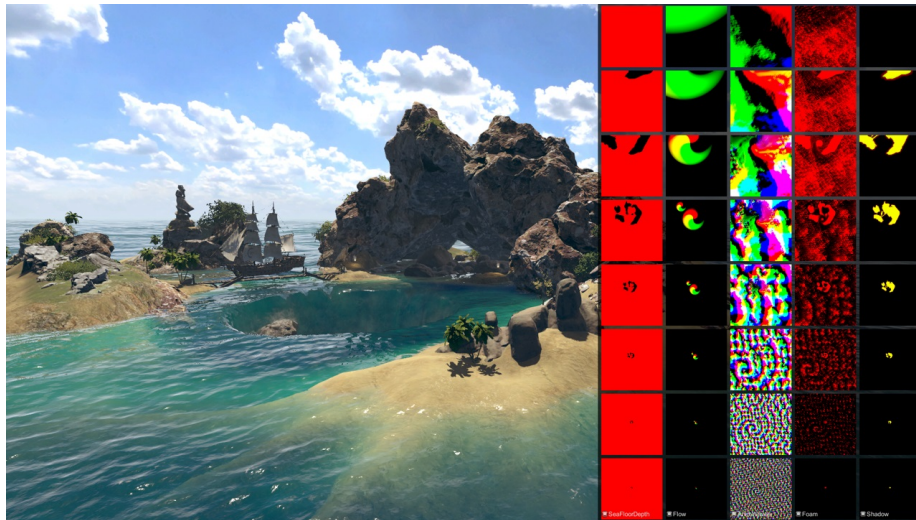


# Crest Ocean System 3.2 for URP

Wave Harmonic Ltd

November 24, 2019



# Contents

<b>1</b>	<b>Overview</b>	<b>3</b>
<b>2</b>	<b>Initial set up</b>	<b>3</b>
2.1	Importing <i>Crest</i> files into project . . . . .	3
2.2	Adding an ocean to a scene . . . . .	4
<b>3</b>	<b>Configuration</b>	<b>5</b>
3.1	Overview . . . . .	5
3.2	Material parameters . . . . .	5
3.3	Reflections . . . . .	8
3.4	Ocean construction parameters . . . . .	9
<b>4</b>	<b>Ocean features</b>	<b>10</b>
4.1	Animated waves . . . . .	10
4.2	Dynamic waves . . . . .	10
4.3	Foam . . . . .	11
4.4	Sea floor depth . . . . .	11
4.5	Shadows . . . . .	12
<b>5</b>	<b>Shallow water and shorelines</b>	<b>13</b>
<b>6</b>	<b>Collision Shape for Physics</b>	<b>14</b>
6.1	Compute Shader Queries . . . . .	14
6.2	Gerstner Waves CPU . . . . .	14
6.3	Ocean Displacement Textures GPU (DEPRECATED) . . . . .	14
<b>7</b>	<b>Other features</b>	<b>15</b>
7.1	Underwater . . . . .	15
7.2	Masking out surface . . . . .	15
7.3	Floating origin . . . . .	15
7.4	Buoyancy / floating physics . . . . .	16
<b>8</b>	<b>Technical documentation</b>	<b>17</b>
8.1	Core data structure . . . . .	17
8.2	Implementation notes . . . . .	19
8.3	Render order . . . . .	20
<b>9</b>	<b>Performance</b>	<b>21</b>
9.1	Quality parameters . . . . .	21
9.2	Potential optimisations . . . . .	21
<b>10</b>	<b>Q&amp;A</b>	<b>22</b>

# 1 Overview

*Crest Ocean System* is a technically advanced ocean system for Unity3D. It is architected for performance and makes heavy use of Level of Detail (LOD) strategies and GPU acceleration for fast update and rendering. It is also highly flexible and allows any custom input to the water shape/foam/dynamic waves/etcetera and customizable material appearance.

This documentation covers the version of *Crest* targeted at the Universal Render Pipeline (URP).

## 2 Initial set up

This section has steps for importing the *Crest* content into a project, and for adding a new ocean surface to a scene.

To augment / complement this written documentation we published a getting started video available here: [https://www.youtube.com/watch?v=TpJf13d\\_-3E](https://www.youtube.com/watch?v=TpJf13d_-3E).

**Note** When setting up Universal Render Pipeline in Unity, or when making changes to packages, we sometimes find that our projects temporarily appear broken. This may manifest as spurious errors in the log or materials appearing magenta. Often, restarting the Editor fixes it. Clearing out the */Library* folder can also help to reset the project and clear temporary errors. These issues are not specific to *Crest*, but we note them anyway as we find our users regularly encounter them.

### 2.1 Importing *Crest* files into project

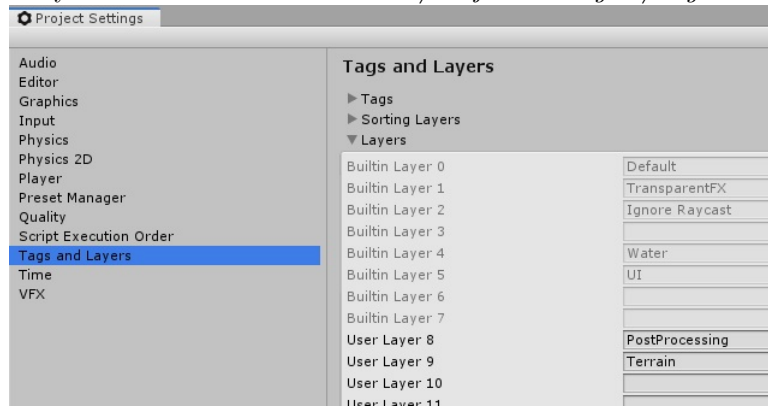
The steps to set up *Crest* in a new or existing project are as follows:

1. Install the Universal Render Pipeline (URP). Select *Window/Package Manager* and install the package *Universal RP. Version 7.1.2 or later is required by Crest*. Restart the Editor to ensure URP setup is complete. It may be necessary to expand out the package in the left pane and click *See all versions* to see the most up to date versions.
2. Import *Crest* package into project using the *Asset Store* window in the Editor.

