USER MANUAL

Everything, explained.

From a first ocean in two clicks to breaking surf on your own coastline.

- Endless ocean surface
- Dynamic and rogue waves
- Breaking waves and surf
- Coastlines, shoaling, shore wash
- Lakes, rivers, water zones
- Buoyancy and wakes
- Underwater and caustics
- Foam, white water, spray
- Day and night cycle
- Seven ready-made profiles

Version 1.28 · Unity 6000.0+ · Universal Render Pipeline 17
Redicion Studio · Contact@RedicionStudio.com

About this manual

OceanCore is a complete ocean for Unity 6 and the Universal Render Pipeline: an endless animated sea with dynamic waves, breaking crests, coastlines, lakes and rivers, buoyancy, underwater rendering and a day/night cycle.

This manual is written for the person using the asset. It starts with a five minute quick start, then takes each feature in turn and says what it is for, which setting controls it, and what to do when it does not look right. You do not need to read it in order.

Everything here describes OceanCore 1.28. Where a setting is named in **bold** it is a field you will find in the Unity Inspector; where it is in `code` it is something you type.

Just want water?	Chapter 3, Quick start. Two menu items and you are done.
Want it to look different?	Chapter 6, Ocean profiles. Nearly every setting lives there.
Building a coastline?	Chapters 14 to 16.
Floating something?	Chapter 21, Buoyancy.
Something is wrong?	Chapter 33, Troubleshooting.
Build takes too long?	Chapter 29, Shader variants.

Conventions

Tools → OceanCore → Welcome means a Unity menu path. **Wave Detail Fade** means a field in the Inspector. A grey box is something you can paste into a C# script. A blue box is worth knowing but not required reading.

> Throughout this manual, **the ocean** means one Ocean Core Manager component. A scene can have several — a sea and three lakes is a normal setup — and each one is independent.

Contents

Part 1

Getting started

OceanCore — Dynamic Ocean & Wave Simulation for Unity 6 URP

CHAPTER 1

Welcome

OceanCore is a complete ocean for Unity 6 and the Universal Render Pipeline. You drop one object into a scene and you have an endless, animated sea that boats float on, that breaks against your islands, that you can swim under and that looks different at noon than it does at midnight.

This manual is written for the person using it, not for the person who wrote it. Every chapter says what a feature is for, where the setting lives, and what to do when it does not look right. You do not need to read it in order — the Quick start is two pages, and after that you can jump to whatever you are building.

What you get

An endless sea	No visible edge, ever. The surface is rebuilt around the camera every frame.
Waves that are really there	Boats, buoyancy and gameplay read the same water the shader draws.
Dynamic waves	Single large waves that form, travel, grow, break and die — spawn them from script or let them happen.
Rogue waves	The rare big one, with its own probability, size range and cooldown.
Breaking waves	The ocean surface itself folds over into a barrel you can fly a camera through.
Coastlines	Waves shoal, steepen and collapse on your terrain. The water runs up the beach and drains back.
Lakes and rivers	One water surface at many heights, with a real current that carries what floats on it.
Whirlpools	Funnels cut into the real surface, with a current that drags ships round.
Underwater	Fog, absorption, caustics and a proper water line when the camera is half submerged.
Wetness	Rocks, walls and hulls are drawn wet where the water has reached them.
Buoyancy	Multi-point floating with pitch, roll and heave, plus wakes, splashes and ripples.
Day and night	A clock that turns your sun and changes how the water looks as it goes.

What it does not do

OceanCore is water, not a game. It contains no ship system, no weather manager and no gameplay. There is an example boat, and it exists to show how the APIs are used — it is a starting point, not a product feature.

HDRP and the Built-in render pipeline are not supported in this version.

CHAPTER 2

Requirements

Unity	6000.0 or newer
Render pipeline	Universal Render Pipeline 17.x
Shader model	4.5 — the waves are read from a structured buffer
Platforms	PC, consoles, and any mobile GPU that supports shader model 4.5
Textures to import	None. Every pattern is generated in the shader.

Installation

1. Import the OceanCore package, or copy the `Assets/OceanCore` folder into your project.
2. Open **Tools** → **OceanCore** → **Welcome**. This window opens by itself the first time you start the project.
3. At the top of that window is **Project setup**. It checks five things and tells you which are missing.

The setup window never creates anything you did not ask for. It does not add profiles, materials or scenes — it only changes render pipeline settings, and only when you press the button.

What the setup checks

Check	Why
Universal Render Pipeline	OceanCore is URP only. Without a URP asset in Graphics Settings there is nothing to configure.
Opaque texture	The water reads what is behind it for refraction and for absorption. Without it the water renders black.
Depth texture	Used for water depth, shoreline blending and the underwater pass.
Renderer features	**OceanCore Underwater** and **OceanCore Wetness** on your Universal Renderer.
Shaders included in builds	OceanCore creates some materials in code, so those shaders have to be in *Always Included Shaders* or they are stripped from the player.

The **Enable required settings** button is only clickable while something is actually missing. When everything is in place it reads *Project is ready* and does nothing.

> If you ever see the water render **black**, the opaque texture is off. That is the single most common setup mistake, and it is one button to fix.

CHAPTER 3

Quick start

Five minutes, from an empty scene to an ocean.

1. **Tools → OceanCore → Welcome → Enable required settings.** Do this once per project.
2. **GameObject → OceanCore → Ocean.** That is the whole thing: an animated, endless, lit ocean.
3. Move the new object up or down. **Its Y position is the sea level.**
4. In the Inspector, drop one of the shipped profiles into **Profile** to change the sea — or leave it empty and you get Open Ocean, which is the default.

To see everything at once, press **Tools → OceanCore → Create Demo Scene**. It builds an island, a beach, a lake, a boat and a camera, and it does not touch your own scenes.

Making something float

1. Give an object a `Rigidbody` and a collider.
2. Add **OceanCore → Ocean Buoyancy**.

That is all. Sample points are worked out from the colliders, the object floats with pitch, roll and heave, it leaves a wake when it moves, and it splashes when it lands. Nothing has to be registered with anything.

Making waves break on your island

1. Add **OceanCore → Ocean Wave Obstacle** to the island, the rock or the terrain.

That is all as well. There is nothing to bake: OceanCore scans the sea bed under the camera while the game runs, and the component tells it that this object counts. Within a second or two the waves die against the shore and foam appears along the water line.

CHAPTER 4

The demo scene

Tools → OceanCore → Create Demo Scene builds a scene with an island, a beach, a crater lake, floating crates, a boat and a free camera. It is the fastest way to see what every setting does, and it is safe — it creates its own scene and leaves yours alone.

Controls

Key	Action
`1` **…** `7`	blend into one of the shipped ocean profiles
`Space`	spawn a large dynamic wave upwind of you
`R`	force a rogue wave
`C`	remove every dynamic wave
`T`	freeze or restart the day/night clock
`[` **and** `]`	move the time of day by an hour
`H`	hide and show the help panel
`W A S D` **+ right mouse**	fly the camera, or drive the boat when it is selected

The on-screen panel

The demo draws its help and its live state on a real canvas in the scene, not as a debug overlay. Add it to any scene with **Tools → OceanCore → Create Demo HUD**: it builds a canvas with the OceanCore header, the key list and a status block showing the profile, the time of day, the wind, the wave height, how many dynamic waves are alive and whether the camera is under water.

It is ordinary Unity UI. Select it, restyle it, move it to another corner, or make it a prefab. If there is no HUD in the scene, the demo controller falls back to a plain overlay instead so you are never left with nothing.

CHAPTER 5

How the pieces fit together

You will normally only ever touch a handful of components. This is the whole list.

Component	Add it to	What it does
Ocean Core Manager	its own object	One body of water. The sea, a lake, a pool.
Ocean Buoyancy	anything with a Rigidbody	Floats it, wakes it, splashes it.
Ocean Surface Probe	anything	Puts a transform on the water with no physics at all.
Ocean Wave Obstacle	an island, rock, pier, terrain or hull	Tells the water that this is there.
Ocean Wake Emitter	a moving object with no OceanCore buoyancy	Gives it a wake anyway.
Water Exclusion Volume	a box inside a cabin	Cuts the water out of an interior.
Water Zone	an empty object	Changes what the water does in one area.
Ocean Vortex	an empty object	A whirlpool.
Ocean Lake Area	an empty object	Water standing at its own height on land.
Ocean River	an empty object	Water running downhill along a course you draw.
Ocean Water Level Zone	an empty object	The sea, held at a different height.
Day Night Cycle	an empty object	The clock.
Water Audio	the camera	Swaps the ambience above and below the surface.

Everything else in OceanCore is a **setting on an Ocean Profile**, which is the next chapter and by far the most important idea in the asset.

The menus

Menu	What is there
Tools → OceanCore → Welcome	Project setup, the profiles in your project, shader variants, support
Tools → OceanCore → Debugger	Live statistics, the dynamic wave list, the sea bed scan, the scene overlays
Tools → OceanCore → Create Demo Scene	Builds the demo
Tools → OceanCore → Create Demo HUD	Adds the canvas HUD to the open scene
GameObject → OceanCore → Ocean	A new water body
GameObject → OceanCore → Day Night Cycle	The clock
GameObject → OceanCore → Water → …	Lake, River, Water Level Zone, Water Exclusion Area
Assets → Create → OceanCore → Ocean Profile	A new profile
Assets → Create → OceanCore → Shader Stripping Settings	Build options, see chapter 29

Part 2

The sea

OceanCore — Dynamic Ocean & Wave Simulation for Unity 6 URP

CHAPTER 6

Ocean profiles

An **Ocean Profile** is one asset holding every parameter of a sea: the wind, the waves, the colours, the foam, the reflections, the spray, the underwater look. Assign one to a water body and that is what the water is.

This matters more than it sounds. Nearly every setting in OceanCore lives on a profile rather than on the manager, and that is deliberate: it means the whole sea can be swapped, saved, shared between scenes, and — most importantly — **blended**. A storm rolling in is one line of code, because every number in a profile can be interpolated towards every number in another.

The seven shipped profiles

Profile	What it is
Calm Sea	Almost flat water with long, slow swells. Harbours and cutscenes.
Tropical Water	Bright turquoise shallow water with small waves. Islands and reefs.
Open Ocean	The default. Long swells with occasional large dynamic waves.
Rough Sea	Open ocean with towering rogue waves and coarse foam. The demo's heavy sea.
Sunset	Open ocean tuned for warm, low-angle light.
Night Ocean	Dark water with strong specular highlights.
Crater Lake	Sheltered fresh water. A bounded water body with no dynamic waves.

They live under `Assets/OceanCore/Generated/Profiles`. Open Ocean is also what a water body uses when you assign no profile at all, and what a brand new profile asset starts from.

Editing one

Select a profile and the inspector shows nineteen collapsible groups. There are several hundred values in there, so there is a **search box** at the top: type `foam` and only the groups with something to do with foam are left, already open. **All** and **None** expand and collapse everything.

Changes are applied to every live ocean using that profile the moment you make them. You do not have to press play, and you do not have to press anything at all.

At the bottom, **Presets** replaces the whole profile with one of the seven shipped ones, and **Reset to OceanCore defaults** puts it back to Open Ocean.

> **A preset is the shipped profile, exactly.** The preset list is generated from the profile assets, field by field, so loading "Calm Sea" onto a profile gives you precisely the Calm Sea that ships. They cannot drift apart.

Switching profiles at runtime

Instantly:

```
ocean.SetProfile(stormProfile);
```

Or blended, which is what you almost always want:

```
ocean.LerpToProfile(stormProfile, 25f);   // over 25 seconds
```

Every group interpolates — wind, wave heights, colours, foam persistence, rogue wave probability, reflection strength, underwater fog. There is nothing to sequence by hand.

You can also blend towards a set of values that is not saved anywhere:

```
OceanProfileData target = ocean.Current;
target.wind.speed = 26f;
target.spectrum.amplitudeScale = 2.4f;
ocean.LerpToState(target, 12f);
```

And you can override just the colours, without touching the profile asset at all:

```
OceanColorSettings colors = ocean.BaseWaterColors;
colors.deepColor = nightDeep;
ocean.SetWaterColors(colors);
```

> Use `BaseWaterColors` rather than `Current.color` when you are writing colours every frame. `Current.color` is what is on screen, which may already include your own tint — reading it back and tinting it again drives the water to black in about a second. `BaseWaterColors` is what the profile says, before any override.

CHAPTER 7

How the surface is built

You do not have to know any of this to use OceanCore. It is here because two of these decisions have settings attached, and because it explains why the water behaves the way it does.

The endless surface

The visible ocean is a **clipmap**: a dense square grid around the viewer, surrounded by rings that each cover four times the area with the same number of vertices. The whole thing is re-centred on the camera every frame and snapped to whole cells, so the player can never reach an edge and the surface never appears to slide underneath itself.

Because it is always centred on the viewer, a vertex is always the same distance from the middle. That lets the level-of-detail morphing be baked into the mesh when it is built, which removes the cracks between rings completely and costs nothing while the game runs.

Why the water does not crawl

A grid can only show a wave it samples often enough. Once a wave gets close to the vertex spacing, the mesh stops reproducing the wave and starts reproducing whatever the vertices happen to land on — and since the grid slides under a moving camera, that pattern moves with it. It reads as the water crawling, wobbling or shimmering whenever you move.

