, that is a quad set in front of the near clipping plane with the underwater component and material. If you want to use another camera setup you should copy this hierarchy to have the underwater effect active in the game.

Non player boat prefab (NPC boat)

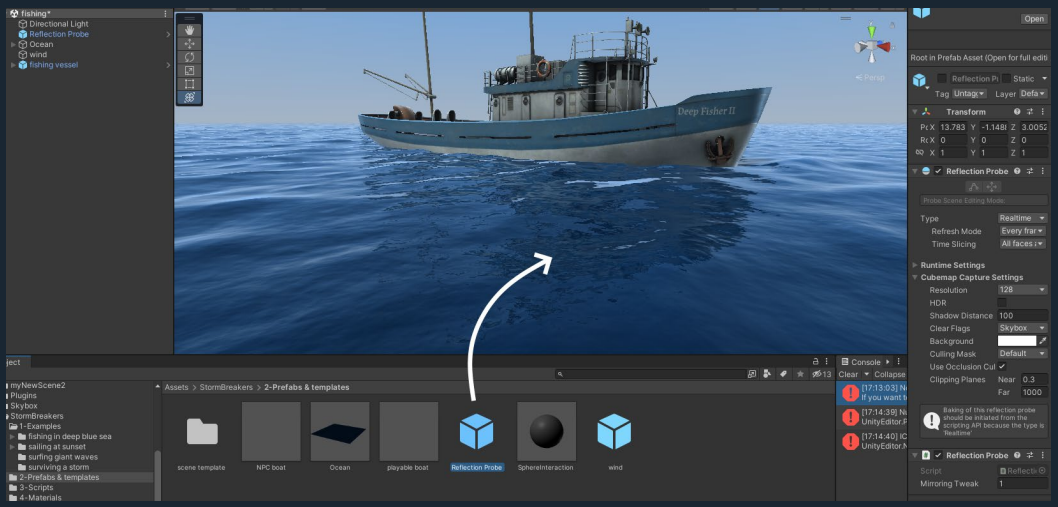
This prefab is very similar to the playable boat prefab. The two differences are the absence of the orbital camera setup and the Boat Controller component set to with the automatic pilot on.

Sphere interaction

This prefab contains the setup to make the Sphere Water Interaction component work : a rigidbody with a collider, the correct visual effect and an audio source. You can use it for round objects or objects that don't require a precise buoyancy like a character in the water.

Reflection probe prefab

Drag this prefab in the scene to activate the reflection of non static objects if the water is flat enough. It will be automatically set at the mirror position of the camera relative to the sea level.



You may have to tweak the reflection probe properties to fit your project quality and hardware, the default setup being of a low quality.

Scripts references

In this section are described all the scripts on both how to use them in editor (manual way) or via scripts (API way, with C# code written in yellow). The requirements to use these scripts are indicated in case you would add them yourself to a gameobject, so you must read a script documentation before using it. Always prefer using the prefab to avoid missing any requirements.

All scripts are implemented in StormBreakers namespace.

AjustOceanBounds.cs

This component is to be used for a mesh rendering the ocean. The component extends the bounds of the game object at startup according to the size of the waves so the frustum culling never goes wrong.

Requirements :

- A mesh filter component rendering the ocean should be attached to this gameobject.

BoatController.cs

This component provides the physics of a boat controller (propeller and rudder force) as well as audio and visual effect generation.

This component should be attached to a game object child of a rigid body with its position set where you want the forces to act, in general at the propeller and rudder position of your boat. Turning on the gizmos draws helpers on the scene to help you position the controller in the correct direction.

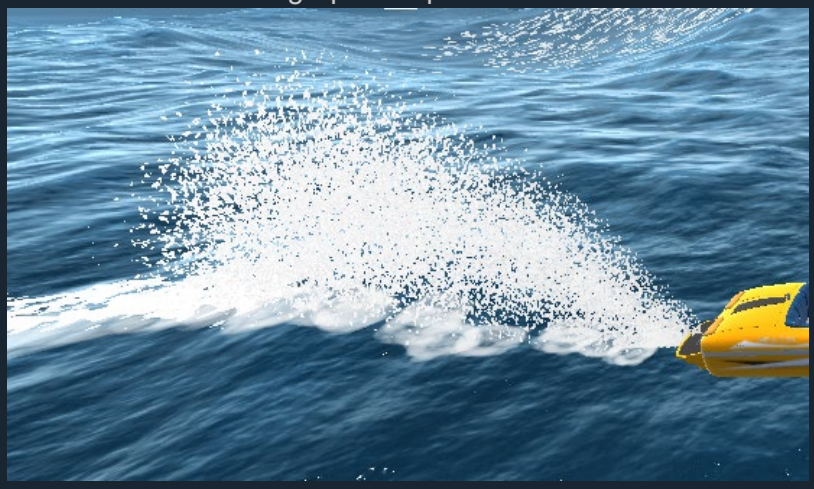
The potential forces are drawn with red lines in the editor,
the spot where the seam foam bursts are created is the white wire sphere.

The forces that the component generates in the water are quite simplified. They only evolve with the depth and so do the visual effects and audio. Future updates might include a more realistic boat controller including speed based force and audio.

The controller can be either piloted by an automatic pilot or by the player input. In that case the old input system is used with the axis named “Horizontal” for the rudder and “Vertical” for the propeller. Future updates might change the input system to the new one.

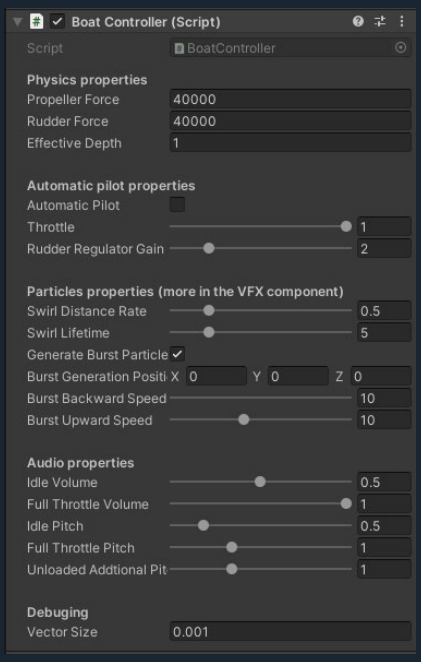
The visual effect graph boatControllerVFX.vfx is expected to be set along with this component. There are 2 kinds of particles generated, the swirls that are generated at a constant distance rate (when possible with the frame rate) directly in the water function of the propeller usage; and the sea foam burst that are generated function of the propeller usage and depth. When falling down the water, the sea foam burst

generates another particle of sea foam floats. Mind that some particles properties are set in this component while most of the others are set in the VFX graph component.



In the speedboat example, the seam foam burst is used to emphasize the power of the engine, we can also see the swirl and the floats on the left.

If there is an audio source component attached, this component will modulate its volume and pitch according to the propeller usage and depth.



property	function
Propeller Force	The force in Newtons that will be applied at the game object position to the parent rigidbody and in the local red axis direction when the axis “Vertical” will be activated.
Rudder Force	The force in Newtons that will be applied at the game object position to the parent rigidbody and in the local blue axis direction when the axis “Horizontal” will be activated.
Effective Depth	The depth in meter at which the controller makes full rudder and propeller forces, near the surface the forces are attenuated to simulate the lack of water to push on. Should be close to the propeller and rudder actual size.
Automatic Pilot	Whether to activate the automatic pilot for this controller or use the input system.
Throttle	The percentage of the propeller force used when the automatic pilot is set to on.
Rudder Regulator Gain	The gain of the regulator that maintains the course when the automatic pilot is set to on. High value makes a more responsive automatic pilot but can also lead to unstable behavior.

(hidden) Course	The vector that the automatic pilot will try to align its red axis to. This vector is automatically set to the current course when the automatic pilot is set to on, but you can modify this value at runtime by script to make an AI. Is expected to be normalized.
Swirl Distance Rate	The distance between each swirl particle's creation. Should be close to the swirl particle size that is defined in the VFX graph inspector.
Swirl Lifetime	The lifetime of the swirl particles when the propeller is at full force.
Generate Burst Particles	Whether to generate the sea foam burst particles.
Burst Generation Position	The local position where the burst particles are generated. You can visualize it by turning on gizmo mode, it's a white wire sphere.
Burst Backward Speed	The maximal local backward speed (m/s) of the burst particle generated near the water surface. You shouldn't need much because of the controller already moving when generating the burst particles.
Burst Upward Speed	The maximal local vertical speed (m/s) of the burst particle generated near the water surface.
Idle Volume	The volume of the audio source when the engine is idle.
Full Throttle Volume	The volume of the audio source when the engine is full throttle.
Idle Pitch	The pitch of the audio source when the engine is idle.
Full Throttle Pitch	The pitch of the audio source when the engine is full throttle and inside water.
Unloaded Additional Pitch	The additional pitch of the audio source when the engine is full throttle and outside water.
Draw Force	Whether to draw forces with lines in gizmos mode.
Vector Size	The size of the vector drawn in gizmos mode to visualize the generated forces.

API :

Method name	Return type	Description
UpdateVFXProperties	void	Is to be called when there are modifications on global sim data that might impact the visual effect attached (lighting, waves, wind).

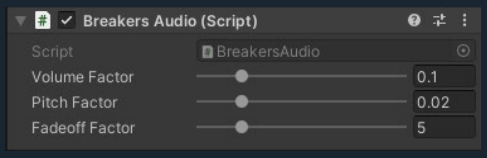
Requirements :

- This component should be attached to a gameobject that is a child of a gameobject with a rigidbody attached.
- A visual effect component splashVFX.vfx should be attached to this gameobject if you want to generate particles.
- An audio source component playing continuously should be attached to this gameobject if you want to generate engine audio.

BreakersAudio

This component generates a fully procedural audio of the breaking waves. To work it needs an audio source component playing without any audio clip. Ocean prefab include it, you can deactivate this component to deactivate the breaking waves audio generation.

The audio is not generated according to the breaking waves particles, but rather by the determinism of the waves. Sometimes we can spot a bit of difference between the audio and particles. See audio overview section of this document for more info on it.



Property	Function
Volume Factor	The volume of the breaking wave audio relative to their size. When too loud, saturation happens and decreases the audio quality. A bit of saturation sounds more powerful though.
Pitch Factor	Defines how high pitched the audio of the wave is. Low values make low audio that looks like bigger waves. Volume factor might need to be increased in case this value decreases, and conversely.
Fade Off Factor	Defines how the sound of the wave fades off with the distance from the camera (volume and pitch). The higher the value, the further we can hear the waves.

Requirements :

- An empty audio source should be attached to this gameobject.

CameraAnchor.cs

This component makes the game object unparent itself and keeps at runtime the same position as its former parent without its rotation. This is used to make the camera anchor object in the orbital camera setup provided by Storm Breakers.

In the orbital camera setup, the camera tracks down a rigid body. If the fixed time step is slower than the game frame rate, then there will be frames without the rigidbody updated by physics, causing the game to jitter. When this happens, either change the fixed frame rate to a faster one, or use the rigid body interpolation/extrapolation.

Note : a try has been made to get a bit of rotation from the former parent to provide more view motion and foster seasickness in case that would be the game mechanic. This attempt has failed because of the limitation of the quaternion interpolation. The solution for this effect might be to use the Cinemachine package.

Requirements :

- This component must be attached to a dedicated gameobject that is a child of the gameobject you want to follow.

CameraController.cs

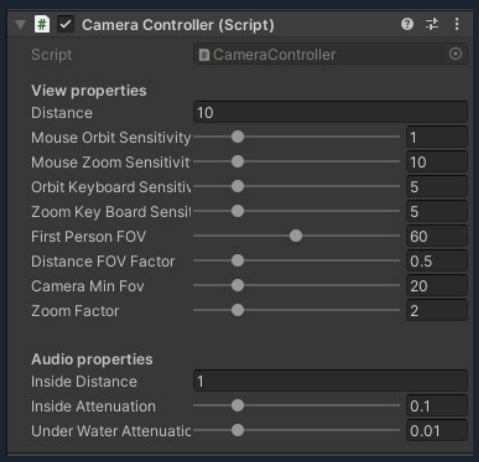
This component is part of the orbital camera setup that Storm Breakers provides as an example. This setup both allows an orbital view around the object it follows (third-person like) and a first-person like camera when the camera is brought back to its pivot. We can switch from one another by simply scrolling the wheel of the mouse.

This component manages the view rotation input with the mouse using the old input system, the axis name being “Mouse X” and “Mouse Y” for the mouse. Future updates might include the new Unity input system.

This component also makes a variation of field of view to emphasize the size of things when looked from far. This variation of field of view also happens when continuing to scroll up when the camera is brought back to its pivot, the field of view will decrease to simulate a zoom.

This component also manages the audio by applying a low pass filter function whether underwater, outside or inside. Camera being underwater is known thanks to the component DetectWater that is automatically added if needed. Being inside is defined solely by the camera distance from its pivot, this can be useful when the pivot is set inside the cabin of a boat.