**Example content** The example scenes under *Crest-Examples* is not required but is provided for illustrative purposes.

3. Enable URP. Open *Edit/Project Settings/Graphics*. Click the select button next to the *Scriptable Render Pipeline Settings* field and select *Assets/Crest/CrestExampleURPAsset*. This has *Depth Texture* and *Opaque Texture* enabled, both of which are required to for water transparency, and has the necessary setup for the Shadows feature (more details in section 4.5).

4. Switch to Linear space rendering under *Edit/Project Settings/Player/Other Settings*. If your platform(s) require Gamma space, the material settings will need to be adjusted to compensate.
5. To run the Crest example scenes, add a layer named *PostProcessing* and a layer named *Terrain* under *Edit/Project Settings.../Tags and Layers*.



6. Open an example scene such as *Assets/Crest/Crest-Examples/PirateCove/PirateCove-Day* and press Play and the ocean will get generated.

## 2.2 Adding an ocean to a scene

The steps to add an ocean to an existing scene are as follows:

1. Create a new GameObject for the ocean, give it a descriptive name such as *Ocean*.
  - (a) Assign the *OceanRenderer* component to it. On startup this component will generate the ocean geometry and do all required initialisation.
  - (b) Assign the desired ocean material to the *OceanRenderer* script - this is a material using the *Crest/Ocean* shader.
  - (c) Set the Y coordinate of the position to the desired sea level for your world.
2. Tag a primary camera as *MainCamera* using the Inspector if one is not tagged already, or set the *Viewpoint* property on the the *OceanRenderer* script. If you need to switch between multiple cameras, update the *Viewpoint* field at run-time to ensure the ocean follows the correct view.
3. To add waves, create a new GameObject and add the *Shape Gerster Batched* component.
  - (a) On startup this script creates a default ocean shape. To edit the shape, create an asset of type *Crest/Ocean Wave Spectrum* and provide it to this script.
  - (b) Smooth blending of ocean shapes can be achieved by adding multiple *Shape Gerstner Batched* scripts and cross-fading them using the *Weight* parameter.

4. For geometry that should influence the ocean (attenuate waves, generate foam):
  - (a) Static geometry should render ocean depth just once on startup into an *Ocean Depth Cache* - the island in the main scene in the example content demonstrates this.
  - (b) Dynamic objects that need to render depth every frame should have a *Register Sea Floor Depth Input* component attached.
5. Be sure to generate lighting from the Lighting window - the ocean lighting takes the ambient intensity from the baked spherical harmonics.

## 3 Configuration

### 3.1 Overview

Some quick start pointers for changing the ocean look and behaviour:

- Ocean material / shading: The active ocean material is assigned to the *OceanRenderer* script. The material parameters are described in section 3.2. Turn off unnecessary features to maximize performance.
- Animated waves / ocean shape: Configured on the *ShapeGerstnerBatched* script by providing an *Ocean Wave Spectrum* asset. This asset has an equalizer-style interface for tweaking different scales of waves, and also has some parametric wave spectra from the literature for comparison.
- Ocean foam: Configured on the *OceanRenderer* script by providing a *Sim Settings Foam* asset.
- Dynamic wave simulation: Configured on the *OceanRenderer* script by providing a *Sim Settings Wave* asset, described in section 4.2.1.
- A big strength of *Crest* is that you can add whatever contributions you like into the system. You could add your own shape or deposit foam onto the surface where desired. Inputs are generally tagged with the *Register* scripts and examples can be found in the example content scenes.

All settings can be changed at run-time and live authored. When tweaking ocean shape it can be useful to freeze time (from script, set *Time.timeScale* to 0) to clearly see the effect of each octave of waves.

### 3.2 Material parameters

**Note:** This section was originally written for the LWRP ocean material. The material parameters may change for Crest URP. We are investigating building the ocean material on ShaderGraph and the material options and parameter names may change. When URP is released and these options are finalised this documentation will be updated.

### 3.2.1 Normal Mapping

**Enable** Whether to add normal detail from a texture. Can be used to add visual detail to the water surface

**Normal Map** Normal map texture (should be set to Normals type in the properties)

**Strength** Strength of normal map influence

**Scale** Scale of normal map texture

### 3.2.2 Scattering

**Diffuse** Base colour when looking straight down into water

**Diffuse Grazing** Base colour when looking into water at shallow/grazing angle

**Shadowing** Changes colour in shadow. Requires 'Create Shadow Data' enabled on OceanRenderer script.

**Diffuse (Shadow)** Base colour in shadow

### 3.2.3 Subsurface Scattering

**Enable** Whether to emulate light scattering through the water volume

**Colour** Colour tint for primary light contribution

**Base Mul** Amount of primary light contribution that always comes in

**Sun Mul** Primary light contribution in direction of light to emulate light passing through waves

**Sun Fall-Off** Fall-off for primary light scattering to affect directionality

### 3.2.4 Shallow Scattering

**Enable** Enable light scattering in shallow water

**Depth Max** Max depth that is considered 'shallow'

**Depth Power** Fall off of shallow scattering

**Shallow Colour** Colour in shallow water

**Shallow Colour (Shadow)** Shallow water colour in shadow (see comment on Shadowing param above)

### 3.2.5 Reflection Environment

**Specular** Strength of specular lighting response

**Smoothness** Smoothness of surface

**Vary Smoothness Over Distance** Helps to spread out specular highlight in mid-to-background. From a theory point of view, models transfer of normal detail to microfacets in BRDF.

