



**Crest**

***Release 4.15.1***

## **Wave Harmonic & Contributors**

**Mar 03, 2022**



# ABOUT

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Sponsorship . . . . .	1
1.2	Social . . . . .	1
<b>2</b>	<b>Known Issues</b>	<b>3</b>
2.1	Unity Bugs . . . . .	3
2.2	Unity Features . . . . .	3
<b>3</b>	<b>Roadmap</b>	<b>5</b>
<b>4</b>	<b>Release Notes</b>	<b>7</b>
<b>5</b>	<b>Getting Started</b>	<b>29</b>
5.1	Requirements . . . . .	29
5.2	Importing <i>Crest</i> files into project . . . . .	30
5.2.1	Pipeline Setup . . . . .	30
5.2.2	Importing <i>Crest</i> . . . . .	30
5.3	Adding <i>Crest</i> to a Scene . . . . .	32
5.4	Frequent Setup Issues . . . . .	32
5.4.1	Errors present, or visual issues . . . . .	32
5.4.2	Compile errors in the log, not possible to enter play mode, visual issues in the scene . . . . .	32
5.4.3	Possible to enter play mode, but errors appear in the log at runtime that mention missing ‘kernels’ . . . . .	33
5.4.4	Ocean framerate low in edit mode . . . . .	33
5.4.5	Ocean reflections/lighting/fog looks wrong <i>HDRP</i> . . . . .	33
<b>6</b>	<b>Configuration</b>	<b>35</b>
6.1	Material Parameters . . . . .	36
6.1.1	Normals . . . . .	36
6.1.2	Scattering . . . . .	36
6.1.3	Subsurface Scattering . . . . .	36
6.1.4	Shallow Scattering . . . . .	36
6.1.5	Reflection Environment . . . . .	37
6.1.6	Add Directional Light . . . . .	37
6.1.7	Procedural Skybox . . . . .	37
6.1.8	Foam . . . . .	38
6.1.9	Foam 3D Lighting . . . . .	38
6.1.10	Foam Bubbles . . . . .	38
6.1.11	Transparency . . . . .	38
6.1.12	Caustics . . . . .	38
6.1.13	Underwater . . . . .	39

6.1.14	Flow	39
6.2	Lighting	39
6.2.1	General	39
6.2.2	Reflections	40
6.2.3	Refractions	42
6.3	Orthographic Projection	42
6.4	Ocean Construction Parameters	42
6.5	Advanced Ocean Parameters	43
<b>7</b>	<b>Ocean Simulation</b>	<b>45</b>
7.1	Animated Waves	45
7.1.1	Overview	45
7.1.2	Simulation Settings	46
7.1.3	User Inputs	46
7.2	Dynamic Waves	47
7.2.1	Overview	47
7.2.2	Adding Interaction Forces	47
7.2.3	Simulation Settings	48
7.2.4	User Inputs	48
7.3	Foam	48
7.3.1	Overview	48
7.3.2	User Inputs	49
7.3.3	Simulation Settings	49
7.4	Sea Floor Depth	50
7.5	Clip Surface	50
7.5.1	Simulation Settings	51
7.5.2	User Inputs	51
7.6	Shadows	52
7.7	Flow	53
7.7.1	Overview	53
7.7.2	User Inputs	53
7.8	Albedo	53
7.8.1	Overview	54
7.8.2	User Inputs	54
<b>8</b>	<b>Wave Conditions</b>	<b>55</b>
8.1	Wave Systems	55
8.2	Authoring	55
8.3	Wave Splines	56
<b>9</b>	<b>Shorelines and Shallows</b>	<b>57</b>
9.1	Setup	57
9.2	Shoreline Waves	58
9.3	Troubleshooting	58
<b>10</b>	<b>Oceans, Rivers and Lakes</b>	<b>59</b>
10.1	Oceans	59
10.2	Lakes	59
10.3	Rivers	60
<b>11</b>	<b>Collision Shape for Physics</b>	<b>61</b>
11.1	Collision API Usage	61
11.2	Compute Shape Queries (GPU)	62
11.3	Baked FFT Data (CPU)	62
11.4	Gerstner Waves CPU (deprecated)	63

<b>12 Underwater</b>	<b>65</b>
12.1 Underwater Renderer	66
12.1.1 Setup	66
12.1.2 Parameters	66
12.1.3 Detecting Above or Below Water	67
12.1.4 Portals & Volumes	67
12.1.5 Underwater Shader API	67
12.2 Underwater Curtain <i>BIRP URP</i>	69
12.2.1 Setup	70
12.3 Underwater Post-Process <i>HDRP</i>	70
12.3.1 Setup	70
<b>13 Time Control</b>	<b>71</b>
13.1 Supporting Pause	71
13.2 Network Synchronisation	71
13.3 Timelines and Cutscenes	72
<b>14 Other Features</b>	<b>73</b>
14.1 Floating Origin	73
14.2 Buoyancy	73
14.2.1 Adding boats	73
<b>15 Rendering</b>	<b>75</b>
15.1 Transparency	75
15.1.1 Transparent Object In Front Of Ocean Surface	75
15.1.2 Transparent Object Behind The Ocean Surface	75
15.1.3 Transparent Object Underwater	75
15.2 Render Order <i>BIRP URP</i>	76
<b>16 Performance</b>	<b>77</b>
16.1 Quality parameters	77
<b>17 Technical Documentation</b>	<b>79</b>
17.1 Core Data Structure	79
17.2 Implementation Notes	82
<b>18 Q &amp; A</b>	<b>83</b>



## INTRODUCTION

*Crest* is a technically advanced ocean system for Unity.

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 has an intuitive and easy to use shape authoring interface.

This documentation is for *Crest* 4.15.1 and targets BIRP (Built-in Render Pipeline).

You can view this [documentation online](#).

This documentation is for *Crest* 4.15.1 and targets HDRP (High Definition Render Pipeline).

You can view this [documentation online](#).

This documentation is for *Crest* 4.15.1 and targets URP (Universal Render Pipeline).

You can view this [documentation online](#).

### 1.1 Sponsorship

---

#### Sponsor

Sponsor Wave Harmonic on [GitHub Sponsors](#) to increase development time on Crest.

Throughout the documentation, you will see sponsor admonitions like this one for features where only expanded funding can help cover development costs.

[Sponsor Us](#)

---

### 1.2 Social

- **YouTube** [https://www.youtube.com/channel/UC7\\_ZKKCXZmH64rRZqe-C0WA](https://www.youtube.com/channel/UC7_ZKKCXZmH64rRZqe-C0WA)
- **Twitter** [https://twitter.com/@crest\\_ocean](https://twitter.com/@crest_ocean)
- **Discord** <https://discord.gg/g7GpjDC>





## KNOWN ISSUES

We keep track of issues on GitHub for all pipelines. Please see the following links:

- [Issues on GitHub](#).
- [BIRP specific issues on GitHub](#).
- [HDRP specific issues on GitHub](#).
- [URP specific issues on GitHub](#).

If you discover a bug, please [open a bug report](#) or mention it on the [bugs channel on our Discord](#).

### 2.1 Unity Bugs

There are some Unity issues that affect *Crest*. Some of these may even be blocking new features from being developed. If you could vote on these issues, that would be greatly appreciated:

- [Gizmos render over opaque objects with Post-Processing stack](#). *BIRP*

### 2.2 Unity Features

There are upcoming features being developed by Unity which will greatly help *Crest*. If you could vote on these features, that would be greatly appreciated:

- [Post Processing Custom Effects](#) *URP*



## ROADMAP

---

### Sponsor

This will help us expand our roadmap and achieve goals sooner. Certain sponsor tiers allows one to vote on roadmap items to guide our priorities.

[Sponsor Us](#)

---

Visit [our roadmap](#) to see where Crest's development is heading:

[Trello Board](#)



## RELEASE NOTES

### 4.15.1

#### Fixed

- Fix shader compiler error.

### 4.15

#### Breaking

- Ocean inputs will now only execute the first shader pass (pass zero). Before all passes were executed in sequence which caused incompatibilities with URP unlit *Shader Graph*. This is only a concern to those who are using custom shaders with multiple passes which we believe is very few.

#### Preview

- Add new CPU-based collision provider - *Baked FFT Data*.
- Add portals and volumes to *Underwater Renderer* (affects both underwater and ocean surface). See [Portals & Volumes](#) for more information.
- Add *Shader API* to *Underwater Renderer* to facilitate adding underwater fog to transparent objects. See [Underwater Shader API](#) for more information.
- Add *Albedo Data* feature which allows layering colour onto the water surface similar to decals.