OceanCore fades those waves out of the geometry instead. Every vertex works out its own local spacing and a wave is faded out below roughly two samples per wavelength. The detail at that scale is carried by procedural normals in the fragment shader instead, where it cannot alias.

Wave Detail Fade on the manager controls it. `0` disables it and the crawling comes back; `1` is the recommended setting; higher fades more and gives a calmer surface. Foam is deliberately left out of the fade, so whitecaps stay visible all the way to the horizon.

> **If the surface shimmers when you move the camera, this is the setting.** Put it back to 1.

Why the highlights drift instead of jumping

The fine surface detail carries the dark patches and the glints, so how it moves is most of what the eye reads as "this is alive" or "something is wrong". Two things make it behave. The scroll offset is accumulated frame by frame rather than computed from a clock, so a gust changes the *speed* of the drift and never its position. And the detail is anchored to the parcel of water rather than to the world, so the pattern travels with the wave orbit instead of sliding across the surface underneath it.

The waves themselves

The base sea is a sum of Gerstner waves spread over four bands:

Band	Typical wavelength	What it is
Ripples	1 – 4 m	close-range surface detail
Small waves	5 – 15 m	wind chop
Medium waves	20 – 50 m	the bulk of the visible motion
Swells	80 – 200 m	long, slow, mostly aligned with the wind

Each band has a wave count, an amplitude, a wavelength, how far its waves may deviate from the wind, how sharp the crests are and a speed multiplier. Above them sit four global scales: **Amplitude Scale**, **Choppiness Scale**, **Time Scale** and **Wind Energy Influence**. In practice you will adjust the global scales far more often than the bands.

Every wave gets its direction, length, height and phase from the profile's **Spectrum Seed**, so the same profile always makes the same sea. Change the seed for a different sea of the same character.

> **Time Scale at 0 freezes the ocean.** Everything derived from the waves freezes with it, including the shore wash. If the beach foam has stopped moving, check this first.

The water a boat feels is the water you see

The wave list is uploaded to the GPU once per frame, and the CPU surface queries read exactly the same array with the same formula. A boat therefore rides the wave the player is actually looking at. There is one deliberate exception, inside a breaking wave, and it is explained in chapter 9.

CHAPTER 8

Dynamic waves

Dynamic waves are what OceanCore is built around. They exist **in addition to** the base spectrum: one localised packet of water that reads as a single large wave with a defined crest, rather than as an endless train.

A dynamic wave forms, grows, travels, reacts to shallow water, breaks, loses energy and dissipates. You can let the profile spawn them automatically, or make them yourself.

Letting them happen

In the profile's **Dynamic Waves** group, switch **Auto Spawn** on and the sea produces them by itself. The settings that matter:

Setting	What it does
Enabled	The whole system. Off, no dynamic waves exist at all.
Auto Spawn / Auto Spawn Interval	Whether the sea makes its own, and how often.
Auto Spawn Distance	How far from the viewer they form.
Height Range	Crest height, in metres, picked between these two.
Wavelength / Width / Crest Length Range	The size of the packet.
Speed Range	How fast they travel.
Steepness	How sharp the crest is.
Formation Time / Lifetime / Dissipation Time	How long it takes to grow, how long it holds, how long it takes to die.
Breaking Strength / Foam Strength	How white it gets.
Random Variation	How much every wave differs from the last.
Direction Spread	How far a wave may run off the wind, in degrees. Default 25°.

Making one yourself

```
var settings = new OceanWaveSpawnSettings
{
    position         = new Vector3(0f, 0f, -400f),
    direction        = new Vector2(0f, 1f),
    height           = 8f,
    wavelength       = 120f,
    halfWidth        = 90f,
    crestLength      = 160f,
    speed            = 12f,
    steepness        = 0.7f,
    formationTime    = 10f,
    lifetime         = 30f,
    dissipationTime  = 12f,
    breakingStrength = 1.2f,
    foamStrength     = 1.4f,
    randomVariation  = 0.2f
};

OceanWaveHandle wave = ocean.SpawnWave(settings);
```

Or the short form, which takes everything else from the profile:

```
ocean.SpawnWave(position, direction, height: 6f);
```

A handle stays safe to use after the wave has died:

```
if (ocean.DynamicWaves.IsAlive(wave))
    ocean.CancelWave(wave);      // fades it out rather than deleting it
```

Waves are pooled. The budget comes from the quality tier — 8 on Low up to 64 on Ultra — and `SpawnWave` returns `OceanWaveHandle.Invalid` when it is full. It never allocates and never grows the pool.

Two events are there for audio and effects:

```
ocean.DynamicWaves.WaveSpawned         += (pos, height, isRogue) => { };
ocean.DynamicWaves.WaveStartedBreaking += (pos, height) => { };
```

Rogue waves

Rogue waves use the same pool with their own size range, probability and cooldown, and they are **on by default**. Their size is biased towards the smaller end of the range so that the really big ones stay rare, and wavelength, width and crest length all scale with height, so a small one is a compact swell rather than a shrunken wall.

Setting	What it does
Probability Per Minute	How often one is attempted.
Min / Max Height	The range it is picked from.
Spawn Distance	How far upwind it appears.
Direction Variation	How far it may run off the wind, in degrees. Default 18°.
Formation Time / Lifetime	Its own life cycle, independent of ordinary dynamic waves.
Cooldown	The shortest time between two of them. Lower it to let them overlap.

Set **Enabled** off in the Rogue Waves group to switch them off completely.

Which way the big waves run

Dynamic waves, rogue waves and the breaking waves that come out of them all run **with the sea**.

That is not a style choice, it is what those waves are. A rogue wave is not a separate event arriving from somewhere else — it is the wave groups already marching downwind briefly drifting into phase with one another, the same sea adding up. So it travels on the sea's heading, and the only question is how far it may wander off it: **Direction Spread** for dynamic waves, **Direction Variation** for rogue waves.

A real swell is *narrower* than the wind sea that made it, not wider. Past about 40° the sea stops reading as one system and starts reading as two crossing each other, which the open ocean does roughly never. If your big waves look like they belong to different weather, these are the two numbers to bring down.

> **For scripts:** `ocean.Wind.Direction` is the wind bearing. `ocean.Wind.WaveTravel` is the direction the water actually moves, and it is the *opposite* — a Gerstner crest travels against its own direction vector as its phase advances. Anything you spawn that should belong to the sea around it wants `WaveTravel`; `SpawnWave` already falls back to it.

CHAPTER 9

Breaking waves

A large wave stands up, its face goes steep, the crest reaches forward and rolls over into an overhang, the overhang falls onto the water in front of it, and what is left spreads out as white water and fades.

Switch it on with **Enable Breaking Waves** in the profile's **Breaking Waves** group.

The wave itself deforms. There is no second mesh and nothing laid on top: a band of the ocean surface around the crest is rotated about an axis running along the crest line, so the wave is one connected body of water with a fold in it. You can fly a camera into the barrel, and the water inside it is the same water as everywhere else — same colour, same lighting, same foam.

It has nothing to do with the shore

Breaking here is decided by the wave's own shape and by nothing else. It does not read the sea bed, it does not need one in the scene, and it is completely separate from **Shore Shoaling** (chapter 16), which is a different feature for waves running into shallow water. A rogue wave in the middle of the deep ocean breaks if it is big enough and steep enough. So does a wave you spawn yourself.

Which waves break

Three measures of the wave, each a floor it has to clear:

Setting	What it means
Minimum Height	Crest-to-trough height in metres below which a wave never breaks, however sharp.
Steepness Threshold	Amplitude over wavelength. Around 0.44 real water cannot hold together at all, so values near that reserve breaking for genuinely violent seas.
Curvature Threshold	How pinched the packet is along its travel axis. 1 means a group one wavelength long that comes to a point; 0.15 means one that runs for about seven and rolls rather than pitches.
Breaking Threshold	The overall bar the three combined have to clear. Raise it to make breaking rarer without touching the individual floors.

Dynamic Waves Can Break and **Rogue Waves Can Break** let you have one without the other. **Breaking Point** decides *when* in a wave's life it goes over: 0.45 means a wave that has grown, held for a while and is about to start fading.

The shape of the break

Setting	What it means
Curl Strength	How far the crest band turns. This is the setting that folds the surface: past about 1.6 radians the wave is genuinely overhanging, below it the crest only leans.
Curl Radius	How far below the crest the axis it turns about sits. **Short is what makes a lip** — only the crest swings on a short arm. A long one carries the whole upper mass round and the wave balloons into a lump instead of throwing over.
Curl Start Height	How much of the wave comes with it. High keeps the roll to the very crest for a thin fast lip; low brings more of the face round for a heavier, slower one.
Lip Forward Distance	A little extra forward throw. A little goes a long way — a large value slides the whole band ahead and flattens the wave into a ramp.
Lip Drop Distance	A little extra downward pull, so the lip plunges rather than rolling in place.
Overhang Strength	How far the surface shears forward under the crest, steepening the face the lip comes over.
Randomness	How much every break differs — when it starts, how hard it turns, how far it peels.
Collapse Decay	Seconds the spent wave takes to sink away. Default 7.

Everything in metres scales with the wave's own height, so one set of numbers reads correctly on a four metre wave and a twenty metre one.

The timing

Cresting → Pitching → Curling → Breaking → Whitewater → Dissolving, each with its own duration. **Lip Speed** scales the whole sequence; **Collapse Speed** only the part after the lip lands.

The break *peels*: one end of the crest goes over before the other, which is what makes it read as a wave breaking along its length rather than a ridge collapsing at once.

The fall is not on a clock. Every tick the system measures the water under the turning crest and sizes the fall so the lip crosses it partway through the breaking phase — so you see the lip fall, and then strike. That contact, not a timer, starts the collapse. **Impact Height Offset** is how close it has to get before it counts. Because the break peels, the impact runs *along* the crest, with a spray burst and a foam scar at each point of contact.

A wave breaks once

The moment the lip strikes, the wave is spent. It never breaks again — the energy that held the crest up has gone into the white water, and a wave that kept rearing up every few seconds reads as a glitch rather than as a sea. Over **Collapse Decay** seconds it loses height smoothly and is retired.

The collapse reaches everything at once: the drawn surface, buoyancy, the foam, the spray and the shore wash all see one shrinking wave. A boat riding it is set down gently as the water goes out from under it.

Ships in a breaking wave

By default a breaking wave runs straight through a ship. **Breaking Waves Interactable** connects them.

Switch it on and every object with an **Ocean Buoyancy** that has **Displace Water** ticked becomes something the break has to get past. Where a hull is standing, the wave face, the curl and the lip stop turning over. The water either side keeps rolling, so the crest **comes apart** and passes the ship on both sides rather than being dented with the fold still in it.

Setting	What it means
Breaking Waves Interactable	Off by default. The master switch.
Interaction Reach	How far past the hull's own half beam the wave gives way, as a multiple of it. 1.1 is tight; 3 or more opens a wide channel.
Interaction Depth	How deeply the water is pressed in under the hull, in metres.
Interaction Shove	How far the water is pushed out and away. This is the part that makes the wave *part* rather than only dent.

Two things are worth knowing. **It is what you see, not what you feel** — the dent is a displacement the boat creates, so it is deliberately kept out of the buoyancy query; otherwise the hull would press the water down, read the lowered water, sink and press harder. And **the gap follows the lip, not the hull** — a curling crest is thrown metres from where it started, so the split is worked out from where that water is going to land.

Sixteen hulls are considered at once.

What it costs

Setting	What it means
Max Active Breaking Waves	How many break at once. Each is a rotation per ocean vertex in range, so this is the dial that decides the cost. Default 4, hard ceiling 16.
Max Distance	Beyond this a wave does not break at all.
Crest Segments	How finely foam, white water and spray follow each crest.
Lod Update Rate	How often per second the state is re-evaluated. A break takes seconds, so 30 is plenty.

When nothing is breaking the whole feature costs one comparison per vertex.

When nothing breaks

Switch on **Show Breaking Debug** and each detected crest is drawn in the Scene view. A **white cube** means that crest has actually folded — if a crest never turns white it is leaning, not curling, and **Curl Strength** is the setting. A **yellow ring** marks a point of impact and should travel along the crest as the break peels. If nothing is drawn at all, no wave is clearing the thresholds.

CHAPTER 10

Foam, white water and spray

Where foam comes from

Three places, and they stack:

1. **Procedural crest foam**, computed in the shader from how pinched a wave crest is. Free, and always available.
2. **Shore wash foam**, the sheet running up a beach and draining back. Chapter 15.
3. **The persistent foam buffer**, a texture that follows the camera, decays over time and drifts with the wind. Breaking waves, boat wakes, propeller foam and splashes are all deposited here, which is what gives a breaking crest a trail that stays behind after the wave has gone.

Foam Persistence is how long foam survives; **Foam Advection** is how hard the wind drags it. The buffer's resolution and the area it covers come from the quality tier.

Foam on a breaking crest

White water on a dynamic or rogue wave sits on the wave's own crest ridges and travels with them. It is not a patch laid over the wave — the pattern is generated in the wave's moving frame, so it is glued to the crest, and it is torn into wisps and streaks with open water between them rather than filling a shape.