This component is a bit basic and might lack functions that you need. You may modify the script or copy past the part that you need from this script without copyright issue (this is provided as an example).



Property	Function
Distance	The distance in meters at which the camera is from the center of rotation at start and runtime.
Mouse Orbit Sensitivity	The sensitivity of the mouse view.
Mouse Zoom Sensitivity	The sensitivity of the mouse zoom.
First Person FOV	The vertical field of view of the camera when in first person.
Distance FOV Factor	Makes the field of view varying with the camera distance from the center of rotation. Set 0 if you do not want a field of view variation.
Camera Min Fov	The minimal field of view when it varies with the distance.
Zoom Factor	Makes the field of view varying with the mouse continuing scrolling in first person and creates a zoom effect. Set to 0 if you don't want a zoom.
Inside Distance	The distance of the camera to the center of rotation below which the inside attenuation audio effect is activated. Set a negative value for no inside effect.

Inside Attenuation	The filter factor for the inside audio effect. The lower the value the more attenuated the audio will be.
Under Water Attenuation	The filter factor for the underwater audio effect. The lower the value the more attenuated the audio will be. Set 1 for no underwater attenuation audio effect.

Requirements :

- There must be a dedicated parent gameobject to act as a pivot.
- Using the old input system.
- A camera tagged MainCamera must be in the scene. This component should be attached to this main camera.
- A listener component should be attached to this gameobject.

DetectWater.cs

This component detects the presence of water at the game object position, public fields indicate it with a boolean and also returns the depth in meters (depth is positive underwater and negative outside).

This component can be useful to trigger an effect in relation with the water, like a leak in a boat that would increase the weight when underwater. The orbital camera setup contains this component to activate the underwater audio filter and the underwater visual effect when the camera is below the water surface.

API :

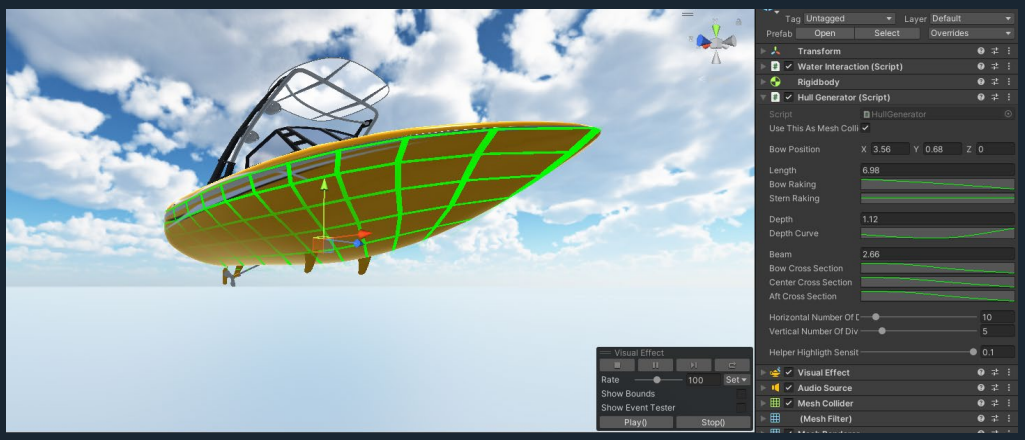
Property name	Type	description
IsUnderwater	bool (private set)	Is the gameobject underwater ?
Depth	float (private set)	The depth in meters, is positive underwater and negative outside.

Requirements :

- None.

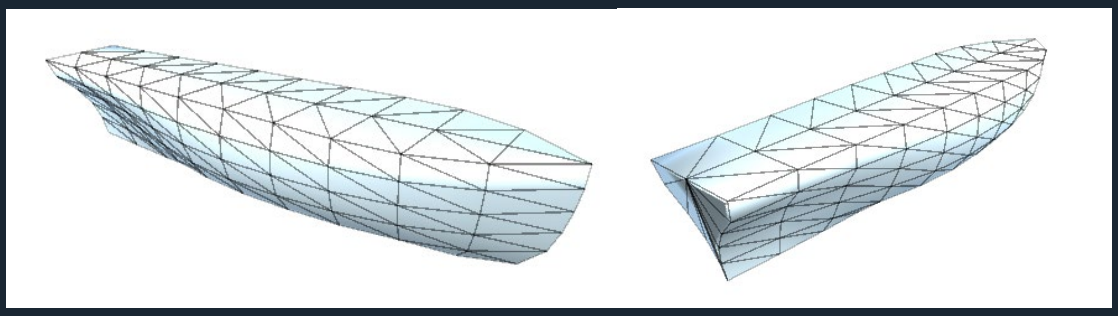
HullGenerator.cs

The hull generator is a component that generates procedurally boat shaped mesh. This is very handy if your boat 3D model does not contain a suitable collider mesh, with this component you can generate it and then optimize the number of triangles and tweak the shape to improve the dynamic attitude.



A view with the hull generator helper renderer activated.
Note how different the mesh is with the 3D model, this is to improve cornering attitude.

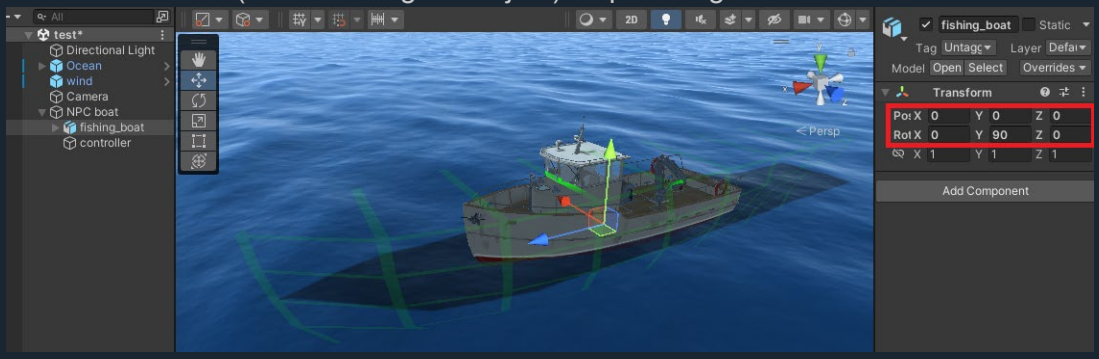
The procedural shape generated is currently limited to a boat-like hull with a sharp bow, a flat transom and a flat deck. The current limitation also makes it difficult to generate a hull that is straight for a long distance like a cargo ship, yet this is not much of a problem if the generated mesh isn't precisely in line with the 3D model, at least for the water interactions.



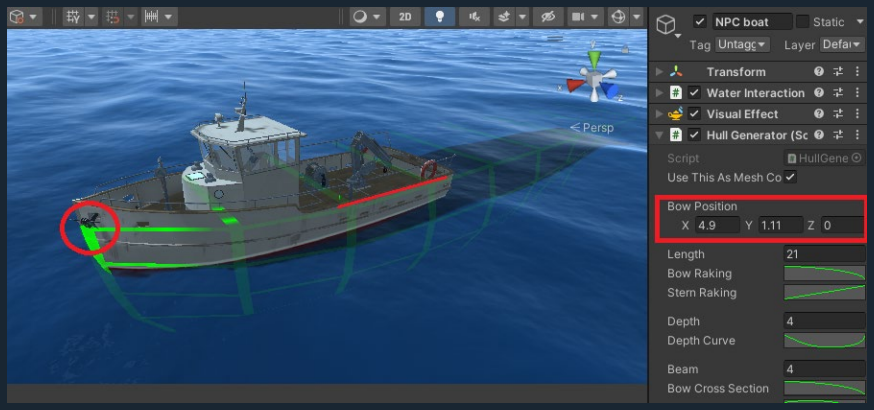
Setting a mesh renderer with the hullGeneratorHelper.mat material helps to visualize the generated mesh and how close it is to the geometry.

Here are the steps to follow when using this component :

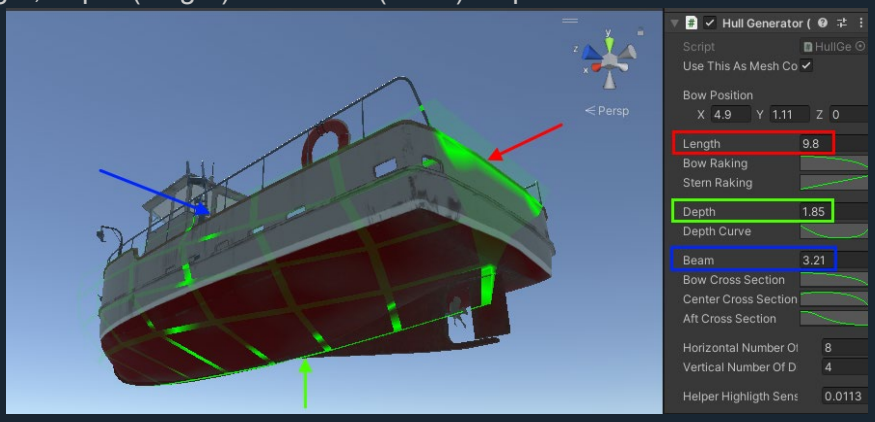
- Orient the 3D model (child of this gameobject) as per the generated hull



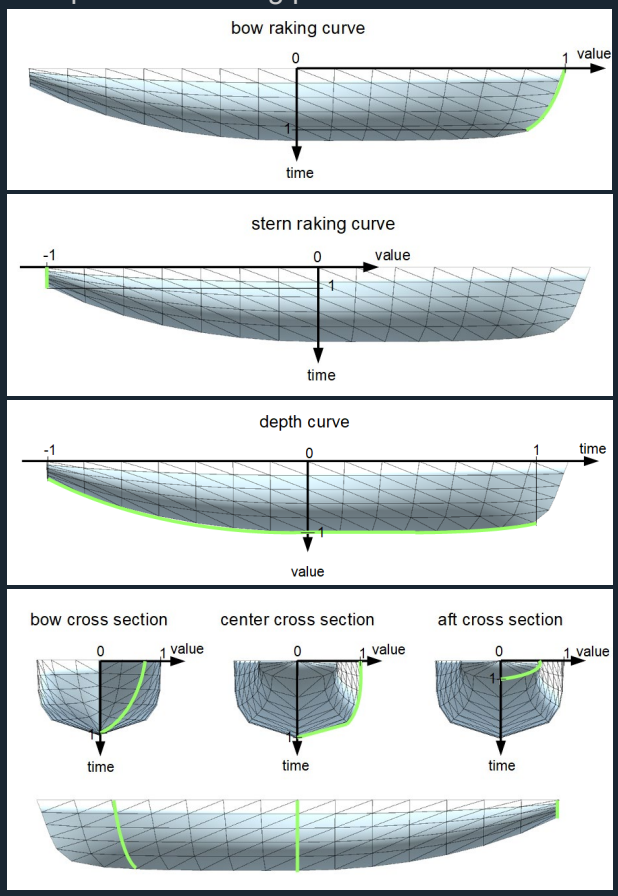
- Align the bow by modifying the Vector3 field in the inspector of this component. Set it as high the railing goes so it will be included in the simulation.



- Set the length, depth (height) and beam (width) as per the 3D model



- Set the raking curves, the depth curve, the beam curve and then the cross section curves to fit the best with the 3D model as per the following pictures :



- Set the horizontal and vertical number of division functions of your simulation : the more division there is the more precise and realistic the simulation will be, but also the more expansive. Try to make the divisions as square as possible.

Property	Function
Use This As Mesh Collider	Defines whether the mesh collider (if any) will be overwritten with the generated mesh.
Bow Position	The local position of the bow. You may include the railing so its surface can act on the water too.
Length	The length of the hull in meters. Factors the raking curves.
Bow Raking	Defines the factored bow position along the factored height. Top of the hull is at 0, bottom at 1.
Stern Raking	Defines the factored aft position along the factored height. Top of the hull is at 0, bottom at 1. Must be negative values.
Depth	The height of the hull in meters. Factor the depth curve.
Depth Curve	Defines the factored height of the hull along its factored length. Aft is at -1, bow at +1.
Beam	The width of the hull in meters. Factor the cross section curves
Bow Cross Section	Defines the factored width near the bow along the factored height. Top of the hull is at 0, bottom at 1.
Center Cross Section	Defines the factored width at the center of the hull along the factored height. Top of the hull is at 0, bottom at 1.
Aft Cross Section	Defines the factored width at the aft of the hull along the factored height. Top of the hull is at 0, bottom at 1.
Horizontal Number Of Division	Defines how many divisions there are along the length. The more there is, the more expansive the simulation will be but also the more precise. Keep divisions as square as possible.
Vertical Number Of Division	Defines how many divisions there are along the height. The more there are, the more expansive the simulation will be but also the more precise. Keep divisions as square as possible.
Helper Highlight Sensitivity	Defines how much the helper (when the corresponding material is set) glows when close to the geometry.