#### Changed

- Add new example scene named *Examples* which contains many mini examples of different features of *Crest*.
- Add new example scene named *LakesAndRivers* for adding lakes and rivers using splines.
- Add support for rendering in edit mode (camera preview and scene view) to *Underwater Renderer*. It can be enabled/disabled with the fog scene view toggle.
- Add *CREST\_OCEAN* scripting defines symbol.
- Add *Depth Fog Density Factor* to *Underwater Renderer* which can be used to decrease underwater fog intensity when underwater. Greatly improves shadows at shorelines.

- Add UV feathering option to Flow shaders.
- Add *Attenuation in Shallows* to *Dynamic Waves Sims Settings*.
- Add *Shallows Max Depth* to *Sim Settings Animated Waves* as an alternative to having to extend terrain to 500m below sea level to avoid discontinuity issues.
- Add *Allow No Shadows* to *Sim Settings Shadows* to allow shadows to be enabled/disabled dynamically.
- Add *Ocean Renderer > Water Body Culling* option so the ocean can ignore culling. Useful if using *Water Body > Override Material* and still want an ocean.
- Improve multiple *Water Body* overlapping case when *Water Body > Override Material* option is used.
- *Water Body* adds an inclusion to clipping (ie unclips) if *Default Clipping State* is *Everything Clipped*.
- Add *Underwater Renderer* support for *Water Body > Override Material*.
- Add scroll bar to *Ocean Debug GUI* when using *Draw LOD Datas Actual Size*.
- Add support for *TrailRenderer*, *LineRenderer* and *ParticleSystem* to be used as ocean inputs in addition to *MeshRenderer*.
- Un-deprecate *ShapeGerstner* as it is useful in some situations for adding a small number of distinct waves with high degree of control.
- Add *Reverse Wave Weight* setting to *ShapeGerstner* for fine control over generated wave pairs.
- Double sample count for *ShapeGerstner* waves to improve quality.
- Tidy up wave spectrum inspector by only showing *ShapeGerstner*-specific controls when editing within a *ShapeGerstner* component.
- Add option (enabled by default) to prewarm foam simulation on load and camera teleports.
- *Underwater Renderer* validates *Ocean Renderer* material.
- Add *Debug > Draw Queries* to *Boat Probes* to draw gizmos for queries.
- *SphereWaterInteraction* component upgraded to produce crisp foam-generating waves without creating large displacements. [#979](#)
- Add new example scene *BoatWakes* to showcase improvements to *SphereWaterInteraction* component.
- Allow scaling FFT waves on spline (not supported previously). *SplinePointDataGerstner* has been renamed to *SplinePointDataWaves* which works for both *ShapeFFT* and *ShapeGerstner*.
- Add *Surface Self-Intersection Fix Mode* (advanced option) to control how self-intersections of the ocean surface caused by intense/choppy waves are handled.
- Add *Maximum Buoyancy Force* for preventing objects from having too much force being applied when fully submerged.
- Updated all example scenes.
- Unity 2021.2 users can now use the Shader Graph version of the ocean shader. The generated shader is deprecated and should not be used as it does not work correctly for 2021.2. *HDRP*
- Add support for *Ray-Traced Reflections* for Unity 2021.2. *HDRP*
- Revert to using Unity's material inspector which gives more control and is more reliable. *HDRP*
- Improve ocean material inspector for Unity 2021.2. *HDRP*
- Caustics and foam textures now use the sampler defined on the texture asset. If using our caustics texture, it will now use trilinear sampling instead of linear. *HDRP*

- Add support for secondary lights like point or spot to ocean shader. Only supports pixel lights and not vertex lights. *URP*

## Fixed

- Fix incorrect baked depth cache data that were baked since *Crest 4.14*.
- Fix XR SPI (Single-Pass Instanced) underwater rendering for Unity 2021.2 standalone.
- Fix *Underwater Renderer* not rendering on *Intel iGPUs*.
- Fix clip surface inputs losing accuracy with large waves.
- Fix waves at shorelines being incorrectly shadowed. [#945](#)
- Fix shadow bleeding at shorelines by using the *Sea Floor Depth* data to reject invalid shadows. [#947](#)
- Fix exceptions thrown for server/headless builds.
- Fix exceptions thrown if foam, dynamic waves and shadows all were disabled.
- Fix *Floating Origin* for *Shape Gerstner* and *Shape FFT*.
- Fix ocean textures popping (normals, caustics etc) when *Floating Origin* teleports.
- Fix collision queries (eg buoyancy) popping when *Floating Origin* teleports.
- Fix ocean scale smoothing on first frame and teleports. This issue appears as the ocean detail being low and slowly becoming high detailed.
- Fix shadow data not always clearing.
- Fix shadow simulation not recovering after error being resolved in edit mode.
- Fix *Allow Null Light* option on *Sim Settings Shadows* not working.
- Fix ocean tiles not reverting to *Ocean Renderer > Material* if *Water Body > Override Material* was used and *Water Body* was disabled or removed.
- Add *Time Scale* control for FFT (*Gravity* setting was broken).
- Fix underwater rendering when the camera's culling mask excludes the *Ocean Renderer > Layer*.
- Fix visible "rings" in dynamic wave sim resulting from fast moving objects that have the *Sphere Water Interaction* component attached. Simulation frequency can be increased to improve result further, at the cost of more simulation steps per frame.
- Fix *Sphere Water Interaction* component not working in standalone builds.
- Fix pop/discontinuity issue with dynamic waves.
- Fix underwater culling when *Ocean Renderer > Viewpoint* is set and different from the camera.
- Fix several minor exceptions in cases where components were not set up correctly.
- Fix possible cases of underwater effect being inverted on self-intersecting waves when further than 2m from ocean surface.
- Fix a per frame GC allocation.
- Fix ocean input validation incorrectly reporting that there is no spline attached when game object is disabled.
- Fix *Shape FFT* with zero weight causing visible changes or pops to the ocean surface.
- Fix *Shape FFT* waves animating too quickly when two or more are in the scene with different resolutions.
- Fix *Shape Gerstner* weight not updating correctly if less than one on game load.

- Fix *Shape Gerstner* weight being applied twice instead of once. You may need to adjust your weight if between zero and one.
- Fix Unity 2021.2 script upgrade requirement.
- Fix compilation error if both HDRP and URP packages are installed.
- Fix shadow simulation null exceptions if primary light becomes null. *BIRP*
- Fix shadows flickering when *Sea Floor Depth* data is populated by preventing shadow passes from executing for *Ocean Depth Cache* camera. *BIRP*
- Fix *Underwater Renderer* using a non directional light when a transparent object is in range of light and in view of camera. *BIRP*
- Fix caustics not rendering if shadow data is disabled. *BIRP*
- Fix *Underwater Renderer* looking washed out due to using incorrect colour space for Unity 2021.2. *BIRP*
- Fix *Underwater Renderer* high memory usage by reverting change of using temporary render textures. *BIRP URP*
- Fix *Underwater Renderer* not using *Filter Ocean Data* for caustics. *BIRP URP*
- Fix ocean input incompatibilities with unlit *Shader Graph*. *URP*
- Fix possible “Extensions” class naming collision compilation error. *HDRP URP*
- Fix motion vectors not working by exposing motion vector toggle on ocean material. *HDRP*
- Fix foam bubbles parallax effect using the incorrect normal space. *HDRP*
- Fix foam bubbles texture scaling. *HDRP*

## Performance

- Reduce cost of populating the ocean depth cache. *HDRP*

## 4.14

### Changed

- Add *Dynamic Waves* reflections from *Ocean Depth Cache* geometry.
- Add inverted option to *Clip Surface* signed-distance primitives and convex hulls which removes clipping.
- Add *Override Material* field to the *Water Body* component to enable varying water material across water bodies.
- *Sphere Water Interaction* component simplified - no mesh renderer/shader setup required, and no ‘register’ component required.
- *Sphere Water Interaction* produces more consistent results at different radii/scales.
- Improve FFT (Fast Fourier Transform) wave quality by doubling the sampling from two to four.
- *RegisterHeightInput* can be used in conjunction with our *Spline* component to offset the water level. This can be used to create water bodies at different altitudes, and to create rivers that flow between them.
- All water features updated to support varying water level.
- Add buttons to *Spline* inspector to quickly enable water features.
- Exposed control over *Spline* ribbon alignment - spline points now define the center of the ribbon by default.