Setting	What it means
Crest Foam Threshold	How pinched a crest has to be before it goes white.
Crest Foam Strength	How much white.
Crest Foam Breakup	0 leaves a smooth closed band along the crest; 1 tears it into wisps.
Crest Foam Scale	The size of those torn structures, relative to the wave's own wavelength.

Scaling with the wavelength rather than with a fixed size is what lets one setting read correctly on a five metre wave and on a hundred metre roller.

The tearing is built per vertex, so how fine it can be is decided by the mesh as well as by the setting. With distance the structures are made broader and eventually close back into a plain band, with the coverage held level so a distant crest is neither bare nor brighter than a near one. Raise **Clipmap Resolution** in the quality tier to keep finer tearing further out.

Spray off a breaking wave

Every breaking dynamic wave throws a curtain of spray along its crest, and everything about it scales with the wave. The water is dense just above the crest and thins into sea mist further up, and the wind carries it sideways.

Setting	What it means
Amount	How much water is torn off the crest.
Height	How high it is thrown, relative to the crest height.
Lifetime	How fast a wisp churns through.
Wind Influence	How far the wind carries it.
Density	Overall opacity, and how far away spray is still drawn.

The curtains are procedural geometry, so there are no particles to budget. At most sixteen are drawn per frame, and the **Spray** quality setting switches the whole thing off.

Spray off an ordinary crest

Wave Crest Spray is a separate system with its own switch, and it is the effect that makes a rough sea read as rough rather than as tall smooth hills. It is the veil of fine water that comes off a large, steep crest in open ocean.

It depends on nothing else. No object has to be in the water, the spray curtain above can be off, shoaling can be off, and there need be no coastline anywhere. A dynamic or rogue wave that is tall enough and steep enough throws spray. That is the whole condition.

Two floors, and a wave has to clear both:

Setting	What it means
Crest Threshold	Wave height in metres below which a wave sprays nothing.
Steepness Threshold	Steepness below which it sprays nothing.

Height alone is the wrong test — a long ocean swell can be six metres tall and perfectly smooth, and smooth water does not come apart. Above each floor the effect comes in over roughly the width of the threshold itself, so "2 m" means nothing at 2 m and full spray by about 4 m.

Setting	What it means
Particle Rate	Puffs per second across every qualifying crest in range. The main cost dial.
Particle Strength	How much water each puff carries and how hard it leaves the crest.
Particle Size / Lifetime / Height	The look of a single wisp.
Wind Influence	How strongly the wind carries it once it is off the crest.
Randomness	How much every puff varies.
Max Distance	Beyond this, crests stop spraying.

Long lifetimes with strong wind give the drifting veils of a storm; short lifetimes with little wind give crisp bursts that fall straight back.

The rate is a budget for the whole sea, not a rate per wave, so six big waves in view cost what one does. **Crest Spray Budget** in the quality settings caps how many particles live at once — zero switches the system off outright.

CHAPTER 11

Water appearance

Colour by depth

The water is coloured in three zones that blend into each other with the amount of water in front of the eye, plus a horizon colour that takes over with distance.

Setting	What it means
Shallow Color / Mid Color / Deep Color	The three zones.
Shallow Depth / Mid Depth / Deep Depth	The water depths in metres where each zone ends.
Horizon Color, Horizon Blend	The colour the water takes at distance, and how quickly.
Scatter Color, Scatter Strength	Light coming back up through the water, and how strong it is.
Scatter Height Influence	How much a raised crest glows.
Absorption	Per-channel absorption. Red goes first in real water, which is why deep water is blue.
Transition Softness	0 gives hard bands, 1 a smooth gradient.
Opacity Depth	How much water it takes to hide what is behind it.

A turquoise lagoon running out into dark blue open water is three colours and three depths, nothing else. The inspector draws the resulting gradient as a strip above the fields, so you can dial it in without switching to the Game view.

Surface detail

Setting	What it means
Detail Tiling A / B	The size of the two procedural normal layers, in metres.
Detail Strength	How strong they are.
Detail Speed	How fast they drift. Wind multiplies this.
Detail Fade Distance	Where they stop being drawn.
Smoothness	How mirror-like the surface is.
Specular Strength	Brightness of the sun glitter.
Fresnel Power / Strength	How much more reflective the water gets at grazing angles.

Scene lighting

The ocean is lit by every light in the scene, not only by the sun. Point and spot lights produce specular highlights built from the same wave normal the sun uses, so a lantern on a pier glints off the crest that is really there and the highlight travels with the wave. Spot cones and light shadows are respected.

Everything is in the profile's **Scene Lighting** group, and every value is a scale, so a profile blend fades it and zero means it contributes nothing:

Setting	What it does
Main Light Enabled	Lets the directional light reach the water at all.
Main Light Shadows	Lets its shadow map darken the water.
Additional Lights Enabled	The point and spot light loop itself.
Additional Specular	Strength of the lamp highlights.
Additional Diffuse	How much lamps brighten the body of the water, not just glint off it.
Additional Light Shadows	Lets lamps cast shadows onto the water.
Additional Highlight Size	Widens lamp highlights into streaks, without touching the sun glitter.
Ambient Strength	How much scene ambient tints the water.

A night harbour is the case all of this exists for: turn the main light down, leave the lamps on, raise Additional Highlight Size so their reflections stretch into the water the way real ones do, and add a little Additional Diffuse so the water under the quay reads as lit.

Refraction

What can be seen through the surface, distorted.

Setting	What it means
Enabled	Off saves a screen colour sample.
Strength	How far the image behind the water is displaced.
Depth Influence	How much shallower water distorts less.
Max Distance	Beyond this the effect is dropped.

Reflections

Three sources, **stacked** rather than chosen between. Each has its own switch and its own weight, so any one can be taken out without disturbing the others — and all three off is a legitimate look for muddy water.

Reflection probes are the probes covering the water, and the sky where there are none, with blending and box projection following your URP asset's own settings. The cheapest layer and the only one that has anything to show at the horizon, so it is usually worth leaving on underneath the others.

Planar is a second camera mirrored about the water plane, rendered into a texture. Exact and stable, including for things off screen, at the price of rendering the scene twice. Use it for calm water and hero shots, and limit what it draws with **Reflection Layers** on the manager.

Screen space traces the depth buffer for what the water is really reflecting. It picks up detail no probe can, but only for what is on screen, so it fades back to the layers underneath at the edges of the frame. Off by default.

Screen space setting	What it means
Screen Space Steps	The cost. More of them reach further and catch thinner objects.
Screen Space Max Distance	How far a ray may travel.
Screen Space Thickness	How deep the depth buffer is assumed to be. Too small and rays pass through things; too large and they hit empty air.
Screen Space Edge Fade	The width of the band at the screen border where the reflection hands back to the probes.

The ocean cannot reflect itself — screen space reflections read the opaque colour buffer, which is filled before any transparent surface is drawn. Seen from below the water they are switched off, since the ray leaves the water and there is nothing for it to find.

CHAPTER 12

Wind and storms

```
ocean.SetWind(directionDegrees: 210f, speed: 24f, gustStrength: 8f);
```

Wind drives wave energy, the dominant wave direction, foam drift, spray, and the direction new dynamic waves travel in. Gusts are deterministic, derived from the ocean's own clock, so a replay or a networked session produces the same wind.

Setting	What it means
Direction Degrees	The bearing, around world Y. 0 is +X, 90 is +Z.
Speed	Metres per second. This is the main dial for how much sea there is.
Gust Strength	Extra speed added by gusts.
Gust Frequency	How quickly gusts rise and fall.
Direction Variation	How far the bearing may wander.

A weather system is usually just a profile blend:

```
IEnumerator StormRollsIn()
{
    ocean.LerpToProfile(roughSea, 40f);
    yield return new WaitForSeconds(60f);
    ocean.LerpToProfile(storm, 60f);
}
```

Wind, storms and the dynamic wave budget are global to a water body. They are properties of the weather, not of a pond.

CHAPTER 13

Day and night

Add **GameObject → OceanCore → Day Night Cycle**. One per scene.

It advances the time of day, turns your directional light with it, and changes how the water looks as it goes.

The clock

Setting	What it does
Time Of Day	The current time in hours, 0 to 24.
Day Length (s)	Real seconds one full day takes. 600 is a ten minute day.
Frozen	Stops the clock. Everything still applies, so the slider above becomes a scrubber — which is exactly what you want while dressing a scene.
Advance In Edit Mode	Lets the clock run outside play mode. Off by default, because the sun is a scene object and a running clock leaves the scene dirty.

The inspector has **Dawn / Noon / Dusk / Night** buttons and shows the sun's real elevation.

From script:

```
cycle.TimeOfDay = 18.5f;      // jump to dusk
cycle.AdvanceHours(+1f);      // move an hour
cycle.SetFrozen(true);        // stop the clock
cycle.ToggleFreeze();
string clock = cycle.TimeText;   // "18:30"
```

The sun

Setting	What it does
Drive Sun	Turn the light and drive its colour and intensity.
Sun	Which light. Left empty, the brightest directional light in the scene is used.
Sunrise Hour / Sunset Hour	When it crosses the horizon.
North Offset	The compass bearing it rises over.
Arc Tilt	How far its arc leans away from straight overhead. 0 puts the sun exactly at the zenith at noon, which is a look almost nowhere on earth has.
Sun Colour	A gradient over the day.
Sun Intensity	A curve over the day.
Drive Ambient	Also write the scene's ambient light. Off by default — ambient belongs to the lighting settings, not to this object, and writing it leaves them changed.

The sun travels one continuous arc: 0° at sunrise, 90° at noon, 180° at sunset and straight on through the night. It does not park at the horizon and jump.

Two ways to change the water

Both are available, they can both be on at once, and **at least one has to be**. Switching both off switches the colour grade back on.

Profile keyframes give each time of day its own ocean profile and blend between them, wrapping around midnight. This is **absolute**: at 03:00 the sea is whatever the 03:00 profile says, wind and waves included.

Colour grade leaves the profile alone and tints the live water colours towards the time of day. This is **relative**: it works on whatever profile happens to be in force, which is why one clock can drive an open sea and a green lake at the same time.

Colour grade setting	What it does
Surface Tint	A gradient multiplied into the shallow, mid, deep and scattering colours.
Brightness	A curve over the day. Also scales subsurface scattering.
Horizon Tint	Blended into the horizon colour after the tint. This is the band the sky meets, so it carries most of a sunset.
Horizon Strength	How much of that gradient replaces the profile's own horizon colour.
Grade Underwater Fog	Tint the underwater fog with the same curves, so a night dive is dark.

Neither writes to a profile asset. The grade replaces the live colours only, and it always reads the profile's own colours rather than its own output, so nothing accumulates.

Targets

Oceans is which water bodies the clock drives. Leave it empty and it drives every ocean in the scene.

> Profile keyframes are absolute, so every targeted ocean is given the same profile. A lake with a look of its own should either be left out of that list, or driven by the colour grade instead, which keeps each water body's own colours. The inspector says so when more than one ocean is targeted.

While profile keyframes are on, the keyframes own the sea: a profile set from a script or from the demo keys is overwritten on the next frame. That is intended — if you have given each hour a profile, the hour decides.

Part 3

Land and water

OceanCore — Dynamic Ocean & Wave Simulation for Unity 6 URP

CHAPTER 14

Islands, rocks and the sea bed

This is the second half of what makes OceanCore distinctive: what the sea does when it reaches land.

Nothing is baked

OceanCore finds the sea bed **while the game runs**. A square window of ground heights follows the camera and is filled by casting rays straight down — a few rows per frame, so it never costs a spike — and swapped in whole when the sweep finishes. There is no bake button, no asset to create, no wizard step and nothing to keep up to date.

That means a procedurally generated island works, a drawbridge going down works, and a terrain you sculpt in the editor works. The window re-sweeps itself every few seconds anyway, so a change is picked up on its own.

The heights are stored **signed**: positive under water, negative for land standing above the water line. That is what lets the water run *up* a beach rather than stopping at the still water contour.

Telling the water what counts

Two settings decide what the waves run into:

- **Wave Collision Layers** on the manager. Everything on these layers is scanned as sea bed. The default is everything.
- **Ocean Wave Obstacle**, on a terrain, an island, a sand bank, a jetty, a rock or a hull.

Adding the component is the whole setup.

Ground or something floating in it

The component works two ways, and **Displacement** decides which.

Mode	What happens
Automatic	The default. Anything with a non-kinematic Rigidbody is treated as floating; everything else as ground.
Displaces	Ground. The sea bed scan sees it, the water line ends at it, waves shoal over it. Islands, rocks, terrain, harbour walls.
Floats	A thing in the water. The scan sees straight through it, so the ocean keeps rendering underneath instead of a hole being cut in it. It still gets foam along its waterline and still stops waves if it has a footprint. Hulls, crates, buoys, wreckage.

This is the setting to reach for when a floating object has cut a hole in the sea. On Automatic it is decided for you, and it is normally right.

> The scan casts rays onto **colliders**. An object with a mesh renderer and no collider is invisible to it — the inspector says so.

The footprint

Separately from the sea bed, an obstacle can have an analytic **footprint**: a circle or a box that waves die against. It needs no scan and it follows the transform, so it suits anything that moves.