**Smoothness Far** Material smoothness at far distance from camera

**Smoothness Far Distance** Definition of far distance

**Smoothness Power** How smoothness varies between near and far distance

**Softness** Acts as mip bias to smooth/blur reflection

**Light Intensity Multiplier** Main light intensity multiplier

**Fresnel Power** Controls harshness of Fresnel behaviour  
**Refractive Index of Air** Index of refraction of air. Can be increased to almost 1.333 to increase visibility up through water surface.  
**Refractive Index of Water** Index of refraction of water. Typically left at 1.333.  
**Planar Reflections** Dynamically rendered 'reflection plane' style reflections. Requires OceanPlanarReflection script added to main camera.  
**Planar Reflections Distortion** How much the water normal affects the planar reflection

### 3.2.6 Procedural Skybox

**Enable** Enable a simple procedural skybox, not suitable for realistic reflections, but can be useful to give control over reflection colour especially in stylized/non realistic applications  
**Base** Base sky colour  
**Towards Sun** Colour in sun direction  
**Directionality** Direction fall off  
**Away From Sun** Colour away from sun direction

### 3.2.7 Foam

**Enable** Enable foam layer on ocean surface  
**Texture** Foam texture  
**Scale** Foam texture scale  
**Light Scale** Scale intensity of lighting  
**White Foam Color** Colour tint for whitecaps / foam on water surface  
**Bubble Foam Color** Colour tint bubble foam underneath water surface  
**Bubble Foam Parallax** Parallax for underwater bubbles to give feeling of volume  
**Shoreline Foam Min Depth** Proximity to sea floor where foam starts to get generated  
**Wave Foam Feather** Controls how gradual the transition is from full foam to no foam  
**Wave Foam Bubbles Coverage** How much underwater bubble foam is generated

### 3.2.8 Foam 3D Lighting

**Enable** Generates normals for the foam based on foam values/texture and use it for foam lighting  
**Normals Strength** Strength of the generated normals  
**Specular Fall-Off** Acts like a gloss parameter for specular response  
**Specular Boost** Strength of specular response

### 3.2.9 Transparency

**Enable** Whether light can pass through the water surface

**Fog Density** Scattering coefficient within water volume, per channel

**Refraction Strength** How strongly light is refracted when passing through water surface

### 3.2.10 Caustics

**Enable** Approximate rays being focused/defocused on underwater surfaces

**Caustics** Caustics texture

**Scale** Caustics texture scale

**Texture Average Value** The 'mid' value of the caustics texture, around which the caustic texture values are scaled

**Strength** Scaling / intensity

**Focal Depth** The depth at which the caustics are in focus

**Depth Of Field** The range of depths over which the caustics are in focus

**Distortion Strength** How much the caustics texture is distorted

**Distortion Scale** The scale of the distortion pattern used to distort the caustics

### 3.2.11 Underwater

**Enable** Whether the underwater effect is being used. This enables code that shades the surface correctly from underneath.

**Cull Mode** Ordinarily set this to *Back* to cull back faces, but set to *Off* to make sure both sides of the surface draw if the underwater effect is being used.

### 3.2.12 Flow

**Enable** Flow is horizontal motion in water as demonstrated in the 'whirlpool' example scene. 'Create Flow Sim' must be enabled on the OceanRenderer to generate flow data.

## 3.3 Reflections

Reflections contribute hugely to the appearance of the ocean. The Index of Refraction settings control how much reflection contributes for different view angles.

The base reflection comes from a one of these sources:

- Unity's specular cubemap. This is the default and is the same as what is applied to glossy objects in the scene. It will support reflection probes, as long as the probe extents cover the ocean tiles, which enables real-time update of the reflection environment (see Unity documentation for more details).
- Override reflection cubemap. If desired a cubemap can be provided to use for the reflections. For best results supply a HDR cubemap.



- **Procedural skybox** - developed for stylized games, this is a simple approximation of sky colours that will give soft results.

This base reflection can then be overridden by dynamic planar reflections. This can be used to augment the reflection with 3D objects such as boat or terrain. This can be enabled by applying the *Ocean Planar Reflections* script to the active camera and configuring which layers get reflected (don't include the *Water* layer). This renders every frame by default but can be configured to render less frequently. This only renders one view but also only captures a limited field of view of reflections, and the reflection directions are scaled down to help keep them in this limited view, which can give a different appearance. Furthermore 'planar' means the surface is approximated by a plane which is not the case for wavy ocean, so the effect can break down. This method is good for capturing local objects like boats and etcetera.

A good strategy for debugging the use of Unity's specular cubemap is to put another reflective/glossy object in the scene near the surface, and verify that it is lit and reflects the scene properly. Crest tries to use the same inputs for lighting/reflections, so if it works for a test object it should work for the water surface as well.

### 3.4 Ocean construction parameters

There are a small number of parameters that control the construction of the ocean shape and geometry:

- **Lod Data Resolution** - the resolution of the various ocean LOD data including displacement textures, foam data, dynamic wave sims, etc. Sets the 'detail' present in the ocean - larger values give more detail at increased run-time expense.
- **Geometry Down Sample Factor** - geometry density - a value of 2 will generate one vert per 2x2 LOD data texels. A value of 1 means a vert is generated for every LOD data texel. Larger values give lower fidelity surface shape with higher performance.
- **Lod Count** - the number of levels of detail / scales of ocean geometry to generate. The horizontal range of the ocean surface doubles for each added LOD, while GPU processing time increases linearly. It can be useful to select the ocean in the scene view while running in editor to inspect where LODs are present.
- **Max Scale** - the ocean is scaled horizontally with viewer height, to keep the meshing suitable for elevated viewpoints. This sets the maximum the ocean will be scaled if set to a positive value.
- **Min Scale** - this clamps the scale from below, to prevent the ocean scaling down to 0 when the camera approaches the sea level. Low values give lots of detail, but will limit the horizontal extents of the ocean detail.