- Caustics no longer render in shadows casted from objects underwater.
- Added motion vectors (for TAA, DLSS and many screen-space effects). *HDRP*
- Added shadow distance fade to shadow data. *URP*
- Improve URP shadow settings validation. *URP*

## Fixed

- Fix lines in foam data producing noticeable repeating patterns when using FFT waves.
- Fix caustics jittering when far from zero and underwater in XR.
- Fix disabled simulations' data being at maximum when "Texture Quality" is not "Full Res". In one case this manifested as the entire ocean being shadowed in builds.
- Fix high CPU memory usage from underwater effect shader in builds.
- Fix FFT spectrum not being editable when time is paused.
- Fix *ShapeFFT* component producing inverted looking waves when enabled in editor play mode.
- Fix SSS colour missing or popping in the distance.
- Fix underwater artefacts (bright specks).
- Fix shadows for MacOS. *BIRP*
- Fix shadows for *Shadow Projection > Close Fit*. *BIRP*
- Fix shadows for deferred rendering path. *BIRP*
- Fix *Crest/Framework* shader compiler errors for 2021.2. *URP*
- Fix "xrRendering" build error. *URP*
- Fix *Default Clipping State > Everything Clipped* not clipping extents. *HDRP*
- Fix Ocean shader compilation errors for HDRP 10.7. *HDRP*

## Removed

- Remove *Texels Per Wave* parameter from Ocean Renderer and hard-code to Nyquist limit as it is required for FFTs to work well.
- Removed *Create Water Body* wizard window. The water body setup has been simplified and works without this additional tooling.
- *Smoothing* feature removed from *Spline*, underlying code made more robust.
- Remove *Assign Layer* component.

## Performance

- Only calculate inverse view projection matrix when required.
- Reduce shader variants by removing GPU instancing (not supported currently).
- Reduce shadow simulation GPU performance cost by almost 50%. *BIRP HDRP*
- Improve *Underwater Renderer* GPU memory usage. *BIRP URP*
- Reduce ocean shader GPU performance cost for shadows. *HDRP*

## Deprecated

- Made *ObjectWaterInteraction* component obsolete, this is replaced by the more simple and robust *SphereWaterInteraction*. Removed usages of this component from the example scenes.
- Made *ShapeGerstner* and *ShapeGerstnerBatched* components obsolete as they are replaced by the *ShapeFFT* component. Example scenes moved over to *ShapeFFT*.

## 4.13

### Changed

- Add signed-distance primitives for more accurate clipping and overlapping. See [Clip Surface](#) for more information.
- Add *Render Texture Graphics Format* option to *Clip Surface Sim Settings* to support even more accurate clipping for signed-distance primitives.
- Add *Render Texture Graphics Format* option to *Animated Waves Sim Settings* to solve precision issues when using height inputs.
- Always report displacement in *Register Height Input* to solve culling issues.
- Add default textures to ocean shader.
- Update ocean shader default values.
- Improve foam detail at medium to long distance.
- Add *Scale By Factor* shader for all inputs which is particularly useful when used with *Animated Waves* for reducing waves.
- Add a simpler custom material inspector. *HDRP*
- Add XR SPI support to *Underwater Renderer*. *URP*

## Fixed

- Fix ocean not rendering on Xbox One and Xbox Series X.
- Fix height input (and others) from not working 100m above sea level and 500m below sea level.
- Fix FFT shader build errors for Game Core platforms.
- Fix FFT material allocations every frame.
- Fix flow simulation sometimes not clearing after disabling last input.
- Fix outline around objects when MSAA is enabled by making it less noticeable.
- Fix pixelated looking foam bubbles at medium to long distance.
- Fix underwater effect undershooting or overshooting ocean surface when XR camera is nearly aligned with horizon.
- Fix underwater effect being flipped at certain camera orientations.
- Fix meniscus thickness consistency (in some cases disappearing) with different camera orientations.
- Fix inputs (eg keyboard) working when game view is not focused.
- Fix *Ocean Depth Cache* disabling itself in edit mode when no ocean is present.
- Fix ocean disappearing when viewed from an area clipped by a clip surface input. *HDRP*
- Fix shadows breaking builds when XR package is present. *HDRP*
- Fix shadows not working with XR SPI. *HDRP*
- Fix 2021.2.0b9 shader compile errors. *HDRP*
- Fix ocean material properties missing for 2021.2 material inspector. *HDRP*
- Fix outline around refracted objects by making it less noticeable. *HDRP*
- Fix *Underwater Renderer* caustics jittering for some XR devices. *BIRP URP*
- Fix shadow artefacts when no shadow casters are within view. *URP*
- Remove sample shadow scriptable render feature error. *URP*

## 4.12

### Breaking

- Set minimum Unity version to 2020.3.10.
- Set minimum render pipeline package version to 10.5. *HDRP URP*
- *Underwater Post-Processing* is disabled by default which means it will be inactive if the *Underwater Volume Override* is not present in the scene. *HDRP*
- Remove *Sample Shadows* Render Feature as it is now scripted. Unity will raise a missing Render Feature reference error. Remove the missing Render Feature to resolve. *URP*

## Changed

- Add new *Underwater Renderer* component which executes a fullscreen pass between transparent and post-processing pass. Please see [Underwater](#) for more information.
- FFT generator count added to debug GUI.
- *ShapeFFT* component allows smooth changing of wind direction everywhere in world.
- Default *Wind Speed* setting on *OceanRenderer* component to 10m/s.
- *CustomTimeProvider* override time/delta time functions are now defaulted to opt-in instead of opt-out.
- Improve meniscus rendering by also rendering below ocean surface line. *HDRP*

## Fixed

- Fix case where normal could be NaN, which could make screen flash black in HDRP.
- Fix *ShapeFFT Spectrum Fixed At Runtime* option not working.
- Fix shader compile errors on Windows 7.
- Fix ocean depth cache shader compile error.
- Fix ocean not rendering on *Unity Cloud Build* (unconfirmed).
- Fix *ShapeGerstner* and *ShapeFFT* having no default spectrum in builds.
- Fix “missing custom editor” error for *Whirlpool* component.
- Fix ocean breaking after leaving a prefab scene.
- Fix underwater breaking for XR SPI. *HDRP*
- Fix underwater artefacts for XR MP (Multi-Pass). *HDRP*
- Fix meniscus rendering incorrectly when camera is rotated. *HDRP*

## Performance

- FFT wave generation factored out so that multiple *ShapeFFT* components sharing the same settings will only run one FFT.
- Underwater ocean mask now deactivates when the underwater effect is not active. *HDRP*

## Deprecated

- The *Underwater Effect* component (including *UnderWaterCurtainGeom.prefab* and *UnderWaterMeniscus.prefab*) has been superseded by the *Underwater Renderer*. Please see [Underwater](#) for more information. *BIRP URP*
- The *Underwater Post-Process* effect has been superseded by the *Underwater Renderer*. Please see [Underwater](#) for more information. *HDRP*

## 4.11

---

**Important:** This will be the last version which supports Unity 2019 LTS.

Spectrum data will be upgraded in this version. Due to a unity bug, in some rare cases upgrading the spectrum may fail and waves will be too large. Restart Unity to restore the spectrum.

---

### Preview

- FFT wave simulation added via new ShapeFFT component.

### Changed

- Sponsorship page launched! Asset Store sales only cover fixes and basic support. To support new feature development and give us financial stability please consider sponsoring us, no amount is too small! <https://github.com/sponsors/wave-harmonic>
- Wind speed added to OceanRenderer component so that wave conditions change naturally for different wind conditions.
- Empirical spectra retweaked and use the aforementioned wind speed.
- Add Overall Normals Scale parameter to material that scales final surface normal (includes both normal map and wave simulation normal).
- Headless support - add support for running without display, with new toggle on OceanRenderer to emulate it in Editor.
- No GPU support - add support for running without GPU, with new toggle on OceanRenderer to emulate it in Editor.
- OceanRenderer usability - system automatically rebuilds when changing settings on the component, 'Rebuild' button removed.
- Ocean material can now be set with scripting.
- Custom Time Provider has pause toggle, for easy pausing functionality.
- Network Time Provider added to easily sync water simulation to server time.
- CutsScene Time Provider added to drive water simulation time from Timelines.
- Made many fields scriptable (public) on *BoatProbes*, *BoatAlignNormal* and *SimpleFloatingObject*.
- Tweaked colours and some of properties for *Ocean-Underwater* material. *BIRP URP*
- *Copy Ocean Material Params Each Frame* is now enabled by default for *Underwater Post Process*. *HDRP*
- Add *Refractive Index of Water* property to ocean material. *HDRP*

## Fixed

- Fix build errors for platforms that do not support XR/VR.
- Fix “black square” bug on Oculus Quest.
- Fix for bugs where a large boat may stop moving when camera is close.
- Fix bad data being sampled from simulations when they’re not enabled like the entire ocean being shadowed when shadow data was disabled.
- Fix null exception for attach renderer help box fix button.
- Fix “remove renderer” help box not showing when it should.
- Fix bug where wind direction could not be set per ShapeGerstner component.
- Fix compilation errors when only Unity’s new *Input System* backend is available.
- Fix null exceptions in validation when *OceanRenderer* is not present.
- Fix incorrect validation showing in prefab mode.
- Fix shadow data for XR/VR SPI from working and breaking builds. *HDRP*
- Fix underwater effect from breaking after all cameras being disabled. *HDRP*
- Fix ocean tiles disappearing when far from zero. *URP*

## Removed

- Remove Phillips and JONSWAP spectrum model options.