Setting	What it means
Enable Footprint	On by default. Off for a terrain, whose shape the scan already has.
Shape	Circle for islands and rocks, Box for piers, walls and rectangular hulls.
Auto Fit	Takes the footprint from the colliders.
Radius / Size / Offset	Set it by hand when Auto Fit is off.
Feather	Distance in metres over which an approaching wave loses its height. Larger dies further out and more gently.
Breaking Strength	How strongly the dying wave reads as breaking, which is what turns it white.

A terrain is not a thing standing in the water, it is the ground the water lies on, so fitting a footprint around one would produce a circle the size of the whole terrain and flatten the sea for a kilometre. On Automatic a terrain therefore contributes to the sea bed only, with no footprint gizmo — that is the correct result, not a missing one.

A rock or a moored hull gets both: the scan for the water line and the colour, and the footprint for the waves that die against it as it moves.

Foam along the water line

Contour Foam draws foam along this object's real contour with the water, found by looking rather than baked, so something that falls in gets its ring of foam and carries it.

Setting	What it means
Enable Contour Foam	On by default. Off, this object produces no foam at all.
Foam Width	How far out to sea the surf reaches from the water line, in metres.
Foam Amount	How much foam there is.
Foam Height Threshold	How deep the water may be and still count as surf. Lower pulls the foam into a tight line at the shore; 0 switches the test off.
Foam Falloff	Shape of the band. 1 is a straight ramp, higher pulls it tighter to the shore.
Foam Noise	Tears the band up so it reads as surf rather than a painted ring.
Foam Fade Distance	How far up the wet sand the foam is carried before it fades.

Waves that come in at this shore

Shoreline Waves sends waves in at this object's own shore, aimed by the shape of that shore rather than by the wind. They form offshore, run straight in along the bearing they sit on, may break in the shallows, and collapse at the water line. Eight are alive at once per obstacle.

Setting	What it means
Enable Shoreline Waves	Off by default.
Wave Spawn Distance	How far out from the water line they form.
Wave Spacing	Metres between one wave and the next. The interval follows from this and the speed.
Wave Height / Length / Speed / Width	The wave itself.
Wave Randomness	How much every value varies from one wave to the next.
Waves Can Break / Wave Breaking Strength	Whether and how hard they break in the shallows.
Wave Fade Start Distance	How far from the water line a wave starts losing height.
Shore Fade Distance	How far past the water line it has finished collapsing.

These waves always fade in smoothly at the shore. However big they are, they are never cut off — a wave that lingers loses height and disappears rather than being removed.

A local ocean profile around one object

Use Local Profile gives the water around this object its own Ocean Profile, exactly as a Water Zone does — a calm bay behind a headland, or a different colour around a reef.

Setting	What it means
Ocean Profile	The profile the water takes on here.
Profile Offset	Moves the area relative to the object, for water that is off to one side.
Profile Radius	How far it applies at full strength.
Profile Feather	How far out the water eases back. Everything about the profile fades across this.
Profile Priority	Who wins where two profiles overlap. Higher has the last word.
Profile Fade Time	Seconds it takes to fade in and out when the object is enabled.

Keeping the water on the ground

A wave trough is a hole in the water, and over a sand bar or the shelf off a beach that hole can drop below the ground — so the sea is cut away and the sand shows through in a band that appears and disappears with every wave.

Keep Water Above Terrain, on the manager, stops that. Where the still water already covers the ground, the drawn surface is lifted to sit on it instead of being cut away.

Setting	What it means
Keep Water Above Terrain	On by default. Leave it on.
Terrain Clearance	How far above the ground the lifted water sits, in metres. Default 0.12 — big enough not to z-fight the sand, small enough not to look like it is floating.
Blend Depth	Metres of standing water the lift fades in over, measured from the water line inwards. Default 0.6. Zero here would put a ridge along the shore; this is what makes the beach and the bar meet smoothly.

Above the water line nothing is clamped, so the beach edge is unchanged. Buoyancy sees the same lift, so a hull aground on a bar floats at the height you can see.

Individual objects can opt out: **Keep Water Above Terrain** on an Ocean Wave Obstacle lets the sea cut through that object's own area whatever the body's global setting says.

The sea bed scan settings

Under **Wave collision and depth** on the manager. You will rarely change any of these.

Setting	Default	What it means
Size	600 m	Edge length of the square the scan covers around the viewer.
Resolution	256	Grid resolution of that square. 256 over 600 m is about one sample every 2.3 m.
Rows Per Frame	4	How many rows are cast per frame. Lower is gentler on the frame and slower to settle.
Refresh Interval	4 s	How often the window is swept again even if nothing moved. 0 never re-sweeps.
Max Runup	25 m	How far above the water line land height is recorded. The run-up cannot climb higher than this.
Hits Per Ray	3	How many surfaces a ray may look through before giving up on finding the bed under a floating object.
Ray Start Height	500 m	How far above the water the rays start. Must clear the tallest thing the waves should break against.
Ray Length	2000 m	How far they travel down.
Smoothing	0	How far the finished sea bed is rounded off, in metres.

Fallback Depth, above them, is the depth reported outside the scanned window — so the open ocean is unaffected by whatever is inside it.

> **Smoothing is the setting for an angular water line.** The water line is a contour of the sea bed, so on a flat beach a centimetre of step in the ground moves the drawn line a long way sideways and the shore looks like it was cut with scissors. Raise Smoothing until it reads as a line. Small rocks are preserved — the filter only rounds off ground that is already close in height, so a boulder standing proud of the sand keeps its foam.

The manager's Depth section shows the scan's live state (*sweeping, 40%* / *scanned, 600 m window*) and has a **Rescan sea bed now** button for when something large has changed and the next second matters. From script that is `ocean.RescanDepth()`.

CHAPTER 15

The shore wash

This is the sheet of water and foam that runs up a beach with every wave, spreads out, and drains back. It is on by default, and it is what you see at the water line.

The whole effect hangs off a single number, the **water film**: how far the displaced water surface stands above the sand underneath it. The water line is simply where the film runs out, and the foam sits where the film is thin. Both come from the same displaced vertex the mesh is drawn with, so the foam edge *is* the water edge and the two cannot drift apart, no matter what the waves do.

That also means the run-up is real rather than animated. As a swell lifts the surface at the beach, the film goes positive further inland and the sheet is drawn further up the sand; as the swell passes, it drains back. On a 1-in-20 beach, half a metre of swell moves the water line about ten metres each way.

Setting	What it means
Enabled	Turns the wash on.
Strength	Overall amount of white. 0 keeps the water line and removes the foam.
Surf Zone Depth	Depth below which a vertex counts as being in the surf. Only these pay for the extra work.
Foam Depth	Film thickness over which the leading foam fades back into open water.
Irregularity	How far the water line wanders, in metres. This is what turns a contour into tongues and bays.
Irregularity Scale	The size of the largest lobes.
Irregularity Drift	How fast the lobe pattern changes, so no two waves break the same way.
Lateral Spread	How much wider the tongues are than they are long.
Run-up Bias	Extra foam on the up-rush compared with the backwash.
Swash Memory	How long the foam remembers where the water just was.
Backwash Foam	Strength of the foam left behind on the draining sheet.
Streak Strength / Streak Length	The long streaks trailing back over the water.
Breadth	How far the spent foam carries back over the water behind the leading edge.
Detail	How hard that foam is torn into clumps and holes. 0 is a flat sheet of white.
Detail Scale	Size in metres of the largest clumps. Two finer octaves follow automatically.
Detail Sharpness	How much finer the breakup gets close to the camera.

Making it look like surf rather than a line

Three things draw white water, all from the same film: the **leading edge**, thickest where the sheet is thinnest; the **swash memory**, foam left where the water stood a moment ago, which is what makes the wash fade out on the wet sand instead of being dragged back out; and the **streaks** trailing seaward, anchored to the water so they travel with it.

A depth contour on its own would read as a rigid mask, so the film is offset by a pair of drifting noise fields before it is used. Because that offset is added to the film — a distance in metres — and not to the finished foam, it moves **the water line itself**: the sheet pushes up the sand in broad tongues and hangs back in bays between them, and because the two fields drift at different rates the shape never repeats.

Breadth and **Detail** are the two settings that turn a line into surf. Breadth carries the spent foam back over the water instead of leaving a rim; Detail tears it up so it reads as bubbles rather than paint.

> The wash never changes the height of a wave. Buoyancy and the visible surface are identical whether it is on or off.

CHAPTER 16

Shoaling and breaking on a coast

Separate from the shore wash, and **off by default**. This is waves slowing, steepening and toppling over as the sea bed rises under them — it changes the shape of the waves themselves.

Switch it on with **Enabled** in the profile's **Shore Shoaling** group when you have a coastline and want the sea bed to reshape the swell as well as the water line.

The model follows linear wave theory:

- The wave **slows down** as the depth falls.
- Its wavelength shortens with the speed, so it **steepens**.
- Green's law makes it **grow taller**.
- Once the wave height passes **Breaking Depth Ratio** × depth, the crest goes unstable, the height is capped, foam is produced and the wave **breaks**.
- Energy bleeds out through **Breaking Energy Loss**, and below **Beach Depth** the surface is flattened onto the beach and the wave **collapses**.

Setting	What it means
Enabled	The whole system.
Shoaling Strength	How much of the theoretical height gain is applied.
Max Shoaling Gain	A ceiling on it, so a wave cannot grow without bound.
Breaking Depth Ratio	Height over depth at which the crest goes unstable.
Breaking Steepness Gain	How much steeper a breaking wave gets.
Breaking Energy Loss	How fast it bleeds height after breaking.
Beach Depth	The depth below which the surface is flattened onto the sand.

Where the shore wash is active it stands this beach fade down, because the fade flattens the surface onto the sand and would pin the water line to the still water contour. The two features do not fight.

CHAPTER 17

Lakes, rivers and local water levels

The sea is at Y = 0. A lake on the hillside is at Y = 80. A river runs from one to the other, and a harbour basin is held five metres lower than the water outside it.

All of it is **one water surface**. There is no second water plane, no fake river mesh and no separate lake system: the ocean's own surface is what stands at eighty metres over the lake, and everything that has ever worked on that surface still works — the swell, dynamic waves, rogue waves, breaking waves, foam, spray, buoyancy, the camera's water line and every surface query a script makes. A boat sailed onto the lake floats on the lake.

Three components, on **GameObject → OceanCore → Water**:

	What it is
Lake	Water standing at its own height somewhere on the land.
River	Water running downhill along a course you draw.
Water Level Zone	The sea that is already here, held at a different height.

Each shows two or three settings; everything else is behind an **Advanced** tick.

> **The one rule that explains all three**
> **The object's Y is the water's height.** Drag a lake up and the lake rises. There is no separate height field to keep in step, because there is only one number and the transform is it.

Lakes

Add one, drag it into a basin, set **Size**. Done.

Where the terrain stands higher than the water, the terrain hides the water — there is nothing to configure for that. On the way out, **Blend Distance** eases the level back to whatever it would otherwise be, so there is no wall of water at the edge. A lake in a basin needs very little, because the ground is already doing the hiding.

Shape is Circle, Box or **Spline** — an outline you draw point by point. Press **Edit outline**, then **Square** or **Ring** to start from something, drag the points, click the small markers between them to insert, Ctrl-click (Cmd on macOS) to remove.

> **A lake high above the sea should be given a Water Depth** (under Advanced). The sea bed scan is measured against the sea's level and stops recording ground height at its **Max Runup**, so above that it cannot tell how deep the lake is and the water is treated as open ocean. Set the depth and the shoaling, the colour and the transparency all have something true to work from.

Rivers

Add one, press **Draw course**, and click along the ground from the lake down to the sea. Each click reads the height of the terrain under it, so the workflow is simply to walk the valley with the mouse. Drag a point to move it; click a marker between two points to insert one; Ctrl-click a point to remove it.

A river is not a tilted sheet of water. Between the points the level follows a curve that matches the slope at both ends, so a handful of clicks becomes a continuous bed with no kinks at the joins. Two rules shape it first:

- **Water does not run uphill.** A point placed higher than the one above it becomes a flat stretch at the upstream height — a pool, which is what actually happens when the ground rises in front of a stream. Rough clicking is forgiven rather

than punished.

- **The curve never overshoots.** The slopes are fitted so the level between two points can never dip below the lower of them or rise above the higher.

So `80 → 72 → 58 → 35 → 12 → 0` comes out as a bed that falls smoothly the whole way, not as five ramps bolted together.

Setting	What it means
Width	How wide the river is, in metres.
Flow Speed	How fast the water runs on the gentle stretches. Steeper ground makes it faster on its own.
Reverse Flow	One button. Sends the water the other way along the same bed.
Transition Length	How far past the bank the water eases back out. This is what blends a river into a lake or the sea instead of ending at a line.
Falloff	The shape of that edge. 1 is the plain S-curve a zone's Feather uses; lower holds the water wider and then lets go.
Mouth Fade	How far back from each connected end the river becomes the water it joins.
Water Depth	How deep it is, for the colour and the transparency.