It is known that the interface of this component is not very user friendly. Future updates might include a new component with custom editor script that would allow for the use of bezier curves and handles directly within the scene editor.

Requirements :

- It should be attached to the same gameobject as the Water Interaction component to be effective.
- A mesh renderer with the material hullGeneratorHelper.mat should be attached to visualize the generated mesh.

Ocean.cs

Ocean.cs is a static class that allows every object in the scene to get access to the global simulation data like the waves properties, the wind and also the lighting. This class also contains the API to get the water data at a given position.

API :

Property	Type	Description
waterColor	Color	The albedo color of the water.
wavelength	float[4]	Each system's wavelength in meters. (L)
direction	float[4]	Each system's direction in degree. (d)
intensity	float[4]	Each system's intensity. Above 1 the waves break. (I)
randomization	float[4]	Each system's randomization value. When 0 the waves make a full density in the rhombus grid, it should be strictly inferior to 1. (r)
setNumber	float[4]	Each system's number of waves per group (or set). (n)
breakSpeedFactor	float	The factor used to compute the additional speed of the water when the waves are breaking.
breakTorqueFactor	float	The factor used to compute the torque created by the breaking wave function of the breaking speed.
wavenumber	float[4]	Each system's spatial frequency. (k = 2.Pi/L)
groupSpeed	float[4]	Each system's group speed, is half wave celerity in deep water. (v = 0.5.sqrt(g/k)
pulsation	float[4]	Each system's temporal frequency. (w = sqrt(g.k)
directionVector	Vector3[]	Each system's direction vector. (D = {cos(d),0,sin(d)})
iVector	Vector3[]	Each system's first vector of the rhombus grid. (I)
jVector	Vector3[]	Each system's second vector of the rhombus grid. (J)
Wind.speed	float	The wind speed in m/s.
Wind.inverseHeight	float	The inverse of the height at which the wind is full strength.
Wind.cosDirection	float	The baked cosine of the direction of the wind.
Wind.sinDirection	float	The baked sine of the direction of the wind.
Wind.direction	float	The direction of the wind in degree relative to the world red axis.
Wind.turbulenceAmplitude	float	The amplitude of the turbulences relative to the wind strength.
Wind.turbulenceFrequency	float	The frequency of the turbulences.
totalLight	Color	The sum of the ambient light color and the halved directional light color.
areBreakers	bool	Are there breaking waves in the ocean ?
surfaceIntensity	float	Defines the maximal slope of the water surface including ripples and waves.
sharedMaterial	Material	The material shared by all ocean renderers.
LIDR	Matrix4x4	The matrix that clusters each system wavelength, intensity, direction and randomization. Each line corresponds to a system.

NKVW	Matrix4x4	The matrix that clusters each system set number, spatial frequency, group speed and temporal frequency. Each line corresponds to a system.

Method name	Return type	Description
ConstructStaticData	void	Use this function at startup because static data can't be managed.
BakeAndPackWaveData	void	Compute the baked wave data with scientific formulas and pack all the data to the matrix that can be used by material and visual effect.
OceanDeformation	Vector3	Compute the deformation caused by the waves at 'time' and at 'position'. This also computes the additional speed caused by the breaking waves.
GetHeight	float	Use this function to get the height of the water below or above position and update the undeformed position.
GetNormal	Vector3	Compute the water normal given data that have already been computed with GetHeight. This function is costly and should be performed only when required.
GetVelocity	Vector3	Compute the water velocity given data that have already been computed with GetHeight. The depth is required because there is a speed attenuation based on it.
GetWindSpeed	float	Compute the wind speed in m/s at position height. The wind is half strength at y=0 and full strength above wind height.
GetWindVelocity	Vector3	Compute the wind velocity in m/s at position height. The wind is half strength at y=0 and full strength above wind height.

Ocean.ConstructStaticData

- Declaration : static public void ConstructStaticData();
- Description : Constructs the static arrays with a size of 4.

Ocean.BakeAndPackWaveData

- Declaration : static public void BakeAndPackWaveData();
- Description : Computes baked wave data (k, v, w, D, I, J) and saves all wave data in the matrices LIDR and NKVW.

Ocean.OceanDeformation

- Declaration : static public Vector3 OceanDeformation(float time, Vector3 undeformedPosition, out Vector3 breakingVelocity);

- Parameters :
    - time : The time at which the deformation is computed. Use Time.time to have the same time as the vertex shader rendering the ocean.
    - undeformedPosition : the position from which the waves are computed, see Physics section of this document for more info on this parameter.
    - breakingVelocity : returns the additional velocity caused by the breaking waves. This velocity is in the horizontal plane, in the y component of this vector is stored the height from the surface at which the breaking velocity and torque should apply above water.

- Description : Returns the sum of each system deformation at a given point and time. Add this deformation to the undeformed position to get the world space deformed position. Use the outed parameter to get the breaking additional velocity, if you don’t need it set out _

Ocean.GetHeight

- Declaration : static public float GetHeight(float time, Vector3 position, ref Vector3 previousUndeformedPosition, out Vector3 deformation, bool recomputeOcean = true);

- Parameter :
  - time : The time at which the water height is computed (this is because C# jobs can't read the game time) . Use Time.time to have the same time as the vertex shader rendering the ocean.
  - position : The position below or above which the water height will be computed.
  - previousUndeformedPosition : The position at the previous frame from which the water deformation caused the water to be below or above position. Pass this parameter as a variable to update it to help the algorithm get the water height vertically closer to position.
  - deformation : The deformation of the ocean at the undeformed position. Save this result for the GetVelocity() and GetNormal() method. Add this to the undeformed position to get the water world space position that vertically matches with position.
  - recomputeOcean : whether to recompute the ocean once more at the undeformed position. This is necessary to get a precise water velocity and normal. You can save on performance if you don't need precision by setting false.

- Description : Return the height in world space of the water below or above a given position. The algorithm needs the undeformed position of the last frame to get the water read vertically closer to the position, when the horizontal difference is larger than a meter the algorithm runs another loop. See Physics overview section of this document for more info on this algorithm.

Ocean.GetNormal

- Declaration : static public Vector3 GetNormal(float time, Vector3 undeformedPosition, Vector3 alreadyComputedDeformation, float precision = 0.1f);

- Parameters :
  - time : The time at which the normal is computed. Use Time.time to have the same time as the vertex shader rendering the ocean.
  - undeformedPosition : The undeformed position of the water from which the normal will be computed. Set it to the variable value that was passed as ref previousUndeformedPosition in the GetHeight() method.
  - alreadyComputedDeformation : The resulting deformation of the GetHeight() method used to avoid computing it again.
  - precision : The spacing in meters of the water samplings. The larger, the smoother the normal will vary.

- Description : Returns the water normal given data that have already been computed with GetHeight. This function is costly and should be performed only when required.

Ocean.GetVelocity

- Declaration : static public Vector3 GetVelocity(float time, Vector3 undeformedPosition, Vector3 alreadyComputedDeformation, out Vector3 breakingTorque, float depth = 0f, float dt = 0.1f)

- Parameters :
  - time : The time at which the velocity is computed. Use Time.time to have the same time as the vertex shader rendering the ocean.
  - undeformedPosition : The undeformed position of the water from which the velocity will be computed. Set it to the variable value that was passed as ref previousUndeformedPosition in the GetHeight() method.
  - alreadyComputedDeformation : The resulting deformation of the GetHeight() method used to avoid computing it again.
  - breakingTorque : the torque generated by the vortex of the water, should be multiplied by the affected surface area and the water density (1000). Is currently only used in WaterInteraction.
  - depth : the depth of the object in the water with positive value underwater. Is used to attenuate the velocity and breaking torque when the object is deep in the water.
  - dt : The delta time between the two water sampling.

- Description : Returns the water velocity caused by the wave and includes the total additional breaking speed when there are breakers here. It also returns the breaking torque.

Ocean.GetWindSpeed

- Declaration : static public float GetWindSpeed(Vector3 position, bool computeTurbulence = false)

- Parameters :
  - position : the world space position where to compute the wind.
  - computeTurbulence : whether to compute the turbulence.

- Description : return the wind strength at a given position. This method doesn't use a vector normalization.

Ocean.GetWindSpeed

- Declaration : static public float GetWindVelocity(Vector3 position, bool computeTurbulence = false)

- Parameters :
  - position : the world space position where to compute the wind.
  - computeTurbulence : whether to compute the turbulence.

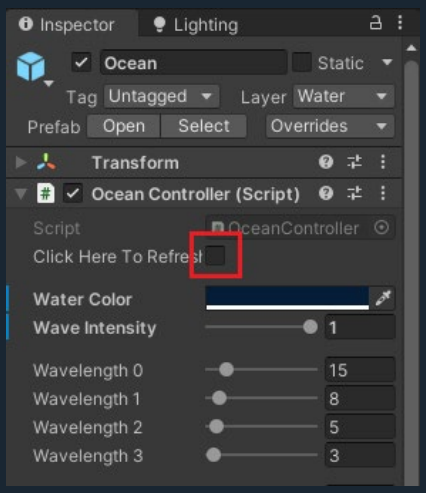
- Description : return the wind velocity at a given position.

OceanController.cs

The ocean controller component is a mandatory component to run the simulation. Not only does it provide an interface to set the waves property, it also manages the static class Ocean so to make the link between the global data and all the components in the scene. This includes the wave properties, the wind properties, but also the lighting (for the particles).

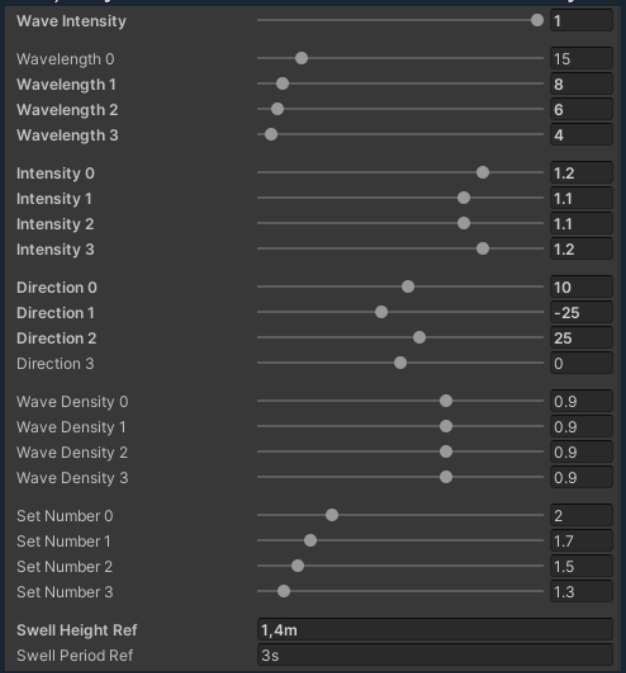
This component also moves the ocean roughly to the camera focus (and not the camera position). This in order to have the highest mesh density rendered close and the ocean visual effects having its bounds further.

In the editor, this component updates the global data only when a modification is done in the inspector (using the message OnValidate). So if you update the lighting, the particles might be ill lit while staying in the editor, you can click on the box Click Here To Refresh to update them. This might also be needed after a scene save or after an asset import : because the material matrices are not serialized they can get back to a previous value. Clicking on this box reset back the matrices with the correct inputs.



Click here when the wave are wrong in editor or to refresh particles in editor

The waves properties are set through a series of sliders, you can control every aspect of the waves. The fact that the properties are bound by sliders is only to provide a better user interface (like it is for most of the Storm Breakers components). If you need value outside the bounds you may modify the code.



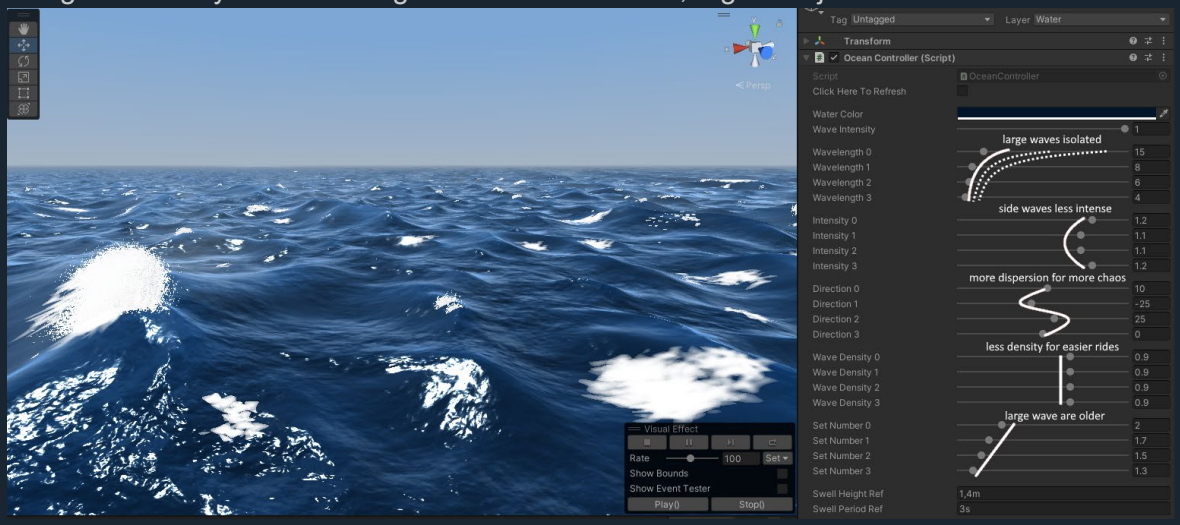
The height of the waves is not a direct input, but a result of several inputs (at maximal intensity a wave height is 0.17 times its wavelength). Readonly fields in the inspector tells you about the average wave height for the swell (first system) as well as its period. The wave intensity affects the wave height and defines how likely the waves are going to break, which happens above 1.