## 4 Ocean features

### 4.1 Animated waves

The Gerstner waves are split by octave - each Gerstner wave component is only rendered once into the most suitable LOD (i.e. a long wavelength will render only into one of the large LODs), and then a combine pass is done to copy results from the resolution LODs down to the high resolution ones.

Crest supports rendering any shape into these textures. To add some shape, add some geometry into the world which when rendered from a top down perspective will draw the desired displacements. Then assign the *RegisterAnimWavesInput* script which will tag it for rendering into the shappe.

There is an example in the *boat.unity* scene, GameObject *wp0*, where a smoothstep bump is added to the water shape. This is an efficient way to generate dynamic shape. This renders with additive blend, but other blending modes are possible such as alpha blend, multiplicative blending, and min or max blending, which give powerful control over the shape.

The animated waves sim can be configured by assigning an Animated Waves Sim Settings asset to the OceanRenderer script in your scene (*Create/Crest/Animated Wave Sim Settings*). The waves will be dampened/attenuated in shallow water if a *Sea Floor Depth* LOD data is used (see below). The amount that waves are attenuated is configurable using the *Attenuation In Shallows* setting.

### 4.2 Dynamic waves

This LOD data is a multi-resolution dynamic wave simulation, which gives dynamic interaction with the water.

One use case for this is boat wakes. In the *boat.unity* scene, the geometry and shader on the *WaterObjectInteractionSphere0* will render forces into the sim. It has the *RegisterDynWavesInput* script that tags it as input.

After the simulation is advanced, the results are converted into displacements and copied into the displacement textures to affect the final ocean shape. The sim is added on top of the existing Gerstner waves.

Similar to animated waves, user provided contributions can be rendered into this LOD data to create dynamic wave effects. An example can be found in the boat prefab. Each LOD sim runs independently and it is desirable to add interaction forces into all appropriate sims. The *ObjectWaterInteraction* script takes into account the boat size and counts how many sims are appropriate, and then weights the interaction forces based on this number, so the force is spread evenly to all sims. As noted above, the sim results will be copied into the dynamic waves LODs and then accumulated up the LOD chain to reconstruct a single simulation.

The dynamic waves sim can be configured by assigning a Dynamic Wave Sim Settings asset to the OceanRenderer script in your scene (*Create/Crest/Dynamic Wave Sim Settings*).

#### 4.2.1 Simulation setup

This is the recommended workflow for configuring the dynamic wave simulation. All of the settings below refer to the *Dynamic Wave Sim Settings*.

1. Set the *Gravity Multiplier* to the lowest value that is satisfactory. Higher values will make the simulated waves travel faster, but make the simulation more unstable and require more update steps / expense.
2. Increase *Damping* as high as possible. Higher values make the sim easier to solve, but makes the waves fade faster and limits their range.
3. Set the *Courant Number* to the highest value which still yields a stable sim. Higher values reduce cost but reduce stability. Put the camera low down near the water while testing as the most detailed waves are the most unstable.
4. Reduce *Max Sim Steps Per Frame* as much as possible to reduce the simulation cost. This may slow down waves in the lower LOD levels, which are the most detailed waves. Hopefully this slight slow down in just the smallest wavelengths is not noticeable/objectionable for the player. If waves are visible travelling too slow, increase it.

The *OceanDebugGUI* script gives the debug overlay in the example content scenes and reports the number of sim steps taken and sim step dt at each frame.

### 4.3 Foam

The Foam LOD Data is simple type of simulation for foam on the surface. Foam is generated by choppy water (specifically when the surface is *pinched*). Each frame, the foam values are reduced to model gradual dissipation of foam over time.

User provided foam contributions can be added similar to the Animated Waves. In this case the *RegisterFoamInput* script should be applied to any inputs. There is no combine pass for foam so this does not have to be taken into consideration - one must simply render 0-1 values for foam as desired. See the *DepositFoamTex* object in the *whirlpool.unity* scene for an example.

The foam sim can be configured by assigning a Foam Sim Settings asset to the OceanRenderer script in your scene (*Create/Crest/Foam Sim Settings*). There are also parameters on the material which control the appearance of the foam.

### 4.4 Sea floor depth

This LOD data stores water depth information. This is useful information for the system; it is used to attenuate large waves in shallow water, to generate foam near shorelines, and to provide shallow water shading. It is calculated by rendering the render geometry in the scene for each LOD from a top down perspective and recording the Y value of the surface.

The following will contribute to ocean depth:

- Objects that have the *RegisterSeaFloorDepthInput* component attached. These objects will render every frame. This is useful for any dynamically moving surfaces that need to generate shoreline foam, etc.
- It is also possible to place world space depth caches. The scene objects will be rendered into this cache once, and the results saved. Once the cache

is populated it is then copied into the Sea Floor Depth LOD Data. The cache has a gizmo that represents the extents of the cache (white outline) and the near plane of the camera that renders the depth (translucent rectangle). The cache should be placed at sea level and rotated/scaled to encapsulate the terrain.

When the water is e.g. 250m deep, this will start to dampen 500m wavelengths, so it is recommended that the sea floor drop down to around this depth away from islands so that there is a smooth transition between shallow and deep water without a visible boundary.

## 4.5 Shadows

The shadow data consists of two channels. One is for normal shadows as would be used to block specular reflection of the light. The other is a much softer shadowing value that can approximate variation in light scattering in the water volume.

This data is captured from the shadow maps Unity renders. These shadow maps are always rendered in front of the viewer. The Shadow LOD Data then reads these shadow maps and copies shadow information into its LOD textures.

We have provided an example configuration with shadows enabled; *Assets/Crest/CrestExampleURPAsset*, which should be set to use the following Custom Renderer: *Assets/Crest/ForwardRendererCrestShadows*. In the setup instructions in section 2, steps to use this asset and renderer were given, and no further action is required if this setup is used.