The current is real

The flow is a vector taken from the direction of the course, not a scrolling texture. The same vector **carries the surface** — the ripples, and with them the glitter, travel downstream at the speed the water is actually doing; **carries the foam**, so it runs with the river instead of sitting in it like paint; **carries what floats**, so a raft goes downstream because the water is going downstream; and **turns white where it steepens**, from the gradient of the river's own profile.

Drawing the course from the sea up to the lake works exactly as well as the other way — which end is downhill is read off the ground, not off the drawing order. **Reverse Flow** is only ever needed as a deliberate override.

The mouth

A river is pinned to the level of whatever its end found — but only that one point is, so without help the entire drop from the point before it lands on a single segment. Click the last point ten metres from a shore six metres up and the water there is a thirty degree ramp with a kink at the top: a visible edge right where the eye is already looking for one.

Mouth Fade is how far back from that end the river starts becoming the sea. Over that distance its surface eases onto the water it joins and arrives *running level with it*.

Mouth Fade	Surface angle at the mouth
off	31°
15 m	0.13°
25 m (default)	0.05°
60 m	0.01°

The same fade dissolves the river's *identity* as well as its height: its current, its own depth, its white water and its profile colour all ease out over the same distance, so by the time you reach the junction the water there is simply ocean. It applies at both ends, and neither fade is ever allowed to reach past the middle of the course.

Auto Connect

Each end looks for something to join onto and is held at exactly that height, so there is no step where they meet. The inspector says what it found:

```
Start    Connected to Lake   ok
End      Connected to Ocean  ok
```

Lakes first, then other rivers, then the sea. Joining the sea takes two things: the end was drawn at about the height the sea stands at there, **and** the ground there is actually under water. Height alone would connect a mountain stream to an ocean a kilometre below it.

If neither end finds anything the river still runs; it may just have a step where it meets other water. Move the end closer, or raise **Connect Distance** under Advanced.

Water level zones

A zone changes the water that is already there. Global ocean at Y = 0, **Target Water Level** at Y = −5, **Blend Distance** 150, and the surface eases from one to the other across that hundred and fifty metres — no step, no seam, and the waves keep running through it.

Zones overlap freely. They are applied in **Priority** order, lowest first, so the highest priority has the last word where they meet.

Giving a lake or a river its own look

Tick **Use Custom Ocean Profile** on a Lake, a River or a Water Level Zone and drop an Ocean Profile under it. That water then looks like its profile instead of like the sea around it: its colour, its absorption, its smoothness, its surface detail, its foam and how much wave it carries.

It eases in and out with the water level. The profile is held by exactly the same blend that holds the height, so a river running out of a lake and into the sea crosses from one look to the next over the same metres it crosses from one height to the next. Lake → River → Ocean never steps, because there is only one blend and all of it is already there.

Where two of them overlap, their profiles average — a river mouth inside a lake reads as a blend of the two rather than as the river stamped over it. A hand-placed Water Zone is different: it is layered *on top*, because someone put it there deliberately.

> **A profile scales the ocean's waves, it does not replace them.** The wave bands — the wavelengths, the directions, how many of each — are one set shared by the whole water body. What a second profile changes is how much of them stands here. A lake profile asking for a tenth of the sea's wave height gets a tenth; the shapes are the ocean's. So a lake can hold at most four times the body's wave height, and where the body's own profile asks for none of something there is nothing here to scale up: a river cannot have ripples on a sea whose Detail Strength is 0.

Under water it follows the **camera**, not the position. The underwater view is one full-screen pass and cannot be two things at once, so it takes the profile of whatever water the camera is actually in and eases towards it over about half a second.

At runtime

```
lake.WaterHeight = 82f;                  // the lake rises
zone.TargetWaterLevel = -5f;             // the basin drains
river.Reverse();                         // the current turns round

float y = ocean.SampleWaterLevel(new Vector2(x, z));   // still water level here
OceanSurfaceSample s = ocean.GetSurfaceData(p);        // level + waves + everything else
```

`GetSurfaceData` reports the local level in `height`, the current in `velocity` and the river's white water in `foam`, so buoyancy, gameplay and the shader all read one answer.

Enable or disable one and it grows in or sinks away over its **Fade Time** rather than popping, and a component toggled in the middle of a fade carries on from where it was.

Limits

24 sources per water body sharing a pool of 1024 outline points; 64 points per lake or zone outline, 96 per river course. Water nowhere near any of them costs one comparison.

> **Narrow rivers.** The surface is the ocean's own mesh, which is dense near the camera and coarser with distance, so a river has to be wide enough for that mesh to hold its shape. Close up anything from a few metres works; a very narrow river seen from far away has fewer and fewer vertices across it and its banks soften. Widen it, or raise the clipmap density.

CHAPTER 18

Water zones

A **Water Zone** marks out a stretch of water and changes what happens there. Put the component on an empty GameObject and place it; the zone follows its transform, so one parented to a boat drifts with it.

Shape	Use for
Circle	A bay, a pond. The cheapest.
Box	A harbour basin, a canal, a pool. Rotates with the transform.
Spline	An outline drawn point by point.

Feather is the width of the soft border, in metres. Everything the zone changes fades in across it, which is what keeps the boundary from reading as a line drawn on the water.

A local profile

Switch on **Override Profile** and the water inside gets its own wave height, surface detail, foam and colours. Point the zone at an Ocean Profile asset to take the colours from it, or set them on the component directly.

Waves are **not** stopped at the border. They run into the zone and lose height as they travel, which is what a sheltered basin does to a swell in reality — the swell dies away across the harbour mouth rather than ending at a line. Buoyancy sees the same calmed water, so a boat inside the basin really does sit still.

> Wave **speed** is deliberately not on the list. Varying it across the water shears the wave phase further apart the longer the session runs, so a boundary that looks right in the first minute looks torn in the tenth. Calm comes from Wave Scale, Detail Scale and Foam Scale instead, which is what the eye reads as still water anyway. For waves that genuinely slow down and shorten, give the zone shallower ground and let shoaling do it properly.

A surface overlay

Switch on **Overlay**, assign a texture, and it is drawn on the water inside the zone — duckweed, algae, blossom, slush, spilled oil. The texture's alpha is the coverage.

It is sampled at the undisplaced parcel of water, the same anchor the foam and the detail normals use, so it rides the wave orbit instead of sliding across the surface underneath it. That is the difference between duckweed floating on a pond and a decal painted on one.

Overlay Edge Fade is its own fade width, separate from the shape feather, because growth on water thins out over a different distance than the water itself changes.

Breakup eats irregular holes into it with a slowly drifting noise field — without it a raft of weed is a solid sheet with a soft edge and reads as paint. **Breakup Scale** is the size of the holes, **Breakup Drift** how fast the pattern rearranges (slow is right, a weed raft changes over minutes), and **Overlay Softness** whether the openings have torn edges or dissolved ones.

Parting

Anything that leaves a wake pushes the overlay aside and drags a lane through it, with a ridge of pushed-together growth along its edge, closing again over **Regrow Time**. A clean hole reads as an eraser; the ridge is what makes it read as something shoved out of the way.

It reads the wake trail every moving hull is already dropping, so every object with an Ocean Buoyancy or an Ocean Wake Emitter does this along the path it actually took. **Part Width** is the width of the lane as a multiple of the hull's beam.

This is the one part of a zone that costs real work per pixel. Set **Part Strength** to zero and that loop is skipped entirely.

> One water body draws one overlay texture: the first active zone that has one wins, and the inspector says so when two zones disagree. Tell two zones apart with **Overlay Color** instead, which blends smoothly where they overlap.

CHAPTER 19

Whirlpools

Put an **Ocean Vortex** on any GameObject. Its position is the eye; everything within **Radius** of it is drawn down into a turning funnel and nothing outside it is touched.

The real water surface is what deforms. There is no second mesh and no decal, so the swell, the dynamic waves, the rogue waves and the breaking waves that were already there run into the hole and go round with it — a wave can break on the wall of a maelstrom.

The three modes

Mode is a starting point, not a switch: pick one, press **Apply mode preset**, and the settings below are filled in. They stay editable and nothing overwrites them afterwards.

Mode	What it is
Whirlpool	A wide, shallow, slowly turning bowl. A tide race, or an eddy behind a headland.
Maelstrom	The big one: deep, broad, three arms, heavy white water and a strong current.
Vortex	Tight and violent. A narrow throat with a long drop and a fast spin.

Radius, the timings and the physics switches are left alone by the preset — those are staging decisions, not part of what a maelstrom is.

Shape setting	What it means
Radius	How far it reaches. The transform's scale multiplies it, so it can be sized by dragging.
Strength	One dial over Depth, Center Depression and the arms, for making a vortex read as bigger without changing its footprint.
Depth	How far the broad funnel drops at the eye, in metres.
Center Depression	Extra metres the throat drops, over a much smaller disc. This is what turns a dish into a funnel with a hole in it.
Falloff	How steep the wall is. Low is a wide saucer; high is flat sea with a shaft punched through it.

Motion setting	What it means
Rotation Speed	How fast the arms turn. Negative turns the other way.
Surface Pull	How hard the surface is drawn in towards the eye.
Twist	How far that draw is swept round, so water spirals in rather than running straight at the eye.
Arm Depth / Arm Count / Spiral Tightness	The arms cut into the funnel. Arm Depth 0 leaves a plain bowl.

> **Surface Pull and Twist are 0–1 rather than metres, and that is deliberate.** Buoyancy answers "how high is the water here" by inverting the sideways displacement of the surface, and that only works while the displacement is gentle enough to have one answer. Pulled hard enough, a ring of water folds through itself and boats would shake on exactly the water meant to be carrying them round. The two dials are spent against a budget measured against that limit, so 1 is as hard as the sea can be drawn in and still be something a hull can float on. Overlapping vortices divide the budget between them.

The arms are free of that limit, because they are cut into the *height* and buoyancy never has to invert a height. That is why they can sweep round without bound and why they, rather than the draw, carry the sense of motion.

Appearing and disappearing

Enabling the component does not make a vortex appear — it grows in over **Spawn Time**. Disabling does not make it vanish — it sinks away over **Fade Out Time**, and only then stops costing anything. A vortex toggled in the middle of a fade carries on from where it was.

Move the GameObject at runtime and the whole field moves with it. Parent one to a boat or animate it along a spline and the water follows exactly.

White water and ships

Foam Amount gathers foam along the arms and around the throat. **Spray Amount** throws water off the wall of the throat.

With **Affect Buoyancy** on, the current carries anything floating: swept tangentially around the eye, drawn inwards, and pulled down at the throat. **Flow Speed** is how fast that current runs at its fastest ring — separate from Rotation Speed, so a slow majestic maelstrom can still have a vicious pull. How strongly a particular object is carried is its own **Flow Influence** on Ocean Buoyancy.

With Affect Buoyancy off the water is still visibly drawn down and a hull still rides the funnel, tilting on its wall — it simply is not swept along.

Snap To Surface keeps the GameObject on the water as the sea rises and falls under it. Play mode only; in the scene view the object stays where you put it.

Eight vortices at once per water body. Water outside a vortex's radius costs one comparison.

CHAPTER 20

Several water bodies in one scene

Each Ocean Core Manager is one body of water. It owns its own material and writes its parameters only into that material, so a storm sea at Y = 0 and a green crater lake at Y = 12 never interfere: different waves, different colours, different sea beds, different profiles.

Extent	Use for
Infinite	The sea. Drawn with the clipmap, rebuilt around the viewer, no reachable edge.
Bounded	Lakes, harbour basins, pools. A fixed rectangular patch that stays where the transform is.

A bounded body has a **Size (X, Z)** and a **Vertex Spacing**; the transform position is its centre and its water level. The inspector shows the resulting vertex count and warns when it gets large.

Which body does an object float on?

Buoyancy components and surface probes work it out from their own position when nothing is assigned by hand: bounded bodies win over the endless sea, and among several candidates the one whose surface is closest. A barrel dropped into the lake floats on the lake; the same prefab dropped into the sea floats on the sea. Assign `OceanBuoyancy.Ocean` explicitly to override that.

```
OceanCoreManager water = OceanCoreManager.GetOceanAt(position);
float height = water != null ? water.GetWaterHeight(position) : 0f;
```

`OceanCoreManager.Instances` lists every active body.

Underwater is the one thing that cannot be per body, because the full-screen pass has no material of its own. The body the camera is inside supplies the underwater fog, absorption and caustics; it is picked automatically, preferring a bounded body the camera is standing in over the open sea. **IsPrimary** on the manager reports which one is currently doing it.

Part 4

Things in the water

OceanCore — Dynamic Ocean & Wave Simulation for Unity 6 URP

CHAPTER 21

Buoyancy

Add **Ocean Buoyancy** to any GameObject with a `Rigidbody`.

Sample points are generated automatically from the attached colliders, or you can list them by hand in local space. Each point contributes an upward force proportional to how deep it is submerged, which produces pitch, roll and heave without any special-case code. The drag is computed **relative to the moving water**, so a hull is carried along by a passing swell instead of only bobbing on the spot.