## Deprecated

- *Layer Name* on the *Ocean Renderer* has been deprecated. Use *Layer* instead.
- The *Refractive Index of Air* on the ocean material will be removed in a future version. *BIRP URP*

## Documentation

- Document issues with transparency in new *Rendering* page.
- Improve *Lighting* section.

## 4.10

### Changed

- Set minimum Unity version to 2019.4.24.
- Spline can now be used with any ocean input type, so can be used to set water level, add flow, and more.
- System for tweaking data on spline points such as flow speed.
- *RegisterHeightInput* component added for a clearer way to change water height (can be used instead of *RegisterAnimWavesInput*).

- More validation help boxes added to catch a wider range of setup issues.
- Fix buttons in help boxes now describe action that will be taken.
- Rename *Add Water Height From Geometry* to *Set Base Water Height Using Geometry*.
- Rename *Set Water Height To Geometry* to *Set Water Height Using Geometry*.
- Improved spline gizmo line drawing to highlight selected spline point.
- Add version and render pipeline to help button documentation links.
- Validate scene view effects toggle options.
- Add various fix buttons for depth cache issues.
- Set minimum render pipeline package version to 7.6 which is correct for 2019.4. *HDRP URP*
- Rearrange some material properties. *HDRP*

## Fixed

- Fix water body creation not being part of undo/redo history.
- Fix spline point delete not being part of undo/redo history.
- Fix validation fix buttons that attach components not being part of undo/redo history.
- Fix ShapeGerstnerBatched not having default spectrum when using “Reset” and correct undo/redo history.
- Fix properties with embedded asset editors appearing broken for Unity 2020 and 2021.
- Fix shader compilation errors for HDRP 10.4. *HDRP*
- Remove duplicate foam bubble properties. *HDRP*
- New horizon line bug fix which is enabled by default (with option to switch back to old safety margin). *HDRP*

## Documentation

- Add *Detecting Above or Below Water* and have Q&A question refer to it.
- Add *Known Issues* page.
- Document *Caustics Distortion Texture*. *HDRP*
- Fixed Underwater *Setup* not being complete. *HDRP*
- Fix broken Unity documentation links by correctly setting minimum render pipeline version. *HDRP URP*

## 4.9

### Breaking

- Dynamic Waves and Foam simulations now run at configurable fixed timesteps for consistency across different frame rates. Tweaking of settings may be required. See [#778](#) for more details.
- Change *Layer Names* (string array) to *Layers* (LayerMask) on *Ocean Depth Cache*.

## Preview

- Add wizard for creating local water bodies. See *Oceans, Rivers and Lakes*.

## Changed

- Add [online documentation](#).
- Set up help button linking to new documentation for multiple components, and added material help button.
- Add inline editing for sim settings, wave spectrums and ocean material.
- Add *Crest* icons to sim settings and wave spectrums.
- Add button to fix issues on some validation help boxes.
- Add validation to inform whether the depth cache is outdated.
- Add validation for ocean depth cache with non uniform scale.
- Add scriptable custom time provider property which accepts interfaces.
- Validate simulation checkboxes and their respective material checkboxes and inputs.
- Add “*Crest*” prefix to component menu items.
- Organise “*Crest*” component menu items into subfolders.

## Fixed

- Fix more cases of fine gaps.
- Fix depth cache not reflecting updated properties when populating cache.
- Fix RayTraceHelper not working.
- Fix ShapeGerstner component breaking builds.
- Fix PS4/PSSL shader errors.
- Fix local waves flickering in some cases.
- Fix VFACE breaking shaders on consoles.
- Fix underwater normals incorrect orientation. *HDRP*
- Fix shader errors for latest consoles. *HDRP*
- Fix gray ocean by forcing depth and opaque texture when needed in the editor. *URP*
- Only feather foam at shoreline if transparency is enabled. *URP*



## Deprecated

- *Assign Layer* component is no longer used in examples and will be removed.

## 4.8

### Preview

- Add new Gerstner component *ShapeGerstner* with better performance, improved foam at a distance, correct wave direction and spline support (preview). See notes in the *Wave conditions* section of the user guide.
- Add new spline tool component *Spline* which can be wave splines for new gerstner system (preview). See notes in the *Wave conditions* section of the user guide.

### Changed

- Change minimum Unity version to 2019.4.9
- Add orthographic projection support to ocean surface
- Add weight control for *Underwater Environmental Lighting* component
- Calculate sub-surface light scattering from surface pinch, to enable other fixes/improvements. May require retweaking of the scattering settings on the ocean material.
- Improve error reporting when compute shaders fail
- Change shader level target for combine shader to 3.5 which might fix some issues on Quest

### Fixed

- Fix dynamic wave sim stability by reducing *Courant number* default value
- Remove warning when camera not set which was displaying even when it shouldn't
- Change ocean depth cache populate event option to Start
- Fix for multiple gaps/cracks in ocean surface bugs
- Fix *Follow Horizontal Motion* for foam override
- Fix normals not being flipped for underwater with flow enabled
- Fix meniscus shader not being enabled *HDRP*
- Fix ocean depth cache triggered by other cameras or probes *URP*
- Fix underwater effect flickering when other cameras are in the scene *URP*

## Performance

- Add option on *AnimWaveSimSetting* to disable ping pong for combine pass. See notes in performance section of user guide.

## 4.7

### Changed

- Add foam override shader and material to remove foam
- Add camera property to *OceanRenderer*. *ViewerHeightAboveWater* will use camera transform
- Add option to add downhill force to buoyancy for some floating objects
- Disable underwater culling if underwater effect is not used *HDRP*
- Underwater effect uses stencil buffer instead of depth buffer again *HDRP*

### Fixed

- Improve platform support by improving texture compatibility checks
- Fix Unity 2020.2 / RP 10 support
- Fix shadows not following scene view camera
- Fix *Follow Horizontal Motion* not working
- Fix *Strength* on *Crest/Inputs/Foam/Add From Texture* being ignored
- Query system - fixed ring buffer exhausted error on some Linux and Android platforms
- Fix shadow data breaking gizmos and GUI *HDRP*
- Fix underwater copy ocean material parameters option not working correctly when unchecked *HDRP*
- Fix underwater anti-aliasing artefacts around objects (HDRP 10+ required. See underwater documentation) *HDRP*

## Performance

- Minor underwater performance improvement
- Improve underwater XR multi-pass support (still not 100%) *HDRP*
- Improve underwater XR single pass instance performance *HDRP*
- Improve underwater performance when using dynamic scaling *HDRP*

## 4.6

### Changed

- Change minimum Unity version to 2019.4.8
- Improve foam texture
- Add height component that uses *UnityEvents* (under examples)
- Add shadow LOD data inputs
- Add support for disable scene reloading
- Add more dynamic waves debug reporting options
- Disable horizontal motion correction on animated waves inputs by default
- Make some shader parameters globally available
- Add reflections to ocean surface underside from water volume *HDRP*