To create this setup from scratch, the steps are the following.

1. On the scriptable render pipeline asset (either the asset provided with Crest *Assets/Crest/CrestExampleURPAsset*, or the one used in your project), ensure that shadow cascades are enabled. Crest requires cascades to be enabled to obtain shadow information.
2. Create a new renderer which will have the sample shadows feature enabled. Right click a folder under Assets and select *Create/Rendering/Universal Render Pipeline/Forward Renderer*. Select the asset and click the '+' icon and select *Crest/SampleShadows*.
3. Enable the new renderer. Select your URP pipeline asset and set *General/Renderer Type* to *Custom* and assign the asset created in the previous step.
4. Enable shadowing in Crest. Enable *Create Shadow Data* on the Ocean-Renderer script.
5. On the same script, assign a *Primary Light* for the shadows. This light needs to have shadows enabled, if not an error will be reported accordingly.
6. If desired the shadow sim can be configured by assigning a *Shadow Sim Settings* asset (*Create/Crest/Shadow Sim Settings*).
7. Enable *Shadowing* on the ocean material to compile in the necessary shader code

The shadow sim can be configured by assigning a Shadow Sim Settings asset to the OceanRenderer script in your scene (*Create/Crest/Shadow Sim Settings*). In particular, the soft shadows are very soft by default, and may not appear for small/thin shadow casters. This can be configured using the *Jitter Diameter Soft* setting.

Note: URP should allow sampling the shadow maps directly in the ocean shader which would be an alternative to using this shadow data, although it would not give the softer shadow component. This would likely work on 2018.

## 5 Shallow water and shorelines

We published a video on this topic, available here: <https://www.youtube.com/watch?v=jcmqUlboTUK>.

*Crest* requires water depth information to attenuate large waves in shallow water, to generate foam near shorelines, and to provide shallow water shading. It is calculated by rendering the render geometry in the scene for each LOD from a top down perspective and recording the Y value of the surface.

When the ocean is e.g. 250m deep, this will start to dampen 500m wavelengths, so it is recommended that the sea floor drop down to around this depth away from islands so that there is a smooth transition between shallow and deep water without a 'step' in the sea floor which appears as a discontinuity and/or line of foam on the surface.

One way to inform *Crest* of the seabed is to attach the *RegisterSeaFloorDepthInput* component. *Crest* will record the height of these objects every frame, so they can be dynamic.

This dynamic update comes at a cost. For parts of the seabed which are static, *Crest* has a mechanism for recording their heights just once, instead of updating every frame, using an ocean depth cache. The *main.unity* example scene has an example of a cache set up around the island. The cache GameObject is called *IslandDepthCache* and has a *OceanDepthCache* component attached. The following are the key points of its configuration:

- The transform position X and Z are centered over the island
- The transform position y value is set to the sea level
- The transform scale is set to 540 which sets the size of the cache. If gizmos are visible and the cache is selected, the area is demarcated with a white rectangle.
- The *Camera Max Terrain Height* is the max height of any surfaces above the sea level that will render into the cache. If gizmos are visible and the cache is selected, this cutoff is visualised as a translucent gray rectangle.
- The *Layer Names* field contains the layer that the island is assigned to - *Terrain*. Only objects in these layer(s) will render into the cache.

On startup, validation is done on the cache. Check the log for warnings and errors.

At runtime, a child object underneath the cache will be created with the prefix *Draw\_* it will have a material with a *Texture* property. By double clicking the icon to the right of this field, one can inspect the contents of the cache.

## 6 Collision Shape for Physics

The system has a few paths for computing information about the water surface such as height, displacement, flow and surface velocity. These paths are covered in the following subsections, and are configured on the *Animated Waves Sim Settings*, assigned to the OceanRenderer script, using the Collision Source dropdown.

The system supports sampling collision at different resolutions. The query functions have a parameter *Min Spatial Length* which is used to indicate how much detail is desired. Wavelengths smaller than half of this min spatial length will be excluded from consideration.

To simplify the code required to get the ocean height or other data from C#, two helpers are provided, *SampleHeightHelper* and *SampleFlowHelper*. Use of these is demonstrated in the example content.

### 6.1 Compute Shader Queries

This is the default and recommended choice. Query positions are uploaded to a compute shader which then samples the ocean data and returns the desired results. The result of the query accurately tracks the height of the surface, including all shape deformations and waves.

This system does not require detailed configuration and has the best performance characteristics.

### 6.2 Gerstner Waves CPU

This collision option is serviced directly by the *GerstnerWavesBatched* component which implements the *ICollProvider* interface, check this interface to see functionality. This sums over all waves to compute displacements, normals, velocities, etc. In contrast to the displacement textures the horizontal range of this collision source is unlimited.

This avoids some of the complexity of using the displacement textures described above, but comes at a CPU cost. It also does not include wave attenuation from water depth or any custom rendered shape. A final limitation is the current system finds the first *GerstnerWavesBatched* component in the scene which may or may not be the correct one. The system does not support cross blending of multiple scripts.

### 6.3 Ocean Displacement Textures GPU (DEPRECATED)

This collision source copies the displacement textures from the GPU to the CPU. It does so asynchronously and the data typically takes 2-3 frames to arrive. This gives the final ocean shape, including any bespoke shape rendering, attenuation from water depth, and any other effects.

It uses memory bandwidth to transfer this data and CPU time to take a copy of it once it arrives, so it is best to limit the number of textures copied. If you know in advance the limits of the minimum spatial lengths you will be requesting, set these on the *Animated Waves Sim Settings* using the *Min Object Width* and *Max Object Width* fields.

As described above the displacements are arranged as cascaded textures which shift based on the elevation of the viewpoint. This complicates matters significantly as the requested resolutions may or may not exist at different times. Call *ICollProvider.CheckAvailability()* at run-time to check for issues and perform validation.