Setting	What it means
Depth Before Submerged	How deep a point has to sink to count as fully submerged.
Buoyancy Strength	1 makes the object float at rest; higher values ride higher.
Water Drag / Angular Drag	Resistance while submerged.
Flow Influence	How strongly horizontal water motion — a river current, a whirlpool — pushes the object. 0 leaves it riding the surface without being swept by it.
Emit Splashes	Foam and a splash burst where the object drops through the surface.
Ripples	An expanding ring across the water each time it breaks the surface, going in and coming back out.
Displace Water	Pushes the water aside and leaves a wake behind while it moves.
Hull Half Width	The beam, for the wake and for breaking-wave interaction. Leave at 0 to take it from the colliders.

For objects that need no physics at all, **Ocean Surface Probe** simply places a transform on the water and optionally aligns it to the surface normal.

Asking the water a question

```
float   h = ocean.GetWaterHeight(position);
Vector3 n = ocean.GetWaterNormal(position);
Vector3 v = ocean.GetWaterVelocity(position);
float   d = ocean.GetDepth(position);

OceanSurfaceSample s = ocean.GetSurfaceData(position);
// s.height, s.normal, s.velocity, s.depth, s.foam, s.breaking
```

`GetSurfaceData` is the one that does the work; the others are thin wrappers around it. Gerstner waves displace the surface sideways as well as up and down, so the query runs a short fixed-point iteration to find the surface point that ends up above the position you asked about. Two or three iterations are visually indistinguishable from an exact solve even in heavy seas; the count is **Surface Query Iterations** on the manager.

A query allocates nothing. Cache the result if you need several fields at one position.

CHAPTER 22

Wakes

A moving hull pushes water aside and leaves a wake behind it. Both come from one component and need no setup beyond adding it.

- **Ocean Buoyancy** already does it for anything floating on OceanCore's own buoyancy. Switch **Displace Water** on — it is on by default.
- **Ocean Wake Emitter** is for everything else: a ship on your own physics, an animated creature, a platform on a spline. It measures its own velocity from the rigidbody when there is one and from the transform when there is not, so an animated object leaves the same wake as a simulated one.

What it draws

The shape is the **Kelvin wake**, and it is worth knowing what that means because it is what makes the result read as real rather than as a texture dragged behind a boat.

- **Bow waves running outwards.** The water the hull shoulders aside leaves as a wave train travelling out to either side. This is real displacement, so the surface visibly heaves up and settles again as a boat passes.
- **A train of transverse crests behind the boat**, whose wavelength scales with speed — and hard: doubling the speed quadruples the wavelength. A slow launch leaves ripples behind it; a fast hull leaves long rollers.
- **The hull's own displacement**: a trough alongside with a raised shoulder either side. Unlike the rest it belongs to the boat rather than to the trail, so it fades within a fraction of a second of the hull passing.
- **The turbulent trail**: churned white water straight behind, widening slowly. This is the only foam the wake produces.

How it is built

Not painted around the boat. Every metre and a half the hull drops a point on its path recording where it was, which way it was pointing and how fast it was going, and each point carries the slice of wake belonging to that stretch of water. A boat that turns leaves a wake that curves along the track it actually took, and a boat that stops leaves its wake behind instead of dragging it around.

The same evaluation runs on the CPU, so a boat rides the wake another boat threw up.

Setting	What it means
Enabled	Turns hull displacement and wakes on. Objects still deposit foam when it is off.
Wave Height	Height of the wave a hull pushes up at the reference speed.
Reference Speed	Roughly the top speed of the boats in your scene. Below it the wake scales with speed squared.
Minimum Speed	Speed below which a hull leaves no wake at all.
Lifetime	How long a stretch of wake takes to fade away.
Stamp Spacing	Distance between two points of the trail. Tighter follows a turn more closely and costs more budget.
Side Wave Strength / Length / Speed / Train / Reach	The waves thrown out sideways.
Transverse Strength	The crests inside the V.
Wavelength Scale	Stretches or shortens the transverse wavelength. 1 is the physical value.

Setting	What it means
Hull Strength / Hull Time	The trough the hull itself digs, and how briefly it lingers.
Hull Affects Buoyancy	Off by default. See below.
Foam Strength / Foam Width / Foam Spread	The churned trail behind the hull.
Foam Wash	The soft sheet of disturbed water between the two arms of the V.
Foam Rail / Foam Rail Width	The two bright arms themselves, and how wide each is.
Foam Breakup / Foam Breakup Scale	How hard the trail is torn into clumps, and the size of them.

Two things worth knowing

The bow waves make no foam on purpose. Foam on the flanks reads as painted stripes trailing beside the hull, which is the one thing that gives a wake away as a decal. The lateral displacement is geometry only, and all the white water sits in the trail behind where it belongs. If you want more visible sideways motion, raise Side Wave Strength or shorten Side Wave Length; if you want more white behind, raise Foam Strength and Foam Rail.

Hull Affects Buoyancy is off on purpose. The newest point of the trail sits directly under the boat that dropped it, so letting the hull trough into the surface query would have the boat measure the hole it just dug, float lower for it, dig deeper, and oscillate. The trough is still drawn either way; it simply does not feed back into the physics that produced it.

Budget. The wake is evaluated analytically rather than rendered into a texture, which is what lets the CPU and the GPU agree on it exactly. **Wake Stamp Budget** in the quality settings bounds it — 48 on Low up to 384 on Ultra. Longer lifetimes and tighter stamp spacing both eat budget; when it runs out it is the oldest, faintest end of a wake that is dropped, never the bright water behind the boat.

CHAPTER 23

Splashes, ripples and interaction

Anything with an Ocean Buoyancy reacts to the water the moment it touches it, including objects instantiated during play. Nothing is baked and nothing is registered by hand: the component finds its water body every physics step, so a crate spawned from a script leaves a wake immediately, and one that drifts from a lake out into the sea takes its wake with it.

Splash particles appear at the exact contact point — the object's own position at the height the water actually has there, so a splash on the face of a wave lands on the face of the wave — and how fast it was moving sets how far the water is thrown. They are switched with **Impact Particles Enabled** in the profile's Spray group, independently of the crest spray curtain: a calm sea preset with no spray at all still splashes when something falls in.

Ripples are rings, not packets — they travel outwards in every direction rather than along a heading, which is the one shape the dynamic wave system cannot make. They are evaluated on the GPU for the drawn water and on the CPU from the same numbers, so a float bobbing where a crate just landed rides the ring you can see.

Impact Ripples setting	What it means
Amplitude	Height of the ring a solid impact throws out.
Wavelength	Distance from one ring crest to the next. Small is fizzy, large is a single slow swell.
Speed	How fast the leading ring travels outwards.
Lifetime	How long a ring takes to fade completely.
Width	Half width of the crest train following the front.
Foam Strength	Foam laid down at the impact. Zero leaves it clean.

A floating object bobs across the surface constantly, and treating every one of those crossings as an impact would ring the water several times a second for nothing. Three settings on the component stop that, and are what to reach for if it still happens:

Setting	What it means
Crossing Depth	How far past the surface a point has to get before the crossing counts.
Ripple Min Speed	How fast it has to cross, measured against the water's own vertical motion.
Impact Cooldown	Shortest time between two impacts from this object.

The cooldown is per object, not per sample point, so a hull with nine points dropping onto a wave together makes one splash rather than nine.

Doing it yourself

Both effects have a public entry point, for rings and splashes nothing buoyant made — a cast fishing line, a cannonball, a spell landing on the water:

```
ocean.SpawnRipple(contactPoint, strength: 1f);
ocean.EmitSplashParticles(contactPoint, impactVelocity, strength: 1f);
ocean.AddSplash(worldPosition, radius: 3f, strength: 0.8f, velocity);
```

Assign your own particle system to **Splash Particles** on the manager to replace the built-in look. It is switched to world simulation space and driven entirely from OceanCore — its own emission and shape modules are not used, because

position, velocity, size, lifetime and colour all arrive per particle.

To drive something else entirely — your own audio, your own VFX Graph — listen for the impact:

```
ocean.Interaction.Impact += e =>
{
    myEffect.transform.position = e.position;
    myEffect.Emit((int)(e.strength * 40f));
};
```

Continuous foam sources implement `IOceanWakeSource`:

```
public bool EmitWake(out Vector3 position, out float radius, out float strength)
{
    position = transform.position;
    radius   = 2f;
    strength = 0.5f;
    return true;
}
```

Register with `ocean.Interaction.RegisterWakeSource(this)`. Ocean Buoyancy and the example boat already do this for hull wakes and propeller foam.

CHAPTER 24

Keeping the water out of a cabin

A boat with an interior has a problem the wake system cannot solve: the ocean surface runs straight through the cabin floor. **Water Exclusion Volume** cuts a hole in it.

Put the component on an object, park that object inside the space it should hollow out, and make it a child of the boat. That is the whole setup.

```
Boat  (Rigidbody, Ocean Buoyancy)
 ├── Hull                  (visible mesh)
 ├── Cabin                 (visible mesh)
 └── Cabin Interior Mask   (Water Exclusion Volume, renderer off)
```

Because the volume is rebuilt from its transform every frame, it follows the boat through every roll, pitch and turn without any bookkeeping on your side.

The three shapes

Box takes the mesh's bounding box. Six planes, and the right answer for a cabin, a hold or any broadly rectangular space. Scale a default cube to fit and you are done.

Convex Mesh follows the mesh's own faces, which is what you want for a sloped roof, a wedge-shaped bow compartment or a tapered hold. The mesh needs **Read/Write Enabled** in its import settings; without it the bounding box is used instead and the inspector says so.

Spline is an outline you draw in the Scene view — no mesh, no child objects, no list of transforms. Press **Edit outline**, then **Square**, **Ring** or **Fit to mesh** to start from something, drag the points, click the small marker on an edge to insert one, Ctrl-click (Cmd on macOS) to remove one. **Top** and **Depth** set how far the hole reaches above and below the object.

A spline may be any shape, including L-shaped and U-shaped ones: it is split into convex pieces automatically when you stop dragging. A shape that crosses itself has no inside, so it is drawn in red and excludes nothing rather than excluding something arbitrary.

For Box and Convex Mesh the region is always convex — an L-shaped interior is not one convex region, so give it two volumes. They cost almost nothing and overlap harmlessly.

Padding

Padding grows the volume by a set number of metres in every direction. A centimetre or two hides the seam where the water surface meets the cabin wall, which is otherwise visible as a thin bright line at grazing angles. Negative padding pulls the mask back inside the mesh, for when the hole is showing at its edges instead.

What it does and does not do

It **hides** the water. It does not hold it back. Buoyancy, `GetWaterHeight` and every other surface query still see the ocean inside the volume — which is exactly right, because that is what keeps the hull floating on the water its cabin is cut out of. If a piece of gameplay needs to know whether a point is in a dry space, ask:

```
if (ocean.Exclusion.Contains(transform.position))
{
    // Inside a cabin. The sea is drawn nowhere near here.
}
```

An underwater camera inside a volume is treated as being above water, so walking into the cabin of a half-sunk boat does not turn the screen green.

The test is per pixel and exact, not a stencil trick, so a cabin in front of the horizon masks the water inside itself and not the sea a kilometre behind it. Draw order does not matter. Up to 32 volumes are considered at once, with 24 planes each and 512 planes in total across the scene; the inspector tells you if that budget runs out.

CHAPTER 25

Boats

OceanCore does not contain a ship system and is not meant to. **Ocean Example Boat** exists to demonstrate the APIs: it drives a rigidbody that floats with Ocean Buoyancy and feeds propeller foam back into the ocean.

Throttle and steering are public, so replacing the input handling is a two-line change:

```
boat.Throttle = myInput.forward;
boat.Steering = myInput.turn;
```

Part 5

The camera

OceanCore — Dynamic Ocean & Wave Simulation for Unity 6 URP

CHAPTER 26

Underwater

The underwater view is a URP renderer feature. It reconstructs the world position of every pixel from the depth buffer and then applies absorption of light over the distance travelled through the water, darkening with depth, fog towards the water colour, animated caustics on submerged geometry, and a slight screen distortion.

Setting	What it means
Enabled	The whole pass.
Fog Color / Fog Density	The colour the picture fades to, and how quickly.
Depth Darkness	How much darker it gets with depth below the surface.
Distortion	Screen wobble.
Caustics Enabled	The moving light patterns on submerged geometry.
Caustics Strength / Tiling / Speed	How bright, how large and how fast.
Caustics Fade Depth	The depth at which they have faded out.

The camera crossing the surface is a smooth blend, not a switch. Hook the transition for audio:

```
ocean.Underwater.SurfaceCrossed += wentUnder =>
{
    audioMixer.SetFloat("LowPass", wentUnder ? 900f : 22000f);
};
```

The ocean surface itself is drawn from both sides and flips its normal when seen from below, so the underside of the waves renders correctly.

A half-submerged camera

The effects do not wait for the camera to go completely under. As soon as the surface cuts the camera, the half of the screen below it is fully under water — fog, absorption, caustics — and the half above renders normally.

The split is per pixel, taken from the local water surface *as a plane*, normal included, from the same wave evaluation the surface is drawn with. So the line rolls and tilts with the swell the camera is sitting in instead of staying level while the sea heaves around it.

Camera Radius is what makes this work. A camera is treated as a small sphere, not a point. Below the surface every direction is under water and the whole screen is submerged; above it, none are. In between — exactly the range a real lens crosses, and where water clings to it — the split travels across the screen. A point-sized camera would cut from all above to all below in one frame.