### Fixed

- Fix precision artefacts in waves for mobile devices when far away from world centre
- Fix spectrum editor not working in play mode with time freeze
- Fix build error
- Fix *UnderwaterEnvironmentalLighting* component restoring un-initialised values
- Fix precision issues causing very fine gaps in ocean surface
- Fix some memory leaks in edit mode
- Fix mesh for underwater effects casting shadow in some projects *URP*
- Fix caustics moving, rotating or warping with camera for URP 7.4+ *URP*
- Fix caustics breaking for VR/XR SPI *URP*
- Fix underwater material from breaking on project load or recompile *URP*
- Fix underwater surface colour being added to transparent parts of ocean surface when underwater *HDRP*
- Fix sample height warning for XR multi-pass *HDRP*
- Fix underwater caustics not working in build due to stripping *HDRP*
- Fix shadows breaking VR/XR single pass instanced *HDRP*
- Fix deprecated XR API call warning *HDRP*
- Fix underwater breaking camera when ocean is disabled during run-time *HDRP*
- Fix ocean falloff parameters allowing bad values *HDRP*

## Performance

- Improve performance by reducing work done on scripted shader parameters every frame

## 4.5

### Changed

- Add option to ocean input to allow it to move with ocean surface horizontally (was always on in last version)
- Allow save depth cache to file in edit mode
- Remove ocean depth cache updating every frame in edit mode
- Improve feedback in builds when spectrum is invalid
- Improve spectrum inspector
- Validate OceanRenderer transform component
- Validate enter play mode settings
- Add soft/volume shadows support *HDRP*
- Add light/shadow layer support *HDRP*
- Remove caustics strength scaling by sun light and sea depth *HDRP*
- Add option to clip ocean surface under terrain *URP*
- Use local shader keywords *URP*

### Fixed

- Fix undo/redo for spectrum inspector
- Fix dynamic waves crashing when flow or depth sim not enabled
- Fix culling issues with turbulent waves
- Fix precision issues causing gaps in ocean surface
- Fix shadow sampling not following camera after changing viewpoint
- Fix shadow sampling not following scene camera
- Fix caustics and shadows not being correctly aligned
- Fix material being allocated every frame in edit mode
- Fix underwater effect for MSAA *HDRP*
- Fix many cases where gaps would appear with underwater effect *HDRP*
- Fix underwater effect rendering at top of viewport in certain cases *HDRP*
- Fix shader errors for HDRP 8.2 *HDRP*
- Fix underwater effects for URP 7.4+ *URP*

## 4.4

### Changed

- Gerstner waves from geometry shader - allow wave scaling using vertex colour
- Usability: disable inactive fields on ocean components in Inspector
- Validation: improve lighting settings validation
- XR: add single pass instanced support to underwater effects *HDRP*
- XR: add Single Pass Instanced support *URP*

### Fixed

- Fix for buffer overrun in height query system which caused crashes on Metal
- Fix for height query system breaking down at high frame rates when queries made from FixedUpdate
- Fix height queries when Scene Reload is disabled
- Fix various null reference exceptions in edit mode
- Fix for small wavelengths that could never be disabled
- Fix popping caused by shallow subsurface scattering colour
- Fix some null exceptions if OceanRenderer is not enabled in scene
- Fix mode (Global/Geometry) not applying in edit mode for ShapeGerstnerBatched component
- Clean up validation logging to console when a component is added in edit mode
- Fix global keywords not being local in underwater shader *HDRP*
- Fix ocean material keywords not applying to underwater *HDRP*
- Fix underwater breaking when dynamic scaling is used *HDRP*
- Fix caustics occasionally appearing on underside of surface *HDRP*
- Fix caustics briefly being too intense when switching cameras with adaptive exposure *HDRP*
- Fix indirect lighting controller multipliers not being applied *HDRP*
- Fix primary light intensity not reducing when primary light goes below the horizon *HDRP*
- Fix null exceptions when primary light is unset *HDRP*
- Fix underwater shader/material breaking on project load *URP*
- Fix shadow sampling running on cameras which isn't the main camera *URP*

## Performance

- Fix for ocean depth cache populating every frame erroneously

## 4.3

---

**Important: Crest LWRP deprecated.** We are no longer able to support LWRP, and have removed the LWRP version of Crest in this release. Do not install this version if you need to remain on LWRP.

---

## Changed

- Ocean now runs in edit mode
- Realtime validation in the form of inspector help boxes
- Add Submarine example scene created by the Digital Wizards team (Aldana Zanetta and Fernando Zanetta). *HDRP*
- Make compatible with dynamic batching *URP*
- Add option to disable occlusion culling in planar reflections to fix flickering (disabled by default) *URP*

## Fixed

- Fix *Segment registrar scratch exhausted* error that could appear in editor
- Fix underwater effect rendering when using baked occlusion culling *HDRP*
- Fix gaps appearing in underwater effect for very turbulent water *HDRP*
- Fix underwater raising exception when switching cameras *HDRP*
- Fix caustics rendering short of ocean surface when underwater *HDRP*

## 4.2

## Changed

- Scale caustics intensity by lighting, depth fog density and depth.
- Show proxy plane in edit mode to visualise sea level.
- Validate ocean input shader, warn if wrong input type used.
- Warn if SampleHeightHelper reused multiple times in a frame.
- Clamp reflection ray to horizon to avoid picking up below-horizon colours. *HDRP*
- Use sampler settings for normal map textures to allow changing filtering settings. Turned on anisotropic sampling to reduce blurring. *HDRP*

## Fixed

- Fix leaked height query GUIDs which could generate ‘too many GUIDs’ error after some time.
- Fix for cracks that could appear between ocean tiles.
- Fix for null ref exception in SRP version verification.
- Metal - fix shader error messages in some circumstances.
- Fix for erroneous water motion if Flow option enabled on material but no Flow simulation present.
- Fix sea floor depth being in incorrect state when disabled.
- Fix for a few cases where a crack or line is visible at the horizon. *HDRP*
- Fix for caustics showing above surface. *HDRP*
- Fix foam normals which were not working. *HDRP*
- Fix caustics stereo rendering for single-pass VR *URP*

## 4.1

### Changed

- Clip surface shader - add convex hull support
- Add support for local patch of Gerstner waves, demonstrated by `GameObject GerstnerPatch` in *boat.unity*
- Darkening of the environment lighting underwater due to out-scattering is now done with scripting. See the *UnderwaterEnvironmentalLighting* component on the camera in *main.unity*.
- Remove object-water interaction weight parameter on script. Use strength on material instead.
- Automatically pick the *sun* light if no *Primary Light* is specified. *HDRP*
- Bump version to 4.1 to match versioning with *Crest HDRP*. *URP*

### Fixed

- Fix garbage allocations.
- Fix PS4 compile errors.
- Multiple fixes to height query code that could produce ‘flat water’ issues or use incorrect wave data.
- Better retention of foam on water surface under camera motion.
- Fix flow not affecting displaced waves. *HDRP*
- Fix flow not working in *Whirlpool* example scene in standalone builds. *HDRP*
- Fixed caustics effect when underwater and added distortion. *HDRP*

## 4.0 HDRP

- First release!

## 3.8 URP

### Changed

- Refactor: Move example content into prefabs to allow sharing between multiple variants of Crest

### Fixed

- Fix for missing shadergraph subgraph used in test/development shaders. This does not affect main functionality but fixes import errors.

## 3.7 URP

### Changed

- Clip surface shader - replaces the ocean depth mask which is now deprecated
- Exposed maximum height query count in *Animated Wave Settings*
- Support disabling *Domain Reload* in 2019.3 for fast iteration

### Deprecated

- Ocean depth mask - replaced by clip surface shader

### Removed

- Removed the deprecated GPU readback system for getting wave heights on CPU

## 3.6 URP

### Changed

- Third party notices added to meet license requirements. See *thirdpartynotices.md* in the package root.