**Note** This system has been deprecated in favour of the Compute Shader Queries path (section 6.1).

## 7 Other features

### 7.1 Underwater

*Crest* supports seamless transitions above/below water. This is demonstrated in the *main.unity* scene in the example content. The ocean in this scene uses the material *Ocean-Underwater.mat* which enables rendering the underside of the surface, and has the prefab *UnderWaterCurtainGeom* parented to the camera which renders the underwater effect. It also has the prefab *UnderWaterMeniscus* parented which renders a subtle line at the intersection between the camera lens and the water to visually help the transition.

### 7.2 Masking out surface

There are times when it is useful to mask out the ocean surface which prevents it drawing on some part of the screen. The scene *main.unity* in the example content has a rowboat which, without masking, would appear to be full of water. To prevent water appearing inside the boat, the *WaterMask* GameObject writes depth into the GPU's depth buffer which can occlude any water behind it, and therefore prevent drawing water inside the boat. The *RegisterMaskInput* component is required to ensure this depth draws early before the ocean surface.

### 7.3 Floating origin

*Crest* has support for 'floating origin' functionality, based on code from the Unity community wiki. See the original wiki page for an overview and original code: <http://wiki.unity3d.com/index.php/Floating-Origin>.

It is tricky to get pop free results for world space texturing. To make it work the following is required:

- Set the floating origin threshold to a power of 2 value such as 4096.
- Set the size/scale of any world space textures to be a smaller power of 2. This way the texture tiles an integral number of times across the threshold, and when the origin moves no change in appearance is noticeable. This includes the following textures:
  - Normals - set the Normal Mapping Scale on the ocean material
  - Foam texture - set the Foam Scale on the ocean material
  - Caustics - also should be a power of 2 scale, if caustics are visible when origin shifts happen

By default the *FloatingOrigin* script will call *FindObjectsOfType()* for a few different component types, which is a notoriously expensive operation. It is possible to provide custom lists of components to the 'override' fields, either by hand or programmatically, to avoid searching the entire scene(s) for the components. Managing these lists at run-time is left to the user.

## 7.4 Buoyancy / floating physics

*SimpleFloatingObject* is a simple buoyancy script that attempts to match the object position and rotation with the surface height and normal. This can work well enough for small water craft that don't need perfect floating behaviour, or floating objects such as buoys, barrels, etc.

*BoatProbes* is a more advanced implementation that computes buoyancy forces at a number of *ForcePoints* and uses these to apply force and torque to the object. This gives more accurate results at the cost of more queries.

*BoatAlignNormal* is a rudimentary boat physics emulator that attaches an engine and rudder to *SimpleFloatingObject*. It is not recommended for cases where high animation quality is required.

### 7.4.1 Adding boats

Setting up a boat with physics can be a dark art. The authors recommend duplicating and modifying one of the existing boat prefabs, and proceeding slowly and carefully as follows:

1. Pick an existing boat to replace. Only use *BoatAlignNormal* if good floating behaviour is not important, as mentioned above. The best choice is usually boat probes.
2. Duplicate the prefab of the one you want to replace, such as *crest/Assets/Crest/Crest-Examples/BoatDev/Data/BoatProbes.prefab*
3. Remove the render meshes from the prefab, and add the render mesh for your boat. We recommend lining up the meshes roughly.
4. Switch out the collision shape as desired. Some people report issues if there are multiple overlapping physics collision primitives (or multiple rigidbodies which should never be the case). We recommend keeping things as simple as possible and using only one collider if possible.
5. We recommend placing the render mesh so its approximate center of mass matches the center of the collider and is at the center of the boat transform. Put differently, we usually try to eliminate complex hierarchies or having nested non-zero'd transforms whenever possible within the boat hierarchy, at least on or above physical parts.
6. If you have followed these steps you will have a new boat visual mesh and collider, with the old rigidbody and boat script. You can then modify the physics settings to move the behaviour towards how you want it to be.
7. The mass and drag settings on the boat scripts and rigidbody help to give a feeling of weight.



8. Set the boat dimension:
  - BoatProbes: Set the *Min Spatial Length* param to the width of the boat.
  - BoatAlignNormal: Set the boat Boat Width and Boat Length to the width and length of the boat.
  - If, even after experimenting with the mass and drag, the boat is responding too much to small waves, increase these parameters (try doubling or quadrupling at first and then compensate).
9. There are power settings for engine turning which also help to give a feeling of weight
10. The dynamic wave interaction is driven by the object in the boat hierarchy called *WaterObjectInteractionSphere*. It can be scaled to match the dimensions of the boat. The *Weight* param controls the strength of the interaction.

The above steps should maintain a working boat throughout - we recommend testing after each step to catch issues early.

## 8 Technical documentation

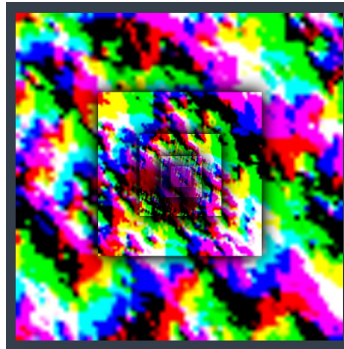
We have published details of the algorithms and approaches we use. See the following publications:

- Crest: *Novel Ocean Rendering Techniques in an Open Source Framework*, Advances in Real-Time Rendering in Games, ACM SIGGRAPH 2017 courses <http://advances.realtimerendering.com/s2017/index.html>
- *Multi-resolution Ocean Rendering in Crest Ocean System*, Advances in Real-Time Rendering in Games, ACM SIGGRAPH 2019 courses <http://advances.realtimerendering.com/s2019/index.htm>