The line itself is a band with real thickness:

Setting	What it does
Waterline Enabled	The band. Off restores a single fade across the whole screen.
Waterline Radius	How large the camera is treated as being, in metres.
Waterline Thickness	Width of the band in metres.
Waterline Refraction	How far the band bends the image across its width.
Waterline Rim	Brightness of the lit edge along its top.
Waterline Wobble / Waterline Wobble Scale	How much the line ripples, and how many ripples fit across the screen.
Waterline Tint	How far the band is tinted towards the fog colour on its underwater side.
Waterline Overlap	How far above the line the fog starts. See below.

What makes it read as a rounded body of water rather than a drawn line is that the image is *sampled across* it: the band pulls the picture from further into the water below and further into the air above, the way a meniscus does. A cross-fade between two images has no thickness; a refraction does.

> **Why there is an Overlap setting.** The mask that decides which pixels are under water is built from a description of the water surface, and the drawn mesh and any description of it cannot agree to the last pixel. So the fog starts *above* the line by the overlap, and the band is drawn across exactly that strip — whatever disagreement is left ends up underneath the band rather than beside it, and a gap of un-fogged background cannot form. If one ever does, raise the overlap.

Sound above and below

Add **Water Audio** to the camera and give it four clips: an above-water ambience loop, an underwater ambience loop, a dive sound and a surfacing sound. Going under plays the dive sound once and hands over to the underwater loop; coming up plays the surfacing sound and hands back.

The two loops run continuously and trade places by volume over **Crossfade Time**, rather than starting and stopping — a listener bobbing on the surface would otherwise hear the bed restart from the top on every crossing. **Transition Cooldown** stops the one-shots machine gunning in the same situation.

It makes its own audio sources by default. Turn **Create Sources** off and assign your own if they need to be routed to a mixer group.

CHAPTER 27

Wetness

Everything the water has recently touched is drawn wet: rocks, harbour walls, jetties, hulls, shorelines. They go darker and glossier below the level the water reached, and dry from the top down after it drops away.

It works in screen space from the depth and normals buffers, so **nothing in the scene needs a different material** — a wet quay is the quay you already have, drawn wet. It needs the **OceanCore Wetness** renderer feature on your Universal Renderer; the setup button in the Welcome window adds it.

Two things make a wet surface look wet, and both are here. It is **darker**, because a film of water fills the pores and stops the surface scattering light back out of them. And it is **glossier**, because that film is smooth where the surface underneath is not. The darkening alone reads as a stain; the highlight alone reads as plastic.

Setting	What it does
Enabled	The whole effect.
Strength	Overall amount. Zero leaves everything dry.
Darkening	How dark wet surfaces go.
Gloss	How mirror-like the film is — tight bright highlight, or broad sheen.
Gloss Strength	Brightness of that highlight.
Drying Speed	How fast the wet line falls, in metres per second.
Wet Height	How far above the water spray and run-up leave things wet.
Fade Distance	How softly the wetness ends at its upper edge.
Upward Bias	How much more a ledge holds than a wall.

The wet line is a **level**, not a per-position simulation. A tide line on a harbour wall, the dark band round a rock, the wet strake along a hull — all of them are level, because water is. It rises the instant a crest goes higher and falls at Drying Speed afterwards, which is what makes a rock stay dark for a while after the wave that hit it has gone.

The cost is a normals prepass, which URP would otherwise skip. Switch wetness off in the profile or in the quality settings and the pass is not scheduled at all.

Part 6

Performance and building

OceanCore — Dynamic Ocean & Wave Simulation for Unity 6 URP

CHAPTER 28

Quality settings

Quality Level on the manager picks one of four tiers, or **Custom** to set everything by hand. High is the default.

Setting	Low	Medium	High	Ultra
Dynamic wave budget	8	16	32	64
Wake stamp budget	48	96	192	384
Ripple budget	12	24	48	64
Spray particle budget	0	256	1024	2048
Crest spray budget	0	400	1500	3000
Clipmap levels	5	6	7	8
Clipmap resolution	48	64	96	160
Clipmap base size (m)	48	64	80	96
Foam buffer resolution	256	512	768	1024
Foam buffer coverage (m)	384	512	640	768
Foam buffer	on	on	on	on
Spray	**off**	on	on	on
Underwater	on	on	on	on
Refraction	**off**	on	on	on
Additional lights	on	on	on	on
Screen space reflections	**off**	**off**	on	on
Wetness	**off**	on	on	on
CPU samples per frame	256	512	1024	2048

A budget of **zero switches that effect off outright**, and on a Custom tier that zero is respected — it is a setting, not an empty field waiting to be filled in.

What actually costs what

- The clipmap is drawn with one draw call per level per camera. Seven levels is seven draw calls.
- Wave evaluation is **per vertex**. The clipmap *resolution*, not the area it covers, is what drives the vertex shader cost.
- Shoaling is evaluated once per spectrum band rather than once per wave, so the cost does not grow with how many waves the spectrum has.
- Dynamic waves cost one iteration per wave per vertex, but the loop exits early outside the wave envelope.
- Nothing in the runtime allocates per frame. Wave arrays, the foam list and the GPU buffers are all allocated once and reused.

Wave Detail Fade is the other performance lever worth knowing. Raising it above 1 fades short waves out of the geometry earlier, which lets you drop the clipmap resolution without the surface looking noisier.

Turning things off

Foam buffer	Off in the quality settings keeps procedural crest foam only.
Refraction	Off saves a screen colour sample.
Underwater	Off removes the full-screen pass.
Wetness	Off removes the normals prepass.
Dynamic waves	`dynamicWaves.enabled = false` empties the buffer; the shader skips the loop on one compare.
Planar reflections	The most expensive single option, by a wide margin — it renders the scene twice.
Screen space reflections	The most expensive of the three reflection layers, and the first worth dropping on a lower tier.

The limits

These are hard caps, not settings. They exist so nothing can grow without bound.

Breaking waves at once	16 (the profile setting's own range)
Hulls a breaking wave considers	16
Spray curtains drawn per frame	16
Vortices per water body	8
Water level sources per body	24, sharing 1024 outline points
Points per lake or zone outline	64
Points per river course	96
Shoreline waves per obstacle	8
Exclusion volumes	32, with 24 planes each and 512 planes in total
Ripples alive at once	64
Wake stamps in the buffer	512, and at most half of them on any one trail

CHAPTER 29

Shader variants and build time

Shaders are compiled once per **variant**, and variants multiply. A shader with fourteen independent on/off features has 2^14 = 16384 of them before the render pipeline's own lighting and fog variants are counted — and that is how a water shader ends up taking hours to build.

OceanCore's surface shader carries **five** feature keywords: foam, shore wash, refraction, planar reflection and screen space reflection. Thirty-two combinations. Each depth pass carries one.

Everything else optional — dynamic waves, breaking, obstacles, wakes, impact rings, vortices, the water level field, exclusion volumes and zones — is a **uniform test** rather than a keyword: the shader skips it on one comparison against a count those systems already upload. That keeps the variant count small *and* means those features can be switched on and off while the game runs without ever compiling a shader mid-frame.

Stripping unused variants

Optional, and off unless you create the settings asset: **Assets → Create → OceanCore → Shader Stripping Settings**, or the button in **Tools → OceanCore → Welcome**.

Each of the five features can be set to:

Include	Build it. The default, and the only setting under which the feature can be switched on at any time.
Auto	Build it only if some ocean profile in the project switches it on.
Strip	Never build it. The feature cannot then be switched on in the player.

Nothing is stripped by default, because **a variant that is not in the build cannot be switched to at runtime**. Every removal is named in the build log, so a feature that disappears is never a mystery.

Two things to know. **Foam ignores Auto and behaves as Include**, because a wake trail and a whirlpool's white water switch foam on without any profile saying so — scanning the profiles would give the wrong answer. And the stripper runs on **player builds only**; an AssetBundle or Addressables content build does not go through it.

The Welcome window's **Shader variants** card shows the current combination count, and the build logs how many variants were built and how many were removed.

> During a release build, OceanCore also warns if a surface material still has the LOD debug view switched on. That is a warning rather than a strip, because removing the variant would delete the only one that material could use.

CHAPTER 30

Debugging

Tools → OceanCore → Debugger shows live statistics, the dynamic wave list, the sea bed scan's progress and the scene view overlays, and lets you spawn a wave by hand.

Overlays, available both on the manager and in the debugger:

Wave Direction · Dynamic Waves · Rogue Waves · Clipmap LOD · Water Depth · Shoreline · Breaking Zones · Foam · Buoyancy Points · Surface Normals · Wave Height Grid

Show Breaking Debug, in the profile's Breaking Waves group, is separate and draws each detected breaking crest coloured by state. See chapter 9.

Part 7

Reference

OceanCore — Dynamic Ocean & Wave Simulation for Unity 6 URP

CHAPTER 31

Scripting

Every namespace is under `OceanCore`: `OceanCore`, `.Waves`, `.DynamicWaves`, `.Breaking`, `.Shore`, `.Foam`, `.Rendering`, `.Underwater`, `.Wetness`, `.Buoyancy`, `.Interaction`, `.Water`, `.Zones`, `.Vortex`, `.DayNight`, `.UI`, `.Samples`, `.Editor`.

Finding the ocean

```
OceanCoreManager ocean = OceanCoreManager.Instance;              // the first one
OceanCoreManager here  = OceanCoreManager.GetOceanAt(position);  // the one covering a point
IReadOnlyList<OceanCoreManager> all = OceanCoreManager.Instances;
```

Ocean Core Manager

```
// --- State ------------------------------------------------------------------
float                SeaLevel           { get; }   // the transform's world Y
float                OceanTime          { get; }
OceanProfileData     Current            { get; }   // the live state, overrides included
ref readonly
     OceanProfileData BaseState         { get; }   // what the profile says, before overrides
OceanColorSettings   BaseWaterColors    { get; }
OceanProfile         Profile            { get; }
Camera               Viewer             { get; }
OceanQualitySettings Quality            { get; }
OceanExtent          Extent             { get; }
Vector2              BoundedSize        { get; }
Material             RuntimeMaterial    { get; }
bool                 IsPrimary          { get; }   // drives the underwater pass
bool                 IsCameraUnderwater { get; }
OceanDebugFlags      DebugFlags         { get; set; }

// --- Surface queries --------------------------------------------------------
float              GetWaterHeight(Vector3 position);
Vector3            GetWaterNormal(Vector3 position);
Vector3            GetWaterVelocity(Vector3 position);
float              GetDepth(Vector3 position);
float              SampleWaterLevel(Vector2 worldXZ);   // still level, no waves
OceanSurfaceSample GetSurfaceData(Vector3 position);
OceanWaterlineSample SampleWaterline(Vector3 position, float spacing);
bool               Contains(Vector3 position);

// --- Waves ------------------------------------------------------------------
OceanWaveHandle SpawnWave(in OceanWaveSpawnSettings settings);
OceanWaveHandle SpawnWave(Vector3 position, Vector2 direction, float height);
void            CancelWave(in OceanWaveHandle handle);
void            ClearDynamicWaves();

// --- Profiles and weather ---------------------------------------------------
void SetProfile(OceanProfile profile);
void LerpToProfile(OceanProfile profile, float duration);
void LerpToState(in OceanProfileData target, float duration);
void RefreshFromProfile();
void SetWind(float directionDegrees, float speed, float gustStrength = -1f);
void SetWaterColors(in OceanColorSettings colors);
void SetWaveSpectrum(in OceanSpectrumSettings spectrum);
void SetUnderwaterFog(Color fogColor, float density);

// --- Interaction ------------------------------------------------------------
void AddSplash(Vector3 position, float radius, float strength, Vector3 velocity = default);
void SpawnRipple(Vector3 position, float strength = 1f);
```

```
void EmitSplashParticles(Vector3 position, Vector3 impactVelocity, float strength = 1f);

// --- The sea bed ------------------------------------------------------------
LayerMask WaveCollisionLayers { get; set; }
void      RescanDepth();

// --- Subsystems -------------------------------------------------------------
OceanWindSystem        Wind         { get; }
OceanWaveProvider      Waves        { get; }
OceanDynamicWaveSystem DynamicWaves { get; }
OceanRogueWaveDirector RogueWaves   { get; }
OceanDepthProvider     Depth        { get; }
OceanFoamSystem        Foam         { get; }
OceanSurfaceRenderer   SurfaceRenderer { get; }
OceanSprayRenderer     Spray        { get; }
OceanCrestSpray        CrestSpray   { get; }
OceanBreakingWaveSystem Breaking    { get; }
OceanUnderwaterSystem  Underwater   { get; }
OceanWetnessSystem     Wetness      { get; }
OceanInteractionSystem Interaction  { get; }
OceanSplashParticles   SplashParticles { get; }
OceanObstacleSystem    Obstacles    { get; }
OceanExclusionSystem   Exclusion    { get; }
OceanZoneSystem        Zones        { get; }
OceanVortexSystem      Vortices     { get; }
OceanWaterLevelSystem  WaterLevel   { get; }
```

> `RequestDepthBake()` and `BakeDepthNow()` still exist but are obsolete: nothing is baked any more. Both forward to `RescanDepth()`. `SetDepthData()` and the `OceanDepthData` asset type have been removed — the sea bed is scanned while the game runs.