## 3.5 URP

### Changed

- Gizmos - color coded wireframe rendering of geometry for ocean inputs
- Object-water interaction: ‘adaptor’ component so that interaction can be used without a ‘boat’. See *AnimatedObject* object in *boat.unity*.
- Object-water interaction: new script to generate dynamic waves from spheres, which can be composed together. See *Spinner* object in *boat.unity*.
- Input shader for flowmap textures
- Better validation of depth caches to catch issues
- Documentation - link to new tutorial video about creating ocean inputs

### Fixed

- VR refraction fix - ocean transparency now works in VR using *Single Pass* mode.
- Fix visual pop bug at background/horizon when viewer gains altitude
- Fix for compile errors for some ocean input shaders

## 3.4 URP

### Changed

- Ocean depth cache supports saving cache to texture on disk
- Ray trace helper for ray queries against water
- Input shader for flowmaps
- Shader code misc refactors and cleanup

### Fixed

- Fix for dynamic wave sim compute shader not compiling on iOS

## 3.3 URP

### Fixed

- Fix for compute-based height queries which would return wrong results under some circumstances (visible when using Visualise Collision Area script)
- VR: Fix case where sea floor depth cache was not populated
- VR: Fix case where ocean planar reflections broken

## 3.2 URP

### Changed

- Add links to recently published videos to documentation
- Asmdef files added to make Crest compilation self-contained
- Documentation - strategy for configuring dynamic wave simulation
- Documentation - dedicated, fleshed out section for shallow water and shoreline foam
- Documentation - technical information about render/draw order

### Fixed

- Fixes for wave shape and underwater curtain on Vulkan
- Fix for user input to animated wave shape, add to shape now works correctly
- Fix for underwater appearing off-colour in standalone builds
- Fix garbage generated by planar reflections script
- Fix for invalid sampling data error for height queries
- Fix for underwater effect not working in secondary cameras
- Fix waves not working on some GPUs and Quest VR - [#279](#)
- Fix planar reflections not lining up with visuals for different aspect ratios

## 3.1 URP

### Changed

- Preview 1 of Crest URP - package uploaded for Unity 2019.3

### Fixed

- Made more robust against VR screen depth bug, resolves odd shapes appearing on surface
- [#279](#)

## GETTING STARTED

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

**Warning:** When changing Unity versions, setting up a render pipeline or making changes to packages, the project can appear to break. This may manifest as spurious errors in the log, no ocean rendering, magenta materials, scripts unassigned in example scenes, etcetera. 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.

To augment / complement this written documentation we published a video available here:

BIRP

<https://www.youtube.com/watch?v=qsgeG4sSLFw>

Fig. 5.1: Getting Start with Crest for BIRP

HDRP

<https://www.youtube.com/watch?v=FE6l39Lt3js>

Fig. 5.2: Getting Start with Crest for HDRP

URP

[https://www.youtube.com/watch?v=TpJf13d\\_-3E](https://www.youtube.com/watch?v=TpJf13d_-3E)

Fig. 5.3: Getting Start with Crest for URP

### 5.1 Requirements

- Unity Version: 2020.3.10
- Shader compilation target 4.5 or above
- Crest does not support OpenGL or WebGL backends

BIRP

- The *Crest* example content uses the post-processing package (for aesthetic reasons). If this is not present in your project, you will see an unassigned script warning which you can fix by removing the offending script.

HDRP

- The minimum HDRP package version is 10.5

URP

- The minimum URP package version is 10.5

## 5.2 Importing *Crest* files into project

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

### 5.2.1 Pipeline Setup

BIRP

Ensure that BIRP is setup and functioning, either by setting up a new project using the BIRP template or by configuring your current project. This is beyond the scope of this documentation so please see the [Unity documentation](#) for more information.

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. Please see the [Unity documentation](#) for more information.

HDRP

Ensure that HDRP is setup and functioning, either by setting up a new project using the HDRP template or by configuring your current project. This is beyond the scope of this documentation so please see the [Unity documentation](#) for more information.

URP

Ensure that URP is setup and functioning, either by setting up a new project using the URP template or by configuring your current project. This is beyond the scope of this documentation so please see the [Unity documentation](#) for more information.

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. Please see the [Unity documentation](#) for more information.

### 5.2.2 Importing *Crest*

Import the *Crest* package into project using the *Asset Store* window in the Unity Editor.

---

**Note:** The files under *Crest-Examples* are not required by our core functionality, but are provided for illustrative purposes. We recommend first time users import them as they may provide useful guidance.

---

BIRP

Import *Crest* assets by either:

- Currently we do not prepare release packages. However, we do tag each asset store version, so the zip corresponding to each version can be downloaded by clicking the desired version on the [Releases page](#).
- Getting latest by either cloning this repository or [downloading it as a zip](#), and copying the *Crest/Assets/Crest/Crest* folder into your project. Be sure to always copy the .meta files.

---

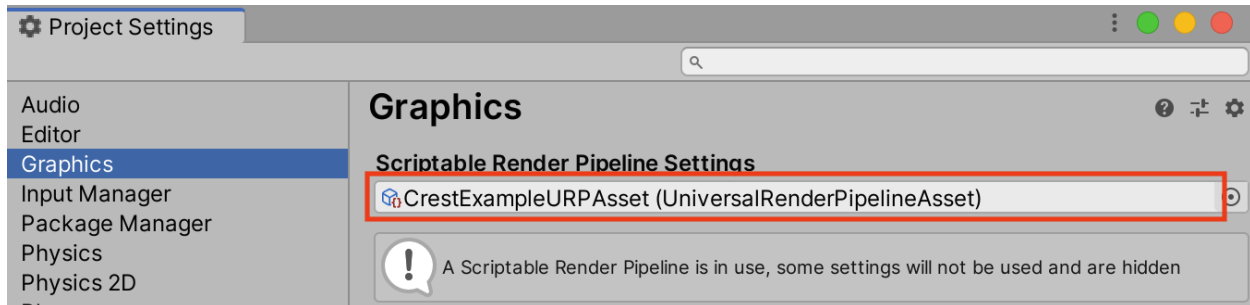
**Note:** The *Crest/Assets/Crest/Development* folder is only used to develop *Crest* and should be skipped.

---

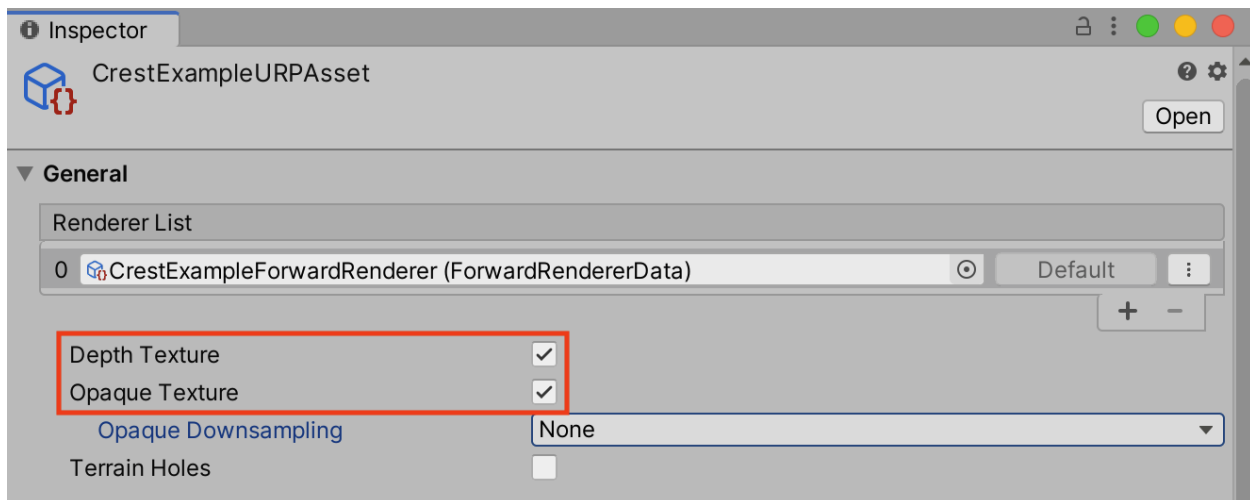
URP

## Transparency

To enable the water surface to be transparent, two options must be enabled in the URP configuration. To find the configuration, open *Edit/Project Settings/Graphics* and double click the *Scriptable Render Pipeline Settings* field to open the render pipeline settings. This field will be populated if URP was successfully installed.



After double clicking the graphics settings should appear in the Inspector. Transparency requires the following two options to be enabled, *Depth Texture* and *Opaque Texture*:




---

**Note:** If you are using the underwater effect, it is recommended to set *Opaque Downsampling* to *None*. *Opaque Downsampling* will make everything appear at a lower resolution when underwater. Be sure to test to see if recommendation is suitable for your project.

---

Read [Unity's documentation on the URP Asset](#) for more information on these options.