Wave Property	Function
Click Here To Refresh	Update the materials and VFX in editor.
Water Color	The albedo color of the water. In deep clear water it must be very dark blue as the light ray penetrates deeply into the water and scatters few light rays. In shallow water the color can be brighter with more yellow/white coming from the sand color. In dirty water the color can be bright too as the light ray scatters quickly when penetrating the water.
Wave Intensity	Allows you to quickly set all the waves on or off.
Wavelength x	The wavelength in meters of the wave system x. Big wavelength makes bigger waves. Should be smaller than the one of the previous system.
Intensity x	The steepness of the wave system x. Above 1, waves start to break.
Direction x	The direction in degree of the wave system x relative to the opposite red world axis.
Wave Density x	Wave density of the wave system x. Reduce this value to get a more scattered and random wave pattern. This value also affects each wave group intensity so you might update the system's intensity as well.
Set Number x	Number of waves per group (or set) in the system x. High numbers are only suitable for an old swell.

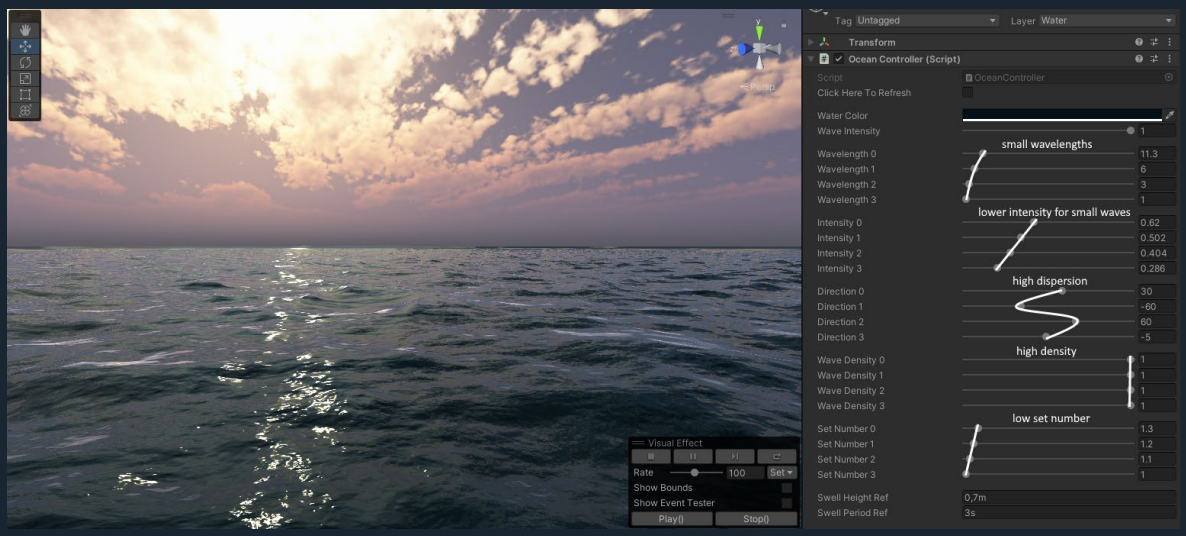
Global tweaks	Function
Breakers Extra Speed Factor	The factor used to compute the additional velocity of the breaking wave. Set it to zero if you want no breaking physic effect.
Breakers Torque Factor	The factor used to compute the torque of the breaking wave. Set it to zero if you want no breaking torque effect.
Fixed Frame Rate	The fixed frame rate used in physics. Because the objects in water have in general slow acceleration, this value can be trimmed down to improve FPS and avoid CPU overhead peaks (beware of the jittering if the camera tracks a rigidbody)
Particle Lighting Factor	The lighting of the particles is computed with this component, but sometimes it gives the wrong intensity and you can tweak it with this value.
Ocean Object Displacement With Camera Angle	Make the gameobject ocean move to this distance in camera forward direction to set the detailed mesh and VFX in focus.

Here are some examples on how to set up realistic waves.

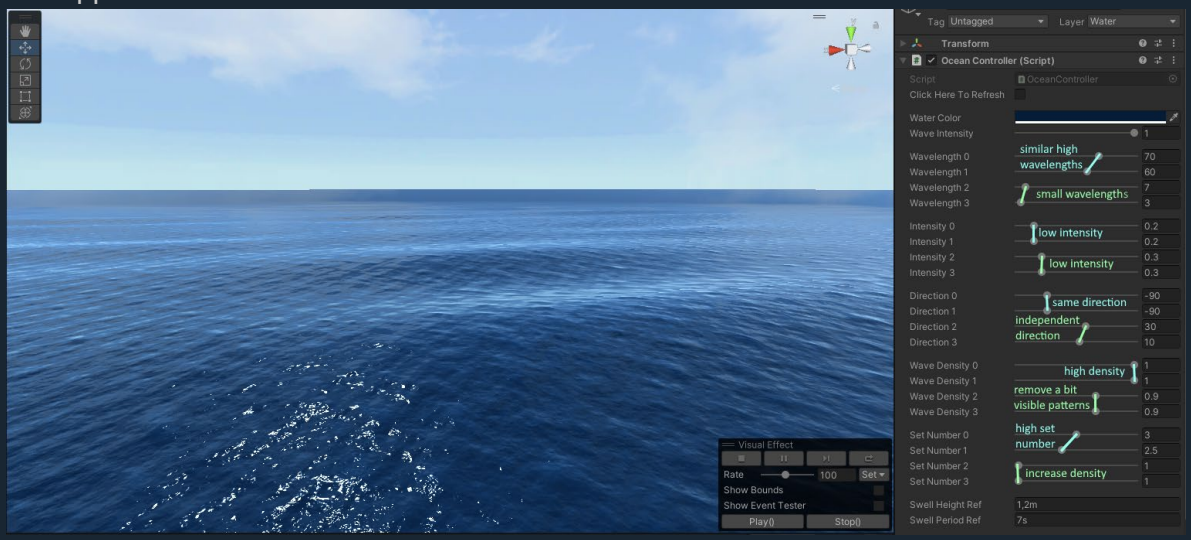
- Rough sea : vary the wavelengths to render a storm, a gale or just a breeze.



- calm sea :



- Old swell : use 2 systems for the swell to have enough swell waves, and set the other system for the ripples.



You can also set up the waves in a non realistic way for gameplay purposes. This is what is done in the example of the speedboat surfing giant waves. The first system is isolated with a very large wavelength, high intensity and high set number. This way the waves are very predictable and easy to surf for a long ride.

Changing global data at runtime

It is possible to change the wave properties, the wind, or the lighting at runtime. **If you change the waves properties at runtime, you should do it very slowly to avoid any artifacts happening.** When doing that through script, you must ensure that every component in the scene will be updated as well, especially the visual effects that need to have their properties updated since they don't refer to the static class. Here are the steps to follow when updating global data in runtime :

- Changing waves properties at runtime
    1. Change slowly the public fields of ocean controller by script
    2. Call OceanController.UpdateWaves() to update static data and the ocean gameobejct's components.
    3. Send the message "UpdateVFXProperties" to every Storm Breakers component generating VFX graph particles (WaterInteraction, BoatController, SphereWaterInteraction). This will update their properties.

- Changing lighting at runtime
    1. Change the lighting (main light, ambient)
    2. Call OceanController.UpdateLighting() to update static data and the ocean gameobejct's components.
    3. Send the message "UpdateVFXProperties" to every Storm Breakers component generating VFX graph particles (WaterInteraction, BoatController, SphereWaterInteraction). This will update the VFX properties.

- Changing wind at runtime
    1. Change the public field of the component WindController of the corresponding gameobject
    2. Call OceanController.UpdateWind() to update static data and the ocean gameobejct's components.
    3. Send the message "UpdateVFXProperties" to every Storm Breakers component generating VFX graph particles (WaterInteraction, BoatController, SphereWaterInteraction). This will update the VFX properties.

API :

Method name	Return type	Description
UpdateWaves	void	Update waves related static data and the ocean gameobject's components.
UpdateWind	void	Update wind related static data and the ocean gameobject's components.
UpdateLighting	void	Update light related static data and the ocean gameobject's components.

Requirements :

- There must be one and only one OceanController component in any scene using Storm Breakers features.
- There must be a mesh renderer with the material ocean.mat attached to this gameobject.
- The visual effect component oceanVFX.vfx should be attached if you want to render ocean particles.
- The ambient light mode must be set to either a solid color or a gradient.
- A camera tagged MainCamera must be in the scene.

ReflectionProbeController.cs

Storm Breakers provide an approximate way to make the reflection of dynamic objects in the water without using the screen space reflections (SSR) that are unavailable in Unity URP. The trick is to position a reflection probe as a mirror to the camera relative to the sea level. This component does this positioning when attached to the reflection probe gameobject.

Using a reflection probe instead of SSR can make a bit of improvement on the water reflection because some geometries become visible, but mainly some defaults. First it's quite expensive for the GPU as it acts as a second camera rendering most of the scene, to solve this you should reduce its resolution and how frequently it is updated, but you'll have much less quality. Secondly it's not accurate enough in the waves, this is why there is a tweak factor to reduce the reflections angle. Though, on glassy water and with good hardware it is possible to render nice reflections.

When there are more waves, the reflection probe increases its depth in its mirror position relative to the camera to reduce the reflection that happens *behind* the objects. In this screenshot we can see some wrong reflection between the sails :

To reduce this you should increase the tweak factor but this reduces the correct reflection as well.

Property	Function
Mirroring Tweak	How much the reflection probe goes far from the camera function of the water surface roughness to reduce wrong reflection behind objects.

Requirements :

- Being in a dedicated gameobject.
- A camera tagged MainCamera must be in the scene.
- A reflection probe component should be attached to this gameobject.
- The waves shouldn’t be too big.

SphereWaterInteraction.cs

The Sphere Water interaction is similar to the Water Interaction component, but is optimized for spheres in the way that it computes the water at only 1 point. It can be used for spherical objects like buoys or objects floating in the water that don't need much precision.

Buoys floating with a sphere interaction component and a custom script for their anchorage

Like Water Interaction, this component will both make physics, visual effect and audio effect. For this it expects the following components to be attached to the same gameobject : a rigidbody, a VFX graph with splashVFX.vfx and an audio source.

Everything is coded so that the effects scale with the size of the object, but some tweaking is inevitable to adjust the way you want it to interact. Knowing how it works helps you setting up the component and its dependencies (vfx, rigid body and audio).

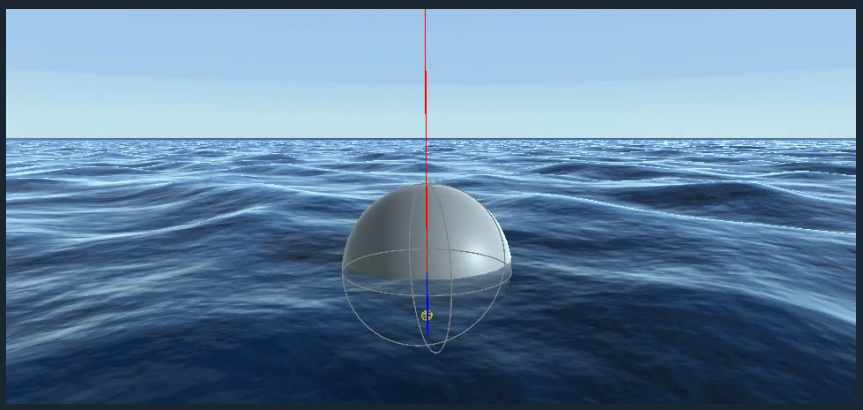
The physics works by creating 2 different forces on the rigidbody. The buoyancy force is proportional to the volume immersed and oriented in the water normal, acting on the center of the sphere. So in the waves the sphere with a low gravity center will align with the local slope and slide down. The drag force evolves with the square of the relative velocity of the sphere to the water. The fact that it evolves with the square speed makes a high force at high speed, but also very low forces at low speed. Because of the numerical approximation and this consideration, it is possible that the sphere oscillates a bit on calm water. What misses is a drag force that evolves proportional to the relative speed to dampen slow movement. To solve this you may **use the built-in drag coefficient of the rigidbody**, even though this drag force will act regardless of being in the water or not. (This has been chosen because it would have been difficult to set up a realistic linear drag that scales with the object.)