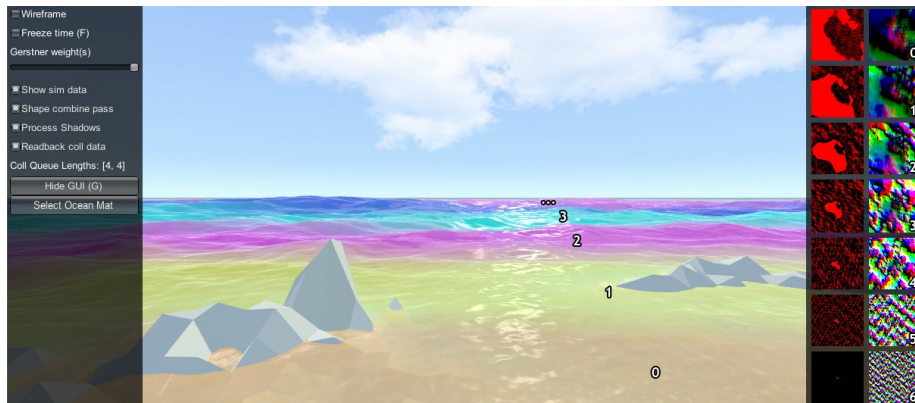
### 8.1 Core data structure

The backbone of *Crest* is an efficient Level Of Detail (LOD) representation for data that drives the rendering, such as surface shape/displacements, foam values, shadowing data, water depth, and others. This data is stored in a multi-resolution format, namely cascaded textures that are centered at the viewer. This data is generated and then sampled when the ocean surface geometry is rendered. This is all done on the GPU using a command buffer constructed each frame by *BuildCommandBuffer*.

Let's study one of the LOD data types in more detail. The surface shape is generated by the Animated Waves LOD Data, which maintains a set of *displacement textures* which describe the surface shape. A top down view of these textures laid out in the world looks as follows:

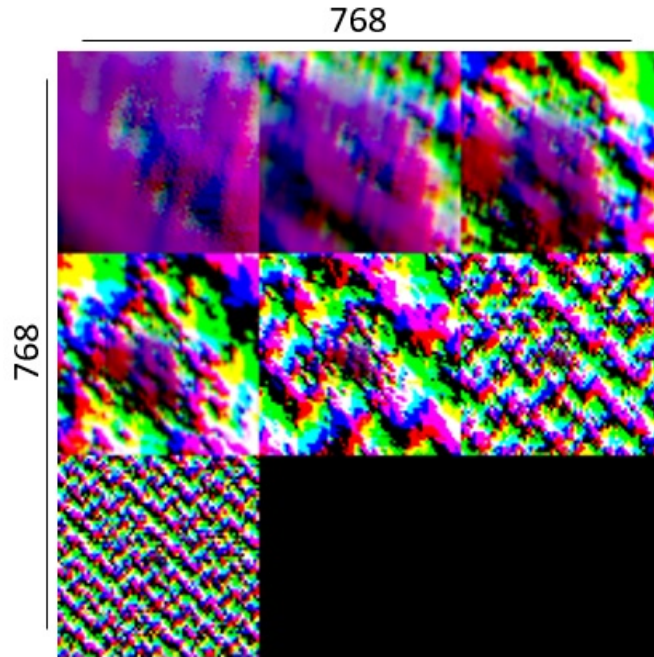


Each LOD is the same resolution (256x256 here), configured on the *Ocean-Renderer* script. In this example the largest LOD covers a large area (4km squared), and the most detail LOD provides plenty of resolution close to the viewer. These textures are visualised in the Debug GUI on the right hand side of the screen:



In the above screenshot the foam data is also visualised (red textures), and the scale of each LOD is clearly visible by looking at the data contained within. In the rendering each LOD is given a false colour which shows how the LODs are arranged around the viewer and how they are scaled. Notice also the smooth blend between LODs - LOD data is always interpolated using this blend factor so that there are never pops or hard edges between different resolutions.

In this example the LODs cover a large area in the world with a very modest amount of data. To put this in perspective, the entire LOD chain in this case could be packed into a small texel area:



A final feature of the LOD system is that the LODs change scale with the viewpoint. From an elevated perspective, horizontal range is more important than fine wave details, and the opposite is true when near the surface. The *OceanRenderer* has min and max scale settings to set limits on this dynamic range.

When rendering the ocean, the various LOD data are sample for each vert and the vert is displaced. This means that the data is carried with the waves away from its rest position. For some data like wave foam this is fine and desirable. For other data such as the depth to the ocean floor, this is not a quantity that should move around with the waves and this can currently cause issues, such as shallow water appearing to move with the waves as in issue 96.

## 8.2 Implementation notes

On startup, the *OceanRenderer* script initialises the ocean system and asks the *OceanBuilder* script to build the ocean surface. As can be seen by inspecting the ocean at run-time, the surface is composed of concentric rings of geometry tiles. Each ring is given a different power of 2 scale.

At run-time, the ocean system updates its state in *LateUpdate*, after game state update and animation, etc. *OceanRenderer* updates before other scripts and first calculates a position and scale for the ocean. The ocean *GameObject* is placed at sea level under the viewer. A horizontal scale is computed for the ocean based on the viewer height, as well as a *\_viewerAltitudeLevelAlpha* that captures where the camera is between the current scale and the next scale ( $\times 2$ ), and allows a smooth transition between scales to be achieved.

Next any active ocean data are updated, such as animated waves, simulated foam, simulated waves, etc. The data can be visualised on screen if the *Ocean-*

*DebugGUI* script from the example content is present in the scene, and if the *Show shape data* on screen toggle is enabled. As one of the ocean data types, the ocean shape is generated by rendering Gerstner wave components into the animated waves data. Each wave component is rendered into the shape LOD that is appropriate for the wavelength, to prevent over- or under- sampling and maximize efficiency. A final pass combines the shape results from the different Gerstner components together. Disable the *Shape combine pass* option on the *OceanDebugGUI* to see the shape contents before this pass.