Events

```
ocean.DynamicWaves.WaveSpawned         += (Vector3 pos, float height, bool isRogue) => { };
ocean.DynamicWaves.WaveStartedBreaking += (Vector3 pos, float height) => { };
ocean.Underwater.SurfaceCrossed        += (bool wentUnder) => { };
ocean.Interaction.Impact               += (OceanImpactEvent e) => { };
// e.position, e.velocity, e.radius, e.strength
```

Buoyancy and probes

```
// OceanBuoyancy
float            SubmergedFraction { get; }
bool             IsInWater         { get; }
OceanCoreManager Ocean             { get; set; }
float            HullHalfWidth     { get; set; }

// OceanSurfaceProbe
OceanSurfaceSample Sample { get; }
OceanCoreManager   Ocean  { get; set; }

// OceanWakeEmitter
OceanCoreManager Ocean      { get; set; }
float            Speed      { get; }
float            Submersion { get; }
float            HalfWidth  { get; set; }
```

Water levels

```
// OceanLakeArea and OceanWaterLevelZone (shared base)
float            WaterHeight     { get; set; }
OceanLevelShape  Shape           { get; set; }
List<Vector2>    SplinePoints    { get; }
bool             UseCustomProfile{ get; set; }
OceanProfile     Profile         { get; set; }
float            WaterDepth      { get; set; }
bool             Contains(Vector2 worldXZ);
float            DistanceToEdge(Vector2 worldXZ);

// OceanWaterLevelZone adds
float TargetWaterLevel { get; set; }
float BlendDistance    { get; set; }
int   Priority         { get; set; }

// OceanRiver
List<OceanRiver.Point> Points { get; }
float Width  { get; set; }   float FlowSpeed { get; set; }
bool  ReverseFlow { get; set; }
float TransitionLength { get; set; }   float Falloff { get; set; }
float MouthFade { get; set; }          float WaterDepth { get; set; }
OceanRiverConnection StartConnection { get; }
OceanRiverConnection EndConnection   { get; }
void  Reverse();
void  AddPoint(Vector3 worldPosition);
void  InsertPoint(int after);
```

Zones, vortices and exclusion

```
// OceanWaterZone
OceanZoneShape Shape { get; set; }
float          Feather { get; }
bool           OverridesProfile { get; }
bool           HasOverlay { get; }
List<Vector2>  SplinePoints { get; }

// OceanVortex
float           Radius  { get; set; }
OceanVortexMode Mode    { get; set; }
float           SpawnTime { get; set; }
float           FadeOutTime { get; set; }
bool            AffectBuoyancy { get; set; }
void            ApplyMode(OceanVortexMode mode);

// OceanWaterExclusionVolume
OceanExclusionShape Shape { get; set; }
int                 PartCount { get; }
List<Vector2>       SplinePoints { get; }
void                Rebake();
bool                Contains(Vector3 p);   // via ocean.Exclusion.Contains
```

The day/night cycle

```
float  TimeOfDay        { get; set; }   // hours, 0 to 24
float  Normalized       { get; }        // 0 at midnight, 1 at the next
float  DayLengthSeconds { get; set; }
bool   IsFrozen         { get; set; }
bool   IsDaytime        { get; }
float  SunElevation     { get; }        // degrees, negative at night
```

```
Light  Sun              { get; }
string TimeText         { get; }        // "18:30"
OceanProfile CurrentKeyProfile { get; }

void SetFrozen(bool frozen);
void ToggleFreeze();
void SetTimeOfDay(float hours);
void AdvanceHours(float hours);
void SetProfileKeyframesEnabled(bool enabled);
void SetColorGradeEnabled(bool enabled);
void SetKeyframes(OceanTimeOfDayKey[] keyframes);
void Apply();

static OceanDayNightCycle Instance { get; }
```

Wave obstacles

```
bool  KeepWaterAboveTerrain { get; set; }
bool  ContourFoam           { get; set; }
bool  ShorelineWaves        { get; set; }
float Feather               { get; set; }
OceanProfile LocalProfile   { get; set; }
OceanWaterDisplacement Displacement { get; set; }   // Automatic / Displaces / Floats
OceanWaterDisplacement ResolvedMode { get; }
static IReadOnlyList<OceanWaveObstacle> Active { get; }
```

CHAPTER 32

Components at a glance

Component	Menu path
Ocean Core Manager	OceanCore → Ocean Core Manager
Ocean Buoyancy	OceanCore → Ocean Buoyancy
Ocean Surface Probe	OceanCore → Ocean Surface Probe
Ocean Wake Emitter	OceanCore → Ocean Wake Emitter
Ocean Wave Obstacle	OceanCore → Ocean Wave Obstacle
Water Exclusion Volume	OceanCore → Water Exclusion Volume
Water Zone	OceanCore → Water Zone
Ocean Vortex	OceanCore → Ocean Vortex
Ocean Lake Area	OceanCore → Water → Ocean Lake Area
Ocean River	OceanCore → Water → Ocean River
Ocean Water Level Zone	OceanCore → Water → Ocean Water Level Zone
Day Night Cycle	OceanCore → Day Night Cycle
Water Audio	OceanCore → Water Audio
Demo HUD	OceanCore → Demo HUD
Ocean Demo Controller	OceanCore → Samples → Ocean Demo Controller
Ocean Example Boat	OceanCore → Samples → Ocean Example Boat
Ocean Free Camera	OceanCore → Samples → Ocean Free Camera

Two renderer features go on your Universal Renderer asset rather than on a GameObject: **OceanCore Underwater** and **OceanCore Wetness**.

CHAPTER 33

Troubleshooting

The water is black. The Opaque Texture is off on the URP asset. Tools → OceanCore → Welcome → Enable required settings.

The water is invisible. Check that the surface material is assigned on the manager and that the shader `OceanCore/Ocean Surface` compiled. It needs shader model 4.5.

Nothing moves in the Scene view. Enable **Animate In Edit Mode** on the manager.

Nothing moves at all, in play mode either. Time Scale in the profile's Wave Spectrum group is at zero. Everything derived from the waves freezes with them, including the shore wash.

The surface crawls, wobbles or shimmers when the camera moves. Wave Detail Fade is too low. Put it back to 1. If it persists at close range, the clipmap resolution is very low for the base size — check *Vertex spacing (closest)* in the Runtime section of the inspector.

Waves flood over an island instead of stopping at it. The island is not being seen by the sea bed scan. Either its layer is missing from **Wave Collision Layers**, or it has no collider, or it needs an **Ocean Wave Obstacle**. Turn on the **Water Depth** overlay to check what the scan has found.

Waves do not break near the coast. The scan has not covered the coastline — raise the scan **Size** — or shoaling is off, which it is by default. Turn **Enabled** on in the profile's Shore Shoaling group.

The water line is angular, like it was cut with scissors. Raise **Smoothing** in the Depth Scan settings. The water line is a contour of the sea bed, so a small step in the ground moves the drawn line a long way sideways on a flat beach.

The water line is a smooth curve rather than tongues and bays. Irregularity in the Shore Wash group is at or near zero. It is a distance in metres added to the water film, so it needs to be large enough to matter against the slope: on a shallow beach half a metre already moves the edge by ten.

The beach foam is a thin line rather than surf. Raise **Breadth**, which carries the spent foam back over the water, and **Detail**, which tears it into clumps instead of leaving a flat sheet.

The foam around an island disappeared when I raised Smoothing. It should not — the filter only rounds off ground that is already close in height, so a rock standing proud of the sand keeps its foam. If it really has gone, check that the rock has a collider and that **Enable Contour Foam** is on for it.

A floating object cut a hole in the sea. Its Ocean Wave Obstacle is set to **Displaces** rather than **Floats**. On Automatic this is decided by whether there is a non-kinematic Rigidbody, so give it one or set the mode by hand.

The sea disappears into the ground in a band that comes and goes. Keep Water Above Terrain on the manager is off. Turn it on.

Buoyancy does not match what I see. Inside a breaking wave the query deliberately stops at the lean — the airborne water above the fold is drawn but not felt, because a fold has two answers above the same point. Everywhere else they agree exactly; if they do not, something has modified one side of the shader/CPU pair.

No spray on the breaking waves. The spray *curtain* only appears on waves that are actually breaking. For spray off ordinary crests in open water, that is **Wave Crest Spray**, a separate switch. Check the **Spray** quality setting is on as well — it is off on Low.

Point or spot lights do not show on the water. Check three things in order: **Additional Lights Enabled** in the profile's Scene Lighting group, **Additional Lights** in the quality settings, and your URP asset's own additional light setting — if that is Per Vertex or Disabled, no shader in the project gets per-pixel lights.

A lamp reflects as a hard bright dot rather than a streak. Raise **Additional Highlight Size**. It widens the lobe for the small lights only, so the sun glitter keeps its own shape.

Screen space reflections smear or show things that are not there. Lower **Thickness** first; a window that is too wide lets rays register a hit several metres behind the object they passed. If reflections stop short at the frame border, that is the edge fade doing its job.

Planar reflections are empty. The **Reflection Layers** mask on the manager excludes everything, or planar reflections are switched off on the active profile.

The underwater effect never appears. The **OceanCore Underwater** renderer feature is missing from your Universal Renderer, or the camera the manager follows is not the one you are looking through.

The underwater effects only appear once the camera is fully under. Water Line is off in the profile's Underwater group, which restores the old whole-screen fade. Turn it back on and check **Camera Radius** is not at its minimum.

A thin gap of background shows between the water and the underwater fog. Raise **Overlap** in the Underwater group.

Nothing looks wet. Three things: **Wetness** in the profile, **Wetness** in the quality settings, and the **OceanCore Wetness** renderer feature on your Universal Renderer.

Wetness reaches too high up the cliffs. Lower **Wet Height** and **Fade Distance**.

Water is still drawn inside the cabin. The exclusion volume is not registered with a water body, which the inspector says at the top — it has to sit inside the ocean's footprint for a bounded body to claim it. Failing that, the mesh may be inverted *and* open, so switch the volume to **Box** or **Spline**.

The mask cuts a hole in the sea outside the boat as well. The mesh is not convex, so the volume covers its convex hull. The inspector says so. Split the space into two Box volumes, or draw it as a Spline, which is decomposed automatically.

A thin bright line where the water meets the cabin wall. Raise **Padding** by a centimetre or two.

A water zone does nothing. Three things to check: it needs **Override Profile** or **Overlay** switched on, it has to sit inside a water body's footprint for a bounded ocean to claim it, and a Spline outline needs at least three points. The inspector says which of the three is missing.

Two zones show the same overlay texture. One water body draws one. Use **Overlay Color** to tell them apart.

A lake renders with the sea's colours, or the other way round. Both bodies point at the same profile. Give each its own. If only the underwater view is wrong, check **IsPrimary** — the camera has to be inside the bounded body's footprint for that body to drive the underwater pass.

The demo keys 1–7 do nothing. A Day Night Cycle with **Use Profile Keyframes** on owns the profile and overwrites it every frame. That is intended; switch the keyframes off, or use the colour grade instead.

A feature that works in the editor is missing in a build. Check the build log for OceanCore's stripping summary. If a feature is named there, it was removed — set it to **Include** in the shader stripping settings. Also check that the required shaders are in *Always Included Shaders*; the Welcome window's setup section tests this.

The build takes a very long time. Check how many variants the Welcome window's **Shader variants** card reports. If it is not 32 for the forward pass, something in the shader has been modified.

CHAPTER 34

Frequently asked

Do I have to bake anything? No. The sea bed is scanned while the game runs, so procedural worlds, moving geometry and terrain you are still sculpting all work with no extra step.

Are there textures to import? No. Foam patterns, detail normals and caustics are all generated in the shader, so every pattern scales with world units and there is nothing to re-import.

Can I use several oceans in one scene? Yes — that is what Extent and the per-body materials are for. A sea and three lakes at different heights is a normal setup.

Can the ocean be somewhere other than Y = 0? Yes. The manager's world Y position is the sea level.

Does it work far from the world origin? The clipmap is camera-relative and snapped every frame, so floating point precision holds up a long way out. The sea bed scan follows the camera, so it does too.

Can I add FFT waves later? Yes. `OceanWaveProvider` is the only thing that produces the base spectrum and the shader reads it through one buffer, so a different spectrum generator can be swapped in without touching dynamic waves, shore behaviour, foam or buoyancy.

Does it support HDRP or the Built-in pipeline? Not in this version. OceanCore is URP only.

Is it multiplayer-safe? The waves are deterministic: the spectrum comes from the profile seed, gusts come from the ocean's own clock, and wave phases are integrated rather than read from an absolute time. Two clients with the same profile and the same ocean time see the same sea. Dynamic waves spawned from script are yours to replicate.

Can I drive it entirely from script? Yes. Every parameter is on a profile, and a profile can be built in memory and applied with `LerpToState`. Nothing requires an asset.

CHAPTER 35

Support

Email	Contact@RedicionStudio.com
Discord	https://discord.gg/Y5s5buNha

When reporting a problem, the two most useful things are your **Unity version and URP version**, and a screenshot of the **Ocean Core Manager inspector** with the Runtime section open — it reports what the sea bed scan has found, how many waves are alive and what the clipmap is doing, which answers most questions immediately.