There are 2 kinds of particles generated, the sea foam burst when the impact speed (here is the same as the relative velocity) exceeds a trigger, and the sea spray when the product of the impact speed and the wind exceeds a trigger. When the seam foam burst falls down on the water, it generates another kind of particle, the sea foam floats. The position of the floats precisely match the one from the burst that generated it even in the waves, but the size and the difference of texture might perturbate the precision of this effect. Mind that some VFX properties are set in the inspector of this component while most of the others are set in the VFX inspector. See the splashVFX references for more details on how to set up visual effects.

The audio is generated by modulating the pitch and volume through a hand-coded filter. Both depend on the dynamic pressure, here is the same as the relative velocity : the faster the louder and lower the audio

will be. Then the audio pitch and volume fade off at independent rates. Storm Breakers provides a suitable audio clip for this effect (splash.wav).

You can turn on the gizmo mode to visualize the simulated sphere (gray wire sphere) and its gravity center position (yellow wire sphere). In play mode you can also visualize the buoyancy force (red line) and drag force (blue line).



Physics Properties	Function
Generate WaterForce	You can choose whether to generate forces. The rigidbody useGravity property will be set to the same. If set to false the force will still be computed and drawn in the gizmos.
Radius	The radius in meters of the simulated sphere.
Gravity Center Height	The position of the gravity center in meters. When positive the gravity center is on the local green axis side.
Density	The relative density of the rigidbody (Ratio of the mass and the water mass for the same volume). Above 1 it sinks.
Drag Coeff	The drag coefficient in every axis. https://en.wikipedia.org/wiki/Drag_coefficient
Inertia Tensor Factor	The multiplier of the moment of inertia in every axis. The rigidbody component computes the inertia tensor based on the hypothesis of a full sphere, so for a hollow object you should increase the inertia by setting this multiplier superior to 1. https://en.wikipedia.org/wiki/List_of_moments_of_inertia

Particles properties	Function
Generated Particles	Whether to generate the particles.
Min Impact Speed	The minimal relative speed of the water to generate particles.
Burst Count	The number of burst particles generated per fixed delta time frame. Depends on the fixed frame rate.
Spray Count Factor	The number of sea spray particles generated per surface area, per relative velocity and per wind velocity.
spray MaxCount	The maximum number of sea spray particles generated per fixed delta time frame.
Spray Lifetime Factor	The sea spray lifetime per impact speed and wind strength.

Audio properties	Function
Generate Audio	Whether to generate audio or not.
Audio Min Impact Speed	The minimal water impact speed at which a triangle adds dynamic pressure for the use of audio computation. Increase this value to avoid audio when the mesh is moving slowly.
Volume Factor	The audio volume per dynamic pressure acting on the mesh.
Max Volume	The maximum audio volume, can exceed 1 to get a bit of saturation and because the pitch effect reduce the audio intensity
Volume Fade Off Rate	The rate at which the audio loses volume after a blast.
Pitch Factor	The audio pitch per dynamic pressure acting on the mesh. You might need to adjust volume level when modifying this property.
Min Pitch	The minimal pitch applied. Too low pitch reduces the power of the noise, so you should actually increase it for large watercraft to get more audio power even though it leads to wrong pitch.
Pitch Fade Off Rate	The rate at which the audio got back to high pitch after a blast.

Debugging properties	Function
Draw Force	Whether to draw forces with lines in gizmos mode.
Vector Size	The size of the vector drawn in gizmos mode to visualize the forces.

API :

Method name	Return type	Description
UpdateVFXProperties	void	Is to be called when there are modifications on global sim data that might impact this visual effect attached (lighting, waves, wind).

Requirements :

- A rigidbody must be attached to this gameobject.
- The splashVFX.vfx visual effect component should be attached if you want to generate particles.
- A 3D audio source with splash.wav playing permanently should be attached if you want to generate audio.

StaticBuoyancy.cs

The Static Buoyancy component provides a very fast way to make objects float kinematically, meaning without any physics. This component will simply apply the wave's displacement to the transform based on its start position. It does not apply any rotation though. This can be enough for small pickable objects floating in the water.

Requirement :

- None

UnderwaterEffect.cs

Storm Breakers provide a very basic underwater effect in case the camera would go underwater. You shouldn't rely on it to render realistic underwater scenes.

This effect does not segregate precisely the pixels in the screen that are actually underwater and the ones that are not, it’s a global fog effect that activates as soon as the camera is near the water surface or below. To make this fog, a quad in front of the camera appears and renders with a depth-dependent transparency shading (see underwater material reference for more info on its shading) This component activates this quad rendering whether the camera is considered underwater.

The threshold at which the camera is considered underwater can be adjusted to reduce the issues when rendering with the water limit visible on the screen (objects above water are fogged while they shouldn't). Another way to reduce this effect is to reduce the near clipping plane of the camera to almost zero, this way there will be less time with the water limit visible on the screen. In that case you may position the quad rendering the underwater effect to a bit behind the new near clipping plane.

This component also modifies the rendering order of the ocean material. Indeed while underwater, the depth effect shader needs to have access to the water screen depth, so the water surface must be rendered in the opaque queue. Outside water, the ocean material can be either in the opaque or transparent queue whether its transparency is set to 1 or below respectively.

Property	Function
Depth Trigger	The depth of the camera in the water at which the underwater activate. When negative, the underwater activates before the camera is fully inside water.

Requirements :

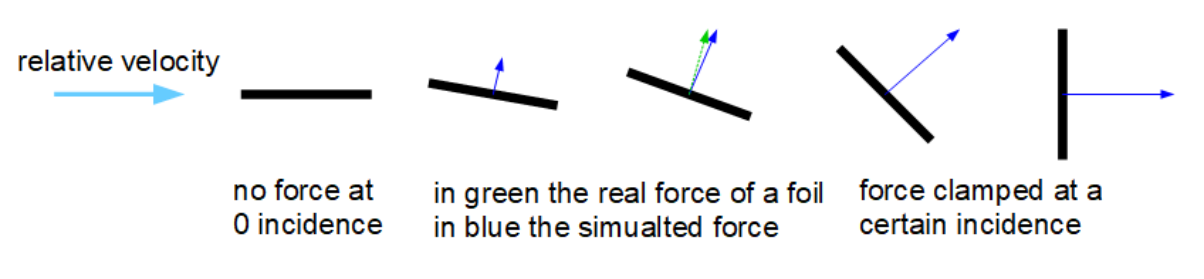
- A camera tagged MainCamera in the scene.
- A quad renderer with the material underwater.mat attached to this gameobject
- This gameobject should be a child of the main camera, positioned just behind the near clipping plane, and sized to render to the whole screen.

Foil.cs

The Foil component makes a simplified lift and drag force of a foil in the water or in the air. This component can simulate fins, rudders, keels or even sails. It can be very handy to stabilize a watercraft roll or a speedboat trajectory.

It works by computing the force on each triangle of the mesh coming from the mandatory mesh filter component, like does the Water Interaction component. Yet you don't need a complex mesh to make a proper force, most of the time a quad or even a single triangle will suffice.

The force it generates is always normal to the triangles of the mesh, acting on its center, proportional to the squared relative speed, the area, the density of the medium (air or water) and function of the incidence angle (computed with the dot product of the normal and the relative speed). This is a simple way to simulate both the lift force and induced drag force of a foil, the worst approximation comes from the fact that the orientation of the force acts like the foil would make the fluid bounce off the foil, while in reality the fluid at max gets aligned with the surface. Thus the simulated incidence angle is twice the one that we would expect, making the drag force building up more quickly. You should take that limitation in account, especially when simulating sails, the incidence angle of the simulated mesh should be halved at least (when you can vary it).



The fact that the force evolves with the square of speed makes this component very efficient at high speed even with a low incidence angle. If you use this component to make a sail force, when going upwind or sidewind the sailboat velocity will add to the wind velocity, thus making a stronger sail force ! (Like it is in reality)

In the sailing example, the sails force and the keel are modeled using invisible triangles, the wind acting on the sails thanks to the wind controller. Note how much the incidence of the sails is different from the 3D model sails, this is to provide more lift than drag, thus limiting the helling.

In the speedboat example, a fin acting on an invisible quad is set near the rudder to increase roll stability at high speed, to stabilize the trajectory. It also increases steering force at high speed thanks to the InputRotation component.

If you don’t want the increased lift force so the mesh acts as a non profiled fin instead of a foil, you should set the lift coefficient to 1. Then the force it generates is similar to the dynamic pressure of the Water Interaction.

Properties	Function
Flat Water	Tip this when the water is flat, or when the surface is always too far or when the object is away from focus, this will prevent the waves from being computed and save some resources.
Lift Coeff	The lift coefficient (https://en.wikipedia.org/wiki/Lift_coefficient) The higher, the more force it will generate at low angle of attack only, at higher angle the force is clamped.
Maximal Pressure	The maximal pressure applied due to the speed. Since it evolves with the square of speed, this can reach higher values that the physic can handle for small objects moving fast. Lower it only when the object makes wrong moves.
Pressure Excess Occurrences	Indicate the number of times the maximal dynamic pressure is reached, if it happens too often this means the physics might be impacted and you should consider raising the maximal pressure if there aren't troubles with it. Doesn't affect builds.
Draw Force	Whether to draw forces with lines in gizmos mode.
Vector Size	The size of the vector drawn in gizmos mode to visualize the forces.

Requirement :

- A mesh filter component attached to the same gameobject.
- The gameobject must be a child of an object with a rigidbody component attached.

CameraTravel.cs

This component allows you to activate the automatic pilot of a given Boat Controller component using a named keyboard key (old input system). It also implements a local camera movement that happens with the horizontal and vertical axis when the autopilot is set to on, for that it must be attached to the same object as the Camera Anchor component .

This component is provided as an example and its code can be copied.

Properties	Function
Boat Controller To Automate	The Boat Controller component for which the automatic pilot will be activated or deactivated.
Toggle Automatic Pilot Key	The key that toggles the automatic pilot.
Speed	The local camera motion speed when the automatic pilot is on, using Horizontal and Vertical axis.

Requirements :

- This component should be attached along with the Camera Anchor component.

WaterInteraction.cs

Water Interaction is a very versatile component that you can use to simulate almost any object interacting with the water, from a small cube to a large ship, without having to tweak much its properties. This component, along with the visual effect splashVFX.vx and an audio source, will make the physics, visual effects and audio effects.

It works using an implicit simulated mesh that is preferably a low poly for performance reasons. This mesh is automatically picked from the convex mesh collider (a required component), or from the Storm Breakers Hull Generator component if there is one.

At runtime, the component will compute per triangle the dynamic and static pressure and the particle generation. The audio will be computed with the total dynamic pressure acting on the mesh. So a very large number of triangles will lead to quite a CPU overhead, the most expensive is computing the waves for each triangles with the algorithm. This is why it is recommended to use the Hull Generator component when suitable, you will have the possibility to easily optimize the number of triangles.

The physics is computed by calculating the dynamic pressure (evolve with the square of the relative speed) and static pressure (proportional to the depth) on each triangles of the mesh. The pressures are summed and clamped to positive value (simulating water cavitation) and multiplied to the triangle area, oriented in the triangle normal and acting on its center. The fact that the buoyancy force is not vertical can be non intuitive but it’s actually realistic (see Physics Overview section of this document). You can set them vertically in case they would lead to a non vertical net force in almost flat water, making the object moving on its own.

The use of the Unity job system can be activated in this component to compute the waves for each triangles in multithread (the forces computation can’t be part of the multi thread yet) In some cases this can increase performance, you have to check it with your setup. Further attempts in using Unity DOTS were unfruitful but they might be implemented in future updates to continue improving frame rate.

The physic also includes the possibility to simulate the wake wave force. Like explained in the Physic Overview section of this document, this effect adds virtual depth based on the height of a wave moving at the relative speed. With this effect some triangles might get underwater or not while they would not otherwise, making more static and dynamic pressure where the relative speed pushes on the mesh, and less pressure where the relative speed goes away from the mesh. This can improve a lot the cornering and surfing attitude, especially for V shaped hulls at high speed. So you can simulate speedboats with this component.