---

**Tip:** If you are starting from scratch we recommend [creating a project using a template in the Unity Hub](#).

---

## 5.3 Adding *Crest* to a Scene

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

- Create a new *GameObject* for the ocean, give it a descriptive name such as *Ocean*.
  - Assign the *OceanRenderer* component to it. This component will generate the ocean geometry and do all required initialisation.
  - Assign the desired ocean material to the *OceanRenderer* script - this is a material using the *Crest/Ocean* shader.
  - Set the Y coordinate of the position to the desired sea level.
- Tag a primary camera as *MainCamera* if one is not tagged already, or provide the *Camera* to the *View Camera* property on the *OceanRenderer* script. If you need to switch between multiple cameras, update the *ViewCamera* field to ensure the ocean follows the correct view.
- Be sure to generate lighting if necessary. The ocean lighting takes the ambient intensity from the baked spherical harmonics. It can be found at the following:

*Window* → *Rendering* → *Lighting Settings* → *Debug Settings* → *Generate Lighting*

---

**Tip:** You can check *Auto Generate* to ensure lighting is always generated.

---

- To add waves, create a new *GameObject* and add the *Shape FFT* component. See *Wave Conditions* section for customisation.
- Any ocean seabed geometry needs set up to register it with *Crest*. See section *Shorelines and Shallows*.
- If the camera needs to go underwater, the underwater effect must be configured. See section *Underwater* for instructions.

## 5.4 Frequent Setup Issues

The following are kinks or bugs with the install process which come up frequently.

### 5.4.1 Errors present, or visual issues

Try restarting Unity as a first step.

### 5.4.2 Compile errors in the log, not possible to enter play mode, visual issues in the scene

Verify that render pipeline is installed and enabled in the settings. See the follow for documentation:

Upgrading to HDRP

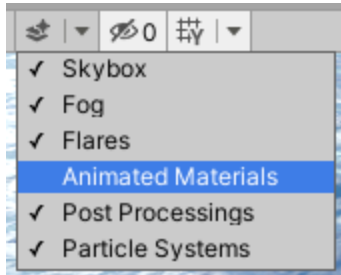
Installing URP into a project

### 5.4.3 Possible to enter play mode, but errors appear in the log at runtime that mention missing 'kernels'

Recent versions of Unity have a bug that makes shader import unreliable. Please try reimporting the *Crest/Shaders* folder using the right click menu in the project view. Or simply close Unity, delete the Library folder and restart which will trigger everything to reimport.

### 5.4.4 Ocean framerate low in edit mode

By default, the update speed is intentionally throttled by Unity to save power when in edit mode. To enable real-time update, enable *Animated Materials* in the Scene View toggles:



See the [Unity Documentation](#) for more information.

### 5.4.5 Ocean reflections/lighting/fog looks wrong *HDRP*

If reflections appear wrong, it can be useful to make a simple test shadergraph with our water normal map applied to it, to compare results. We provide a simple test shadergraph for debugging purposes - enable the *Apply test material* debug option on the *OceanRenderer* component to apply it. If you find you are getting good results with a test shadergraph but not with our ocean shader, please report this to us.





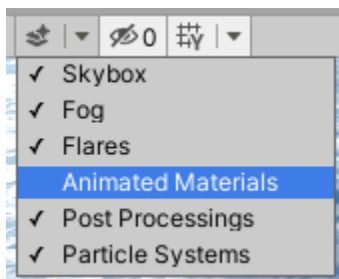
## CONFIGURATION

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

- Ocean surface appearance: The active ocean material is displayed below the *OceanRenderer* component. The material parameters are described in section [Material Parameters](#). Turn off unnecessary features to maximize performance.
- Animated waves / ocean shape: Configured on the *ShapeFFT* 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. See section [Wave Conditions](#).
- Shallow water: Any ocean seabed geometry needs set up to register it with *Crest*. See section [Shorelines and Shallows](#).
- Ocean foam: Configured on the *OceanRenderer* script by providing a *Sim Settings Foam* asset.
- Underwater: If the camera needs to go underwater, the underwater effect must be configured. See section [Underwater](#) for instructions.
- Dynamic wave simulation: Simulates dynamic effects like object-water interaction. Configured on the *OceanRenderer* script by providing a *Sim Settings Wave* asset, described in section [Simulation Settings](#).
- 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.

**Tip:** By default, the update speed is intentionally throttled by Unity to save power when in edit mode. To enable real-time update, enable *Animated Materials* in the Scene View toggles:



See the [Unity Documentation](#) for more information.

## 6.1 Material Parameters

### 6.1.1 Normals

**Overall Normal Strength** Strength of the final surface normal (includes both wave normal and normal map)

**Use Normal Map** Whether to add normal detail from a texture. Can be used to add visual detail to the water surface  
*BIRP URP*

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

**Normal Map Scale** Scale of normal map texture

**Normal Map Strength** Strength of normal map influence

### 6.1.2 Scattering

**Scatter Colour Base** Base colour when looking straight down into water.

**Scatter Colour Grazing** Base colour when looking into water at shallow/grazing angle. *BIRP URP*

**Enable Shadowing** Changes colour in shadow. Requires ‘Create Shadow Data’ enabled on OceanRenderer script.  
*BIRP URP*

**Scatter Colour Shadow** Base colour in shadow. Requires ‘Create Shadow Data’ enabled on OceanRenderer script.

### 6.1.3 Subsurface Scattering

**Enable** Whether to emulate light scattering through the water volume. *BIRP URP*

**SSS Tint** Colour tint for primary light contribution.

**SSS Intensity Base** Amount of primary light contribution that always comes in.

**SSS Intensity Sun** Primary light contribution in direction of light to emulate light passing through waves.

**SSS Sun Falloff** Falloff for primary light scattering to affect directionality.

### 6.1.4 Shallow Scattering

---

#### Deprecated

*Shallow Scattering* will be removed in a future version. A properly tweaked *Depth Fog Density* achieves better results at lower cost. Consider copying over the value from our materials.

---

The water colour can be varied in shallow water (this requires a depth cache created so that the system knows which areas are shallow, see section *Shorelines and Shallows*).

**Enable** Enable light scattering in shallow water. *BIRP URP*

**Scatter Colour Shallow** Scatter colour used for shallow water.

**Scatter Colour Depth Max** Maximum water depth that is considered ‘shallow’, in metres. Water that is deeper than this depth is not affected by shallow colour.

**Scatter Colour Depth Falloff** Falloff of shallow scattering, which gives control over the appearance of the transition from shallow to deep.

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

### 6.1.5 Reflection Environment

**Specular** Strength of specular lighting response.

**Occlusion** Strength of reflection. *HDRP*

**Smoothness** Smoothness of surface. *HDRP URP*

**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. *URP*

**Smoothness Far** Material smoothness at far distance from camera. *HDRP URP*

**Smoothness Far Distance** Definition of far distance. *HDRP URP*

**Smoothness Falloff** How smoothness varies between near and far distance. *HDRP URP*

**Roughness** Controls blurriness of reflection *BIRP*

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

**Light Intensity Multiplier** Main light intensity multiplier. *URP*

**Fresnel Power** Controls harshness of Fresnel behaviour. *BIRP URP*

**Refractive Index of Air** Index of refraction of air. Can be increased to almost 1.333 to increase visibility up through water surface. *BIRP URP*

---

#### Deprecated

The *Refractive Index of Air* property will be removed in a future version.

---

**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. *BIRP URP*

**Planar Reflections Distortion** How much the water normal affects the planar reflection. *BIRP URP*

**Override Reflection Cubemap** Whether to use an overridden reflection cubemap (provided in the next property). *BIRP*

**Reflection Cubemap Override** Custom environment map to reflect. *BIRP*

### 6.1.6 Add Directional Light

**Enable** Add specular highlights from the the primary light. *BIRP*

**Boost** Specular highlight intensity. *BIRP*

**Falloff** Falloff of the specular highlights from source to camera. *BIRP*

**Vary Falloff Over Distance** Helps to spread out specular highlight in mid-to-background. *BIRP*

**Far Distance** Definition of far distance. *BIRP*

**Falloff At Far Distance** Same as “Falloff” except only up to “Far Distance”. *BIRP*

### 6.1.7 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. *BIRP URP*

**Base** Base sky colour. *BIRP URP*

**Towards Sun** Colour in sun direction. *BIRP URP*

**Directionality** Direction fall off. *BIRP URP*

**Away From Sun** Colour away from sun direction. *BIRP URP*

### 6.1.8 Foam

**Enable** Enable foam layer on ocean surface.

**Foam** Foam texture.

**Foam Scale** Foam texture scale.

**Foam Feather** Controls how gradual the transition is from full foam to no foam.

**Foam Tint** Colour tint for whitecaps / foam on water surface. *BIRP URP*

**Light Scale** Scale intensity of lighting. *BIRP URP*

**Shoreline Foam Min Depth** Proximity to sea floor where foam starts to get generated. *BIRP URP*

**Foam Albedo Intensity** Scale intensity of diffuse lighting. *HDRP*

**Foam Emissive Intensity** Scale intensity of emitted light. *HDRP*

**Foam Smoothness** Smoothness of foam material. *HDRP*

### 6.1.9 Foam 3D Lighting

**Enable** Generates normals for the foam based on foam values/texture and use it for foam lighting. *BIRP URP*

**Foam Normal Strength** Strength of the generated normals.

**Specular Fall-Off** Acts like a gloss parameter for specular response. *BIRP URP*

**Specular Boost** Strength of specular response. *BIRP URP*

### 6.1.10 Foam Bubbles

**Foam Bubbles Color** Colour tint bubble foam underneath water surface.

**Foam Bubbles Parallax** Parallax for underwater bubbles to give feeling of volume.

**Foam Bubbles Coverage** How much underwater bubble foam is generated.

### 6.1.11 Transparency

**Enable** Whether light can pass through the water surface. *BIRP URP*

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

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

### 6.1.12 Caustics

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

**Caustics** Caustics texture.

**Caustics Scale** Caustics texture scale.

**Caustics Texture Grey Point** The ‘mid’ value of the caustics texture, around which the caustic texture values are scaled.