Finally *BuildCommandBuffer* constructs a command buffer to execute the ocean update on the GPU early in the frame before the graphics queue starts. See the *BuildCommandBuffer* code for the update scheduling and logic.

The ocean geometry is rendered by Unity as part of the graphics queue, and uses the *Crest/Ocean* shader. The vertex shader snaps the verts to grid positions to make them stable. It then computes a *lodAlpha* which starts at 0 for the inside of the LOD and becomes 1 at the outer edge. It is computed from taxicab distance as noted in the course. This value is used to drive the vertex layout transition, to enable a seamless match between the two. The vertex shader then samples any required ocean data for the the current and next LOD scales and uses *lodAlpha* to interpolate them for a smooth transition across displacement textures. Finally, it passes the LOD geometry scale and *lodAlpha* to the ocean fragment shader.

The fragment shader samples normal and foam maps at 2 different scales, both proportional to the current and next LOD scales, and then interpolates the result using *lodAlpha* for a smooth transition. It combines the normal map with surface normals computed directly from the displacement texture.

### 8.3 Render order

A typical render order for a frame is the following:

- Opaque geometry is rendered, writes to opaque depth buffer
- Sky is rendered, probably at zfar with depth test enabled so it only renders outside the opaque surfaces
- Frame colours and depth are copied out for use later in postprocessing
- Ocean 'curtain' renders, draws underwater effect from bottom of screen up to water line (queue = Transparent-110)
  - \* It is set to render before ocean in UnderwaterEffect.cs
  - \* Sky is at zfar and will be fully fogged/obscured by the water volume
- Ocean renders early in the transparent queue (queue = Transparent-100)
  - \* It samples the postprocessing colours and depths, to do refraction
  - \* It reads and writes from the frame depth buffer, to ensure waves are sorted correctly
  - \* It stomps over the underwater curtain to make a correct final result
  - \* It stomps over sky - sky is at zfar and will be fully fogged/obscured by the water volume

- Particles and alpha render. If they have depth test enabled, they will clip against the surface
- Postprocessing runs with the postprocessing depth and colours

## 9 Performance

The foundation of *Crest* is architected for performance from the ground up with an innovative LOD system. It is tweaked to achieve a good balance between quality and performance in the general case, but getting the most out of the system requires tweaking the parameters for the particular use case. These are documented below.

### 9.1 Quality parameters

These are available for tweaking out of the box and should be explored on every project:

- See section 3.4 for parameters that directly control how much detail is in the ocean, and therefore the work required to update and render it. These are the primary quality settings from a performance point of view.
- The ocean shader has accrued a number of features and has become a reasonably heavy shader. Where possible these are on toggles and can be disabled, which will help the rendering cost (see section 3.2).

### 9.2 Potential optimisations

The following are optimisation ideas that have not been explored further in *Crest* yet, but are listed as ideas in case the reader has the resources to try them out.

- The ocean update runs as a command buffer, portions of which could potentially be ran asynchronously which may improve GPU utilisation. For example dynamic waves and Gerstner waves could be computed in parallel before the combine pass merges them together. We attempted this but found platform support is currently too limited to allow us to implement this in core *Crest*.
- Create multiple ocean materials with different sets of features and switch between them at run-time. An obvious use case would be one for shallow water and one for deep water and perhaps switch between them on a per-tile basis.
- Limit resolution range of LOD data. There is currently a min/max grid size option on the dynamic wave sim to limit what resolutions it runs at, this could be rolled out to other sim types.
- Set LOD data resolution per LOD type - e.g. the ocean depth and foam sim could potentially be stored at half resolution or lower.

- GPU-instance ocean material tiles. Discussed in issue #27 on the *Crest* GitHub. An idea we plan to pursue in the future is to perform height queries on the GPU in a compute shader, thus avoiding the need to transfer data back to the CPU at all. Not currently planned for *Crest*.
- Pre-rendered wave displacements - sample waves from texture instead of computing them on the fly. We did some initial experimentation in the branch *feature/baked-waves* and found it challenging to get good shape without interpolation artifacts. Given that the baking step is also inconvenient, there are no plans to explore this further.

## 10 Q&A

**Is *Crest* well suited for medium-to-low powered mobile devices?** *Crest* is built to be performant by design and has numerous quality/performance levers. However it is also built to be very flexible and powerful and as such can not compete with a minimal, mobile-centric ocean renderer such as the one in the *BoatAttack* project. Therefore we target *Crest* at PC/console platforms.

**Which platforms does *Crest* support?** Testing occurs primarily on Windows. We have users targeting Windows, Mac, Linux, PS4, XboxOne, Switch and iOS/Android. Performance is a challenge on Switch and mobile platforms - see the previous question. WebGL is unfortunately not supported as this platform currently does not support compute shaders which are required by *Crest*.

**Is *Crest* well suited for localised bodies of water such as lakes?** Currently *Crest* is currently targeted towards large bodies of water. The water could be pushed down where it's not wanted which would allow it to achieve rivers and lakes to some extent.

**Does *Crest* support third party sky assets?** We have heard of *Crest* users using TrueSky, AzureSky. These may require some code to be inserted into the ocean shader - there is a comment referring to this, search *Ocean.shader* for 'Azure'.

**Can *Crest* work in Edit mode in the Unity Editor, or only in Play mode?** Currently it only works in Play mode. Some work has been done to make it work in Edit mode but more work/fixes/testing is needed. <https://github.com/huwb/crest-oceanrender/issues/208>

**Can *Crest* work with multiplayer?** Yes the animated waves are deterministic and easily synchronized. See discussion in <https://github.com/huwb/crest-oceanrender/issues/75>. However, the dynamic wave sim is not fully deterministic and can not currently be relied upon in networked situations.