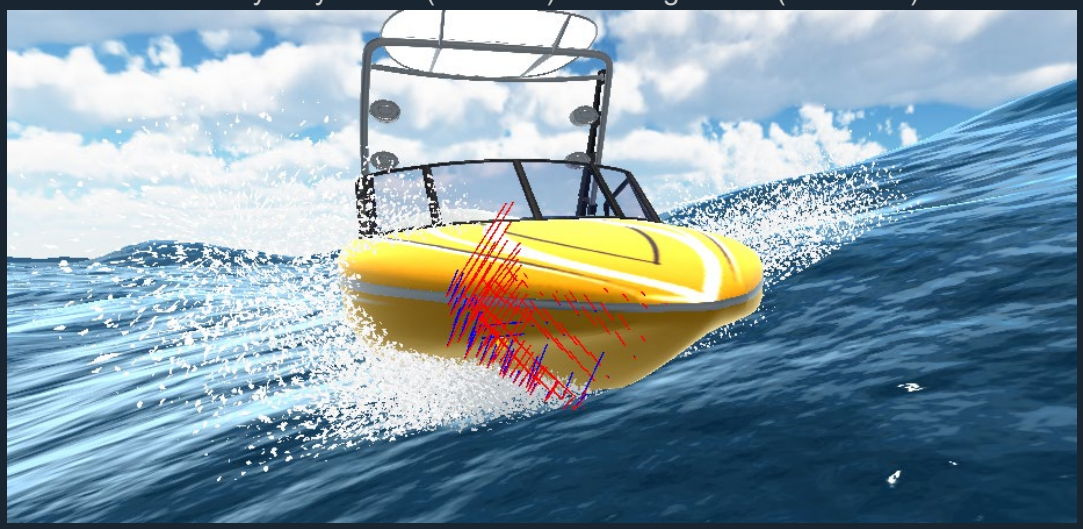
To optimize a simulation with many floating objects, you can switch on Flat Water while the object is not in the camera focus. This will reduce its CPU overhead by not computing the wave but still keep all the physics on. When turning it off again, it will start to float on the waves back again. Don’t do this if the waves are bigger than the object or it could be reactivated being under water. Future updates might include the number of wave systems to be computed to improve this optimization.

There are 2 kinds of particles generated, the sea foam burst when the impact speed exceeds a trigger, and the sea spray when the product of the impact speed and the wind exceeds a trigger. When the seam foam burst falls down on the water, it generates another kind of particle, the sea foam floats. The position of the floats precisely match the one from the burst that generated it even in the waves, but the size and the difference of texture might perturbate the precision of this effect. Mind that some VFX properties are set in the inspector of this component while most of the others are set in the VFX inspector. See the splashVFX references for more details on how to set up visual effects.

Buoyancy and particles generated by the Water Interaction component.

The audio is generated by modulating the pitch and volume through a hand-coded filter. Both depend on the total dynamic pressure acting on the mesh : the more dynamic pressure, the louder and lower the audio will be. Then the audio pitch and volume fade off at independent rates. Currently, for strong blasts on the water, the volume can become so high that the audio can become oversaturated or even inaudible. This bug is meant to be fixed in future updates. Storm Breakers provides a suitable audio clip for this effect (splash.wav).

You can turn on the gizmo mode to visualize the gravity center position (yellow wire sphere). In play mode you can also visualize the buoyancy forces (red lines) and drag forces (blue lines).



In this screenshot we can see how the virtual wake wave lower the forces on one side of the hull

Physics Properties	Function
Generate Water Forces	You can choose whether to generate forces. The rigidbody useGravity will be set to the same. If set to false the force will still be computed and drawn in the gizmos.
Use Jobs	Using jobs can increase FPS when the simulated mesh has a lot of triangles.
Flat Water	Tip this when the water is flat or when the object is away from focus, this will prevent the waves to be computed, saving some resources.

Orient Buoyancy Force Vertically	Tip this when there are almost no waves and to orient the buoyancy force vertically, this will prevent the object to move on its own because of the mesh dissymmetry making a net buoyancy force non vertical. Don't tip this for high speed watercraft.
generate Wind Force	Tip this if you want the mesh to drift with the wind, can add a little overhead. Use Foil components to generate proper sails force.
simulate Wake Wave	Tip this if you want that the system simulates the wake wave in the computation of the forces (no rendering of it), this helps cornering at high speed and leaning while surfing.
relative Density	The relative density of the rigid body (Ratio of the mass and the water mass for the same volume). Above 1 it sinks. Change this to make the object float at its waterline.
drag Coefficients	The drag coefficient in the local axis. https://en.wikipedia.org/wiki/Drag_coefficient
inertia Factors	The multipliers of the moment of inertia in the local axis. The rigid body component computes the inertia tensor based on the hypothesis the mesh is full, so for a hollow object in that direction you should increase the value by setting this multiplier superior to 1. https://en.wikipedia.org/wiki/List_of_moments_of_inertia
gravity Center Shift	The position of the gravity center in the local axis relative to the one automatically computed by the rigid body component. Use this to tweak the stability of the object in the water.

Particles Properties	**Function**
Generate Particles	You can choose whether to generate the particles or not.
Bottom Culling	Culls the particle and audio generation when the local triangle normal vertical component exceeds this value minus one in negative. Increase this value to prevent particles from being generated on the bottom of the object.
Topside Culling	Culls the particle and audio generation when the local triangle normal vertical component exceeds one minus this value. Increase this value to prevent particles from being generated on the top of the object.
Min Impact Speed	The minimal relative speed of the water onto the triangle to generate particles.
Burst Count	The number of burst particles generated per fixed delta time frames per triangle.
Spray Count Factor	The number of sea spray particles generated per impact velocity and wind.
Spray Max Count	The maximum number of particles generated for a triangle. This is to reduce overdraw when the object makes a strong hit.
Spray Lifetime Factor	The sea spray lifetime per impact speed and wind strength.

Audio properties	**Function**
Generate Audio	Whether to generate audio or not.
Audio Min Impact Speed	The minimal water impact speed at which a triangle adds dynamic pressure for the use of audio computation. Increase this value to avoid audio when the mesh is moving slowly.
Volume Factor	The audio volume per dynamic pressure acting on the mesh.
Max Volume	The maximum audio volume, can exceed 1 to get a bit of saturation and because the pitch effect reduce the audio intensity
Volume Fade Off Rate	The rate at which the audio loses volume after a blast.
Pitch Factor	The audio pitch per dynamic pressure acting on the mesh. You might need to adjust volume level when modifying this property.
Min Pitch	The minimal pitch applied. Too low pitch reduces the power of the noise, so you should actually increase it for large watercraft to get more audio power even though it leads to wrong pitch.

Pitch Fade Off Rate	The rate at which the audio got back to high pitch after a blast.

Debugging properties	Function
Maximal Pressure	The maximal pressure applied due to the speed. Since it evolves with the square of speed, this can reach higher values that the physic can handle for small objects moving fast. Lower it when the object bounces too fast off the water.
Pressure Excess Occurrences	Indicate the number of times the maximal dynamic pressure is reached, if it happens too often this means the physics might be impacted and you should consider raising the maximal pressure if there isn't trouble with it.
Maximal Wake Wave Height	The maximal additional depth when the wake wave is computed to prevent high values that wouldn't be manageable by the physics, should be in the vicinity of the object height.
Wake Wave Occurrences	Indicate the number of times the maximal wake wave height is reached, if it happens too often this means the physics might be impacted and you should consider raising the maximal height if there isn't trouble with it.
Draw Force	Whether to draw forces with lines in gizmos mode.
Vector Size	The size of the vector drawn in gizmos mode to visualize the forces.

API :

Method name	Return type	Description
UpdateVFXProperties	void	Is to be called when there are modifications on global sim data that might impact this visual effect attached (lighting, waves, wind).

Requirements :

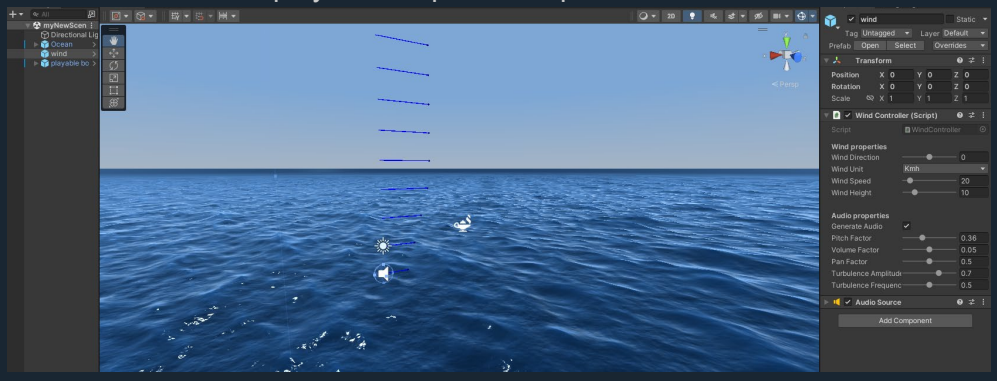
- A rigidbody must be attached to this gameobject.
- A convex Mesh Collider component must be attached to this gameobject. It should have a convex mesh set in the inspector or a Hull Generator component must be attached to this gameobject.
- The splashVFX.vfx visual effect component should be attached if you want to generate particles.
- A 3D audio source playing with splash.wav permanently should be attached if you want to generate audio effects.

WindController.cs

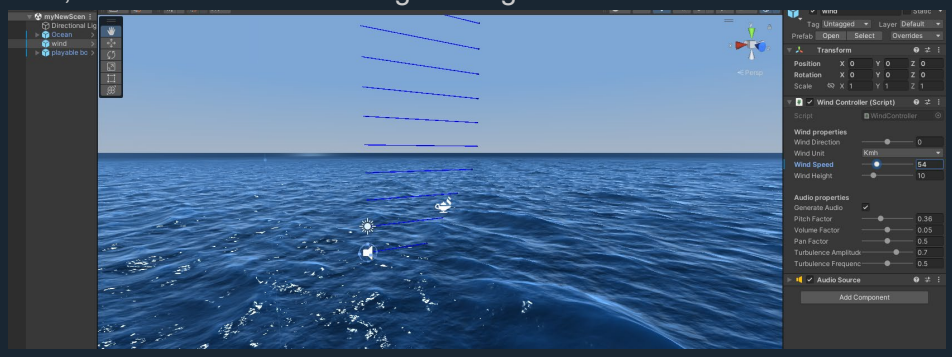
The Wind Controller component allows to set up the wind used in the simulation. It also generates the audio function of the position, orientation and even velocity of the camera in the scene.

The wind strength evolves with the height and the turbulences. At sea level the wind is half strength, it reaches full strength at the height you define; this helps to simulate the wind acceleration on the top of the large waves and the boundary layer effect of the atmosphere.

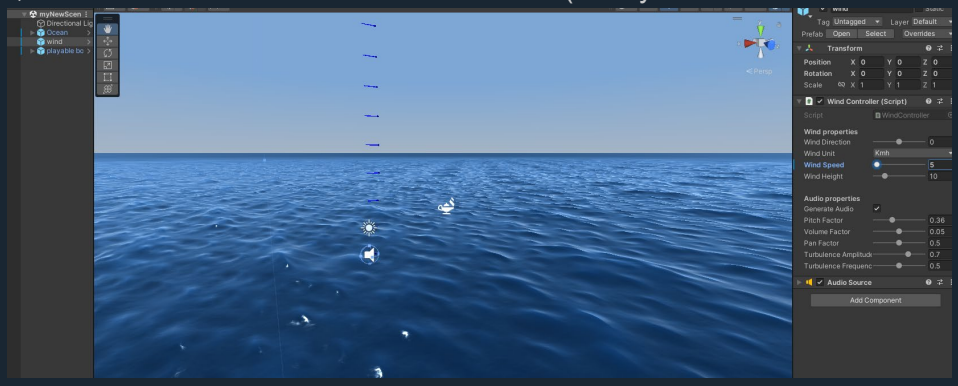
You can choose the unit you set in the inspector (in scripts you get m/s), turning on the gizmos allows you to visualize the wind vector and helps you set up the component.



This component also modifies the water surface roughness according to the wind strength and direction. With a stronger wind, the surface of the water gets rougher :



With the wind off, the surface of the water become smooth (then you should lower the wave intensity) :



In the current state of the package, the wind doesn't change the wave but only the ripples rendered with a normal map. A future update might merge the wind controller with the ocean controller to simplify the setup of the waves.

In the simulation, the wind affects mainly the visual effects and the water smoothness. It can also affect physics on some components if you ask so (Water Interaction, Foil).

If this component is not present in the scene, the default value of 20 km/h in the opposite world red axis will be used with a height of 10m.

Along with an audio source, this component generates the audio by modulating its volume, pitch and pan function of the camera position and orientation. The pan and volume are modulated according to the camera direction to simulate the wind blowing more or less in the virtuals ears Storm Breakers provides an audio clip suitable for this effect (wind.wav).

You can include the camera velocity in the wind computation. This can make some good effects at high speed. The velocity of the camera is computed using its parent's position when there is one so as to avoid the quick orbital movements to create over-fast wind.

Wind Properties	**Function**
Wind Direction	The direction of the wind in degree relative to the opposite world red axis.
Wind Unit	The unit of the wind strength set in the inspector (knot, km/h, mph, m/s)
Wind Speed	The strength of the wind in the wind unit.
Wind Height	The height at which the wind is full strength. At sea level the wind strength is halved.

Audio Properties	**Function**
Generate Audio	Whether to generate audio.
Include Camera Velocity	Whether the camera velocity adds a relative wind.
Pitch Factor	Defines the pitch of the audio source function of the wind strength at the camera's location.
Volume Factor	Defines the volume of the audio source function of the wind strength at the camera's location.
Pan Factor	Defines the pan variation when wind comes from the side of the camera.
Turbulence Amplitude	The percentage of the wind increase during turbulence.
Turbulence Frequency	The frequency of the turbulence per seconds. Affect only audio.

API : use Ocean.cs API.

Requirements :

- A camera tagged Main Camera must be in the scene.
- An object with the Ocean Controller component and rendering the ocean.mat material must be present in the scene
- An audio source should be attached to this gameobject if you want to generate audio effects.

Visual effect references

In this section are described the properties of the visual effect component. See the visual effect overview section of this document for more info on how they work.

boatControllerVFX.vfx

This VFX template is to be used to make the visual effects of a boat propeller.

Unlike the other VFX in Storm Breakers, this one does not scale automatically with the size of the object you are simulating, so many tweaking is to be expected here.

General property	Function
Bounds	The size of the cube that defines the bounding of the system. Should include the whole system.

Swirls Properties	Function
Swirl Texture	The texture used to render the swirl particles.
Swirl Mesh	The mesh used to render the swirl particles. Use a higher poly quad from the model folder when the size of the swirl exceeds the size of the ocean waves.
Swirl Size	The size of the particles when they are created.
Swirl Randomization	The scattering max distance in meters.
Swirl Growth Rate	The rate at which the swirl particles are growing.

Sea foam bursts properties	Function
Burst Texture	The texture used to render the sea foam burst particles.
Burst Alpha	The transparency of the burst particles.
Burst Size	The size of the burst particles.
Burst Growth Rate	The rate at which the burst particles are growing.
Burst Initial Alpha Clipping	The initial alpha clip of the burst particle. The more, the less pixels are drawn.
Burst Pixel Loss Rate	The rate at which the alpha clip increases function of the growth rate. This helps shrink particles in the air. This value is highly sensitive.
Burst Drag	How much burst particles are slowed down by the air or speed up by the wind.

Sea foam floats properties	Function
Float Texture	The texture used to render the floats particles.
Floats Mesh	The mesh used to render the floats particles. Use a higher poly quad from the model folder when the size of the floats exceeds the size of the ocean waves.
Floats Lifetime Factor	The lifetime of the floats relative to their initial transparency.
Float Transparency Factor	The transparency of the float relative to the burst's alpha.

The properties with an underscore before are automatically set by script.

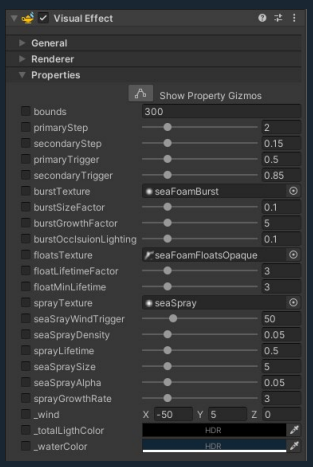
Requirements :

- The boat controller components must be attached to this gameobject.
- The Ocean Controller component must be present in the scene.
- The ambient light mode must be either set to color or gradient color.

oceanVFX.vfx

This VFX template is to be used to render the breaking waves and sea spray visual effects.

The properties are defined to scale with most of the sea state so you shouldn't need to modify any of them. If you want to modify the properties, see first the Visual effect overview section of this document for an extended explanation on how it works.



Grid Properties	Function
Bounds	The size of the box that defines the bounding of the effect. The larger it is, the more particles there will be and so an overhead is to be expected.
Primary Step	The length of the primary grid division. (The invisible particles in the primary grid read the ocean state at their position) If too high, some waves might not break, too small there will be a GPU overhead.
Secondary Step	The length of the secondary random grid relative to the wave size that triggers it. This defines the density of burst particles generated.
Primary Trigger	The water compression that triggers the creation of the secondary grid by the primary grid. If too low the primary grid might miss some breakers, if too high there will be too much of the secondary grid creation and an overhead is to be expected.
Secondary Trigger	The water compression that makes the particles of the secondary grid alive and render sea foam burst. The smaller the value, the longer the waves will break.

Sea foam bursts properties	Function
Burst Texture	The sea foam burst texture.
Burst Size Factor	The size of the burst particles relative to the wave size.
Burst Growth Factor	The growth rate of the burst particles relative to the wave size.
Burst Lifetime Factor	The estimated lifetime factor of the burst particle before dying and creating a float. Increasing this value dramatically increases the number of particles processed and thus shouldn't be changed much.
Burst Pixel Loss Rate	The rate at which the alpha clip of the texture increases during lifetime. Increasing this value makes the particles quickly get thinner.
Burst Occlusion Lighting	Define how much particles are darkened during their lifetime.

Sea foam floats properties	Function
Floats Texture	The sea foam floats texture.
Floats Size Factor	The size of the float relative to the bursts that generated it.
Float Lifetime Factor	The lifetime of the float particles relative to their size.
Float Min Lifetime	The minimum lifetime of float particles to avoid them disappearing too fast.

Sea spray properties	Function
Spray Texture	The sea spray texture.
Sea Spray Wind Trigger	The local wind in km/h that triggers the creation of sea spray particles. The higher this value is, the less sea spray particles there will be.
Sea Spray Density	Another way to tweak the density of the sea particles. Act as a probability of creation.
Spray Lifetime	The average lifetime of the sea spray particles.
Sea Spray Size	The average size of the sea spray particles.
Sea Spray Alpha	The transparency of the sea spray particles. Increasing this value can compensate for a lower number of sea spray particles and reduce overdraw, but making them more obvious at the same time.
Spray Growth Rate	The growth rate of the sea spray particles.

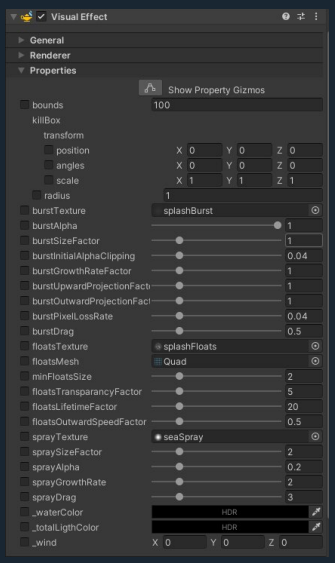
The properties with an underscore before are automatically set by script.

Requirements :

- The Ocean Controller component of the scene must be attached to this gameobject.
- The ambient light mode must be either set to color or gradient color.

waterInteractionVFX.vfx

This VFX template is to be used along with the Water Interaction or Sphere Water Interaction component. It generates sea spray and sea foam burst that leaves sea foam floats on the water. See the Visual effect overview section of this document for an extended explanation.



Kill box properties	Function
Bounds	The size of the cube that defines the bounding system. It should include the whole particle system : burst, spray and floats.
kill Box	The sphere in local space within which the particles die. Use this to cull particles inside a cockpit for example.

Sea foam burst properties	Function
Burst Texture	The texture used to render burst particles.
Burst Alpha	The transparency of the particles.
Burst Size Factor	The size of a burst particle relative to the size of the triangle that generated it.
Burst Initial Alpha Clipping	The initial alpha clip of the particle. Lower it to have more pixels drawn.
Burst Growth Rate Factor	Growth rate of the burst particles relative to the water impact speed.
Burst Upward Projection Factor	How much burst particles are projected upward relative to the impact speed.
Burst Outward Projection Factor	How much burst particles are projected outward the hull relative to the impact speed. Increasing this value helps preventing the particles from going through the hull.
Burst Pixel Loss Rate	The rate at which the alpha clip increases function of the growth rate. This helps shrink particles in the air. This value is highly sensitive.
Burst Drag	How much burst particles are slowed down by the air or speed up by the wind.

Sea foam floats properties	Function
Floats Texture	The texture used to render the floats particles.
Floats Mesh	The mesh used to render the floats particles. Change to a higher poly mesh (provided in Models folder) when the size of the floats exceeds the size of the wave.
Float Min Size	The minimal size of the floats when created, is used to cull their number.
Float Transparency Factor	The transparency of the float relative to the burst's alpha.
Floats Lifetime Factor	The lifetime of the floats relative to their initial transparency.
Floats Outward Speed Factor	Make the floats move outward as the hull passes, leaving a realistic wake.

Sea spray properties	Function
Spray Texture	The texture used to render sea spray
Spray Size Factor	The size of the spray particle relative to the triangle size.
Spray Alpha	Alpha multiplier of the sea spray particles.
Spray Growth Rate	The growth rate of the spray particles.
Spray Drag	How much burst particles are slowed down by the air or speed up by the wind.

The properties with an underscore before are automatically set by script.

Requirements :

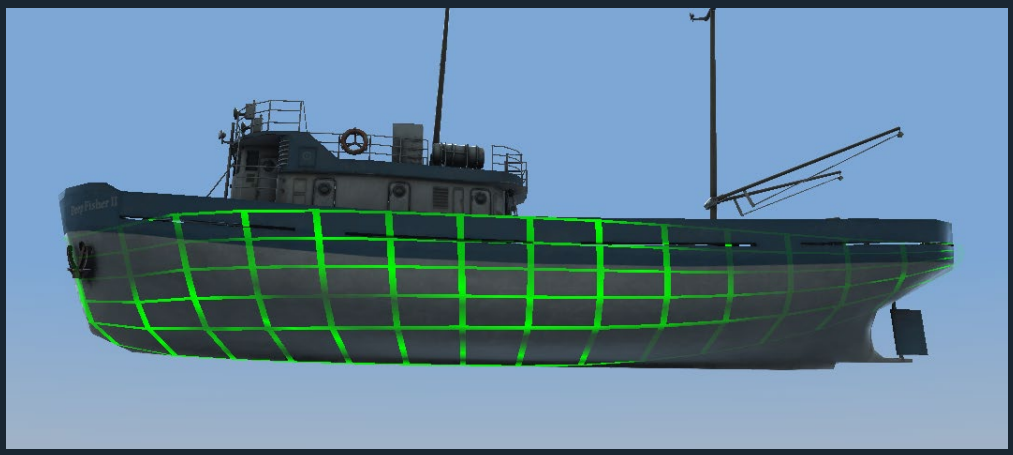
- Either the Water Interaction or Sphere Water interaction components must be attached to this gameobject.
- The Ocean Controller component must be present in the scene.
- The ambient light mode must be either set to color or gradient color.

Materials references

Storm Breakers provides some materials in the Materials folder, some are already included in the prefabs while the others are to be set manually. When doing so you might have to check first their reference here.

hullGeneratorHelper.mat

This material, used along with the Hull Generator component, acts as a helper when generating the hull.



We can spot the unlighted areas where the generated mesh is far to the geometry.

This material uses a custom shader to render a grid that allows to visualize the generated mesh, an element of the grid is two generated triangles. The brightness is a function of the scene depth so as to highlight the zone where the generated mesh is close to the geometry to help setting it up. The sensitivity of this effect is defined in the Hull Generator component.

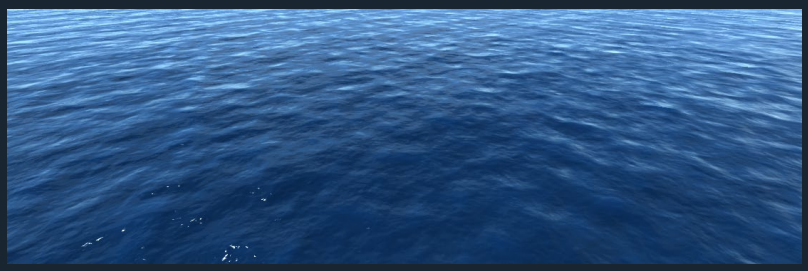
The inspector of this material provides only one property you can modify (the properties with an underscore before are set by script) and it's the maximal transparency : the higher the more highlighted the area far from a 3D object.

To work, this material should be set to a mesh renderer with the Hull Generator component attached to this game object. This component will then deactivate the renderer only if this material is detected so as not to pollute the game with helpers. If it's another material the renderer will be kept enabled because it's likely that it's not a helper (like the playable object from the scene template, the object is visible through this mesh renderer).

ocean.mat

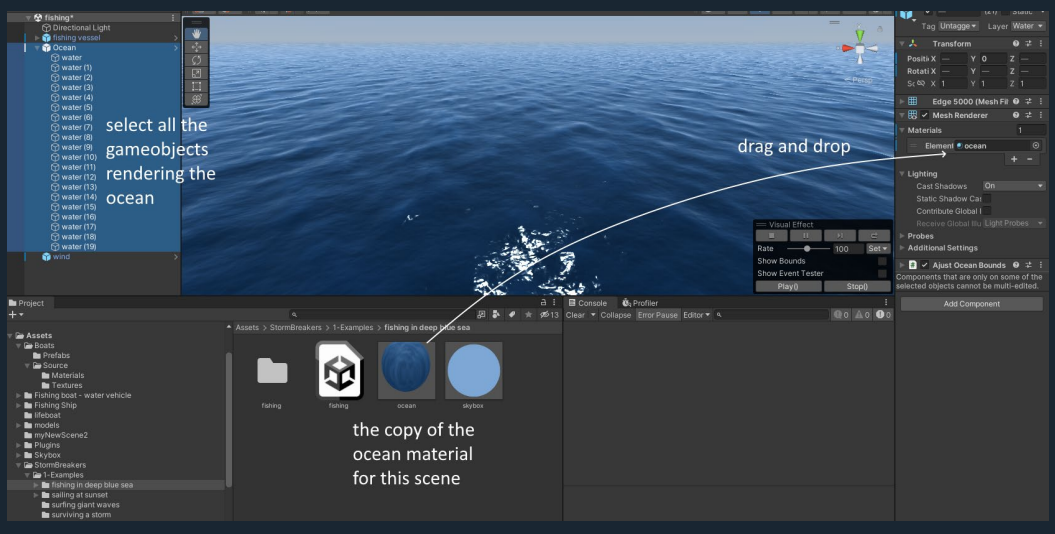
This material is the one used to render the ocean. It contains the vertex shader to render the wave and all the pixel shader for the water rendering. Some effects like the subsurface scattering and the wave normals are computed per vertex to optimize the draw time. This is why with an insufficient mesh density and small waves some artifacts appear, they come from the vertex interpolation being too rough.

Small wave rendered with a too low mesh density,
it makes flashes at runtime because of sudden vertex normal variations.



The same waves rendered by scaling down the ocean gameobject, making a higher mesh density.

The Ocean material should be cloned for every scene so you can make independent changes in the water rendering. This is automatically done when using the scene template, otherwise you may have to copy/paste the material and affect the new one to all the meshes of the ocean.



Note : In the project bowser, the sphere rendering the material can be outside the thumbnail because of the vertex displacement.

This material is rendered in the transparent queue to make the water transparent, so sdepth-dependent shaders don't work on the water. Still, when used the Underwater component will modify the render queue and the transparent behavior of this material so as to be able to read its depth when the camera is underwater.

The albedo of the water is a function of the orientation of the directional light to simulate the light rays penetrating more or less the water (subsurface scattering). So when the directional light is near the horizon or below, the water albedo becomes almost black and its color on the screen is mostly driven by the reflections at grazing angle. Further implementation of the subsurface scattering is done at the sharps crest by adding emission here in the shader. (The use of an emission instead of a change in the albedo is not physically accurate and can lead to glowing-like water, future updates might solve this.)

The reflections automatically come from the environment skybox. If it's not, go in *Lighting/Environment* and set the *Environment Reflection Sources* to "skybox". Here you can also change the *intensity Multiplier* to reduce the intensity of the water reflection if it's too bright.

The smoothness and normal map of the surface is driven by the wind. The more wind there is, the more intense the ripples normal maps will be and the smoothness also decreases to simulate more surface roughness.

Both faces of this material are rendered. Currently it's the same shading on both faces, the trick is to use reflection on a mirrored skybox to render underwater surfaces.

At the horizon, the normals of the wave are attenuated and another fast function (currently a simple Perlin noise) computed in the pixel stage is used to render smooth normal. You can change the setup of those fake waves in the inspector of this material.

Properties	**Function**
SubSurface Scattering	Defines how much the sharp crest of the waves glow to simulate subsurface scattering. Large values increase this effect and can lead to surrealistic rendering.
Minimal Transparency	The minimal alpha of the water. Increasing this value makes a neat separation of the water in front of partially immersed objects. When set to 1 there is no more transparency effect (but the material is still rendered in transparent queue)
Transparency Factor	Defines how much the alpha turns to opaque value with the increasing depth. A smaller value makes the water look clearer.
Ripples 1 Size	The scale of the first ripple normal map.
Ripples Speed	The speed at which the first ripple normal map moves.
Ripples Size	The scale of the second ripple normal map.
Ripples Speed	The speed at which the second ripple normal map moves.
Fake Horizon Waves Intensity	Changes the intensity of the fake waves at the horizon. Their intensity is automatically set to the maximum intensity of the wave systems, but sometimes this is not relevant so you can tweak it.
Fake Horizon Waves Density	Changes the density of the fake waves at the horizon. If you set the value to zero this makes no fake waves and a horizon uniformly tinted with the reflections from slope you provide in the previous property.

The properties with an underscore before are automatically set by script.

<u>Requirements :</u>

- The Ocean Controller component must be attached to a game object using this material in its renderer.

particlesMask.mat

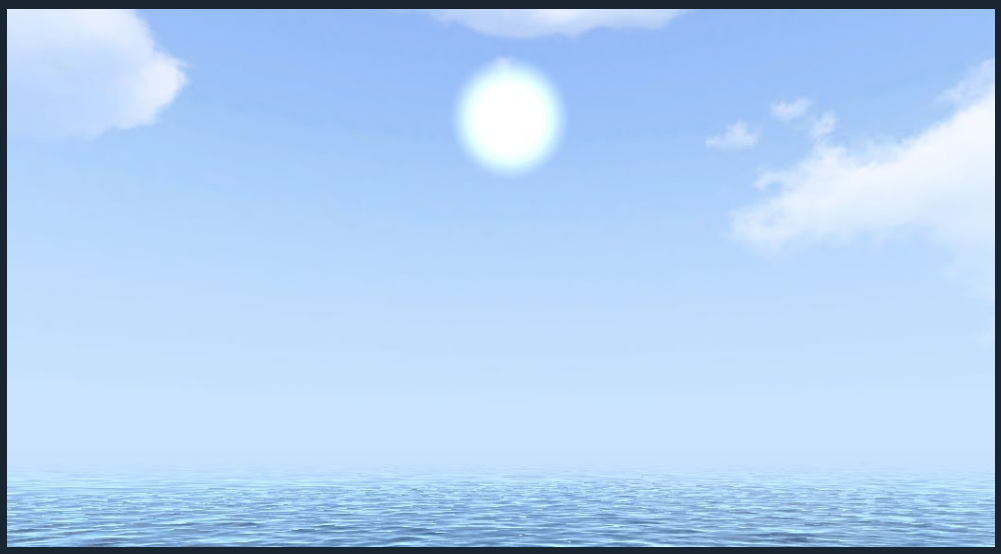
This material acts as a particle mask. You can use it on dedicated meshes to reduce splashes of particles to be visible in some areas where you want to see the water (otherwise use the water&ParticlesMask material).

Since this mask doesn't act as a volume, it is quite hard to clip the particles within a bounded area. One way to do this is to make a pyramidal mesh that joins the camera to the edge of the zone we want no particles within. (See particle clipping section of this document for more ways to prevent particles being visible in some areas.)

The pyramidal particles mask concept explained.

skybox.mat

Storm Breakers package provides a procedural skybox material that supports single color or cubemap texturing, fog and sun to help you render deep water scenes. You can use another skybox but mind that water reflection might be impacted.



A foggy scene using a skybox with a cubemap, fog and sun.

The fog color is the same as the one set in the lighting setting so it perfectly blends with the horizon. You can vary the height where the fog ends.

When using a cubemap texturing, the texture is mirrored from top to bottom. The reason is because many skybox cubemaps have a dark color on the bottom. With this trick it prevents the water from reflecting this wrong color (and also it helps render the back face of the water, but this is temporary).

The rendering of the sun is done using the main directional light as source in both direction and color. It is not currently affected by the fog but added on top of it.

Properties	Function
Texture Skybox	Whether to use a cubemap or a solid color.
Skybox Cubemap	The cubemap used to draw the sky when using it.
Sky Color	The color of the sky when using it.
Fog	Whether to draw the fog.
Fog Intensity	The height at which the fog stops. Near 0 there is almost no fog, at 1 the whole sky is fogged.
Sun	Whether to draw the sun.
Sun Radius	The radius of the sun.
Sun Intensity	The intensity of the color used to draw the sun. Increase it for a weak directional light.

underwater.mat

This material is to be used along with the Underwater component to make a fog when underwater and simulate the water transparency. This effect is currently pretty basic.

The color used for the underwater fog is directly the albedo of the water. There is currently no lighting on this effect. The density of the fog grows exponentially and you can vary it with the property called Density.

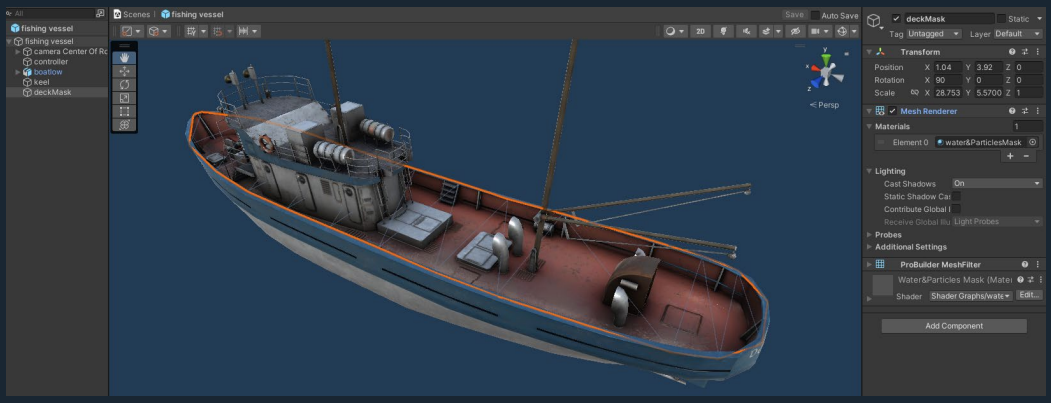
The shader doesn't segregate the pixels that are underwater to the ones that are not. The fog is globally affected as soon as the component considers the camera to be underwater. Still, the upper part of the skybox is cleared from this fog so as to try getting closer to a good segregation, but then we see emerged objects being fogged. Future updates will try to improve this effect based on user experiences, but a precise segregation of the pixels is still out of reach.

The underwater effect will try to be improved over time.

This material is to be set on a mesh set just in front of the camera's near clipping plane and using the Underwater component.

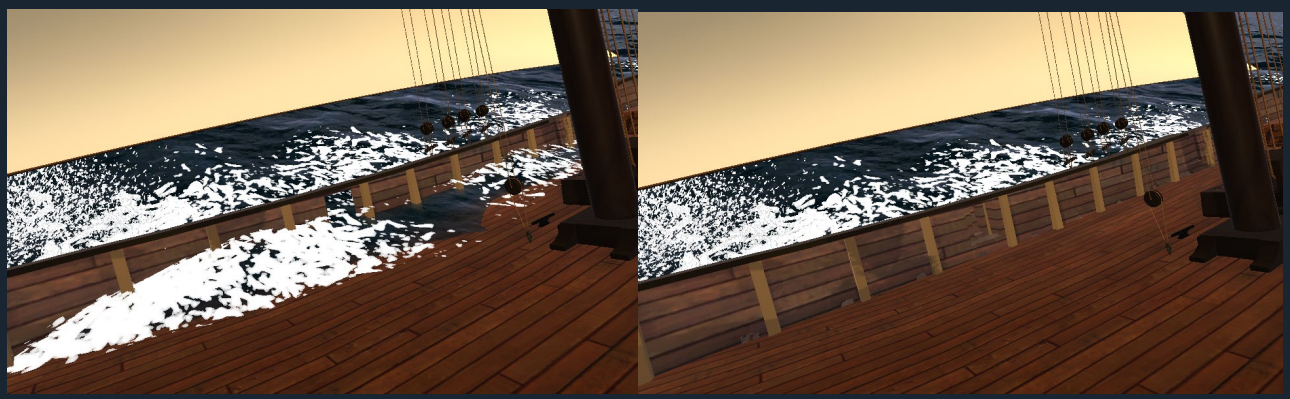
water&ParticlesMask.mat

This material acts as a mask to clip everything drawn below afterwhile, that is all the transparent objects including the water and particles. This is very handy to clip the water within an open boat or to reduce particles and water going through the deck by making a deck mask. To use it, simply set this material to a dedicated geometry. As a general rule it is good to build for any boat a so-called deck mask enclosing the solid railings.



A deck mask built with Probuilder.

Technically it is a transparent shader graph writing its depth on the buffer and rendered before the water. All geometry drawn afterwhile would only draw if they are on top of it and not behind. The trouble is when you want to clip a not fully watertight volume, for example between the railings of a boat with gunports. The way it is currently implemented, the shader will prevent everything from being drawn in the gunport, leaving an unsettled color. This will try to be fixed in future updates by writing a more advanced shader that conditionally writes its depth function of the scene depth.



Without the deck mask (on the right) the water goes through the railing.
With the deck mask on (on the right). The wrong rendering of the gunport is currently not fixed.