**Caustics Strength** Scaling / intensity.

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

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

**Caustics Distortion Texture** Texture to distort caustics. *HDRP*

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

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

### 6.1.13 Underwater

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

**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.

### 6.1.14 Flow

---

#### Example

Flow is demonstrated in the *whirlpool* example scene.

---

**Enable** Flow is horizontal motion in water. ‘Create Flow Sim’ must be enabled on the OceanRenderer to generate flow data.

## 6.2 Lighting

### 6.2.1 General

BIRP

---

#### Sponsor

Sponsoring us will help increase our development bandwidth which could work towards improving this feature.

[Sponsor Us](#)

---

*Crest* BIRP does not support additional lights due to bugs in the pipeline and performance concerns. Please see [#382](#) and [#383](#) for more details.

---

#### TODO

This section is a work in progress.

---

HDRP

As other shaders would, the ocean will get its lighting from the primary directional light (AKA sun). Like other mesh renderers, this can be masked by setting the *Rendering Layer Mask* property on the *Ocean Renderer*. Please see the [HDRP documentation on light layers](#) for more information on setup and usage.

But some lighting will come from the light set as the *Primary Light* on the *Ocean Renderer*. This includes the sub-surface scattering colour.

Lighting can also be overridden with the *Indirect Lighting Controller*. Please see the [HDRP documentation on volume overrides](#) for more information on setup and usage.

For the ocean to have lighting completely separate from everything else, you would need to do all of the above.

URP

---

## Sponsor

Sponsoring us will help increase our development bandwidth which could work towards improving this feature.

[Sponsor Us](#)

---

*Crest* URP currently does not support additional lights.

---

## TODO

This section is a work in progress.

---

### 6.2.2 Reflections

Reflections contribute hugely to the appearance of the ocean. The look of the ocean will dramatically changed based on the reflection environment.

The Index of Refraction setting controls how much reflection contributes for different view angles.

#### BIRP

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.

#### HDRP

*Crest* makes full use of the flexible lighting options in HDRP (it is lit the same as a shadergraph shader would be).

## Planar Reflection Probes

HDRP comes with a *Planar Reflection Probe* feature which enables dynamic reflection of the environment at run-time, with a corresponding cost. See Unity's documentation on [Planar Reflection Probes](#). At time of writing we used the following steps:

- Create new GameObject
- Set the height of the GameObject to the sea level.
- Add the component from the Unity Editor menu using *Component/Rendering/Planar Reflection Probe*
- Set the extents of the probe to be large enough to cover everything that needs to be reflected. We recommend starting large (1000m or more as a starting point).
- Ensure water is not included in the reflection by deselecting *Water* on the *Culling Mask* field
- Check the documentation linked above for details on individual parameters

HDRP's planar reflection probe is very sensitive to surface normals and often 'leaks' reflections, for example showing the reflection of a boat on the water above the boat. If you see these issues we recommend reducing the *Overall Normal Strength* parameter on the ocean material.

The planar reflection probe assumes the reflecting surface is a flat plane. This is not the case for a wavy water surface and this can also produce 'leaky' reflections. In such cases it can help to lower the reflection probe below sea level slightly.

## Screen-Space Reflections

HDRP has a separate setting for transparents to receive SSR (Screen-Space Reflections) and it is not enabled by default. It is important that you understand the basics of HDRP before proceeding.

1. Enable *Screen Space Refection* and the *Transparent* sub-option in the [Frame Settings](#).
2. Add and configure the [SSR Volume Override](#)
  - Please learn how to use the *Volume Framework* before proceeding as covering this is beyond the scope of our documentation:

<https://www.youtube.com/watch?v=vczkfjLoPf8>

Fig. 6.1: Adding Volumes to HDRP (Tutorial)

3. Enable *Receives Screen-Space Reflections* on the ocean material.

## URP

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).
- 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.

### 6.2.3 Refractions

Refractions sample from the camera's colour texture. Anything rendered in the transparent pass or higher will not be included in refractions.

See *Transparent Object In Front Of Ocean Surface* for issues with Crest and other refractive materials.

## 6.3 Orthographic Projection

Crest supports orthographic projection out-of-the-box, but it might require some configuration to get a desired appearance.

Crest uses the camera's position for the LOD system which can be awkward for orthographic which uses the size property on the camera. Use the *Viewpoint* property on the *Ocean Renderer* to override the camera's position.

Underwater effects do *not* currently support orthographic projection.

## 6.4 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.



## 6.5 Advanced Ocean Parameters

These parameters are found on the *Ocean Renderer* under the *Advanced* heading.

- **Surface Self-Intersection Mode** - How Crest should handle self-intersections of the ocean surface caused by choppy waves which can cause a flipped underwater effect. When not using the portals/volumes, this fix is only applied when within 2 metres of the ocean surface. *Automatic* will disable the fix if portals/volumes are used and is the recommended setting.
- **Underwater Cull Limit** - Proportion of visibility below which ocean will be culled underwater. The larger the number, the closer to the camera the ocean tiles will be culled.



## OCEAN SIMULATION

The following sections cover the major elements of the ocean simulation. All of these can be directly controlled with user input, as covered in this video:

<https://www.youtube.com/watch?v=sQIakAjSq4Y>

Fig. 7.1: Basics of Adding Ocean Inputs

---

**Note:** Inputs only execute the first shader pass (pass zero). It is recommended to use unlit shader templates or unlit *Shader Graph* (URP only) if not using one of ours.

---

---

**Tip:** For inputs, you are not limited to only using a [MeshRenderer](#). Almost any renderer can be used like a [TrailRenderer](#), [LineRenderer](#) or [ParticleSystem](#).

---

The following shaders can be used with any ocean input:

- **Scale By Factor** scales the ocean data between zero and one inclusive. It is multiplicative, which can be inverted, so zero becomes no data and one leaves the data unchanged.

## 7.1 Animated Waves

### 7.1.1 Overview

The Animated Waves simulation contains the animated surface shape. This typically contains the ocean waves (see the Wave conditions section below), but can be modified as required. For example, parts of the water can be pushed down below geometry if required.

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 [Sea Floor Depth](#)). The amount that waves are attenuated is configurable using the *Attenuation In Shallows* setting.

## 7.1.2 Simulation Settings

All of the settings below refer to the *Animated Waves Sim Settings* asset.

- **Attenuation In Shallows** - How much waves are dampened in shallow water.
- **Shallows Max Depth** - Any water deeper than this will receive full wave strength. The lower the value, the less effective the depth cache will be at attenuating very large waves. Set to the maximum value (1,000) to disable.
- **Collision Source** - Where to obtain ocean shape on CPU for physics / gameplay.
- **Max Query Count** - Maximum number of wave queries that can be performed when using ComputeShader-Queries.
- **Ping Pong Combine Pass** - Whether to use a graphics shader for combining the wave cascades together. Disabling this uses a compute shader instead which doesn't need to copy back and forth between targets, but it may not work on some GPUs, in particular pre-DX11.3 hardware, which do not support typed UAV loads. The fail behaviour is a flat ocean.
- **Render Texture Graphics Format** - The render texture format to use for the wave simulation. Consider using higher precision (like R32G32B32A32\_SFloat) if you see tearing or wierd normals. You may encounter this issue if you use any of the *Set Water Height* inputs.

## 7.1.3 User Inputs

---

### Preview

Splines now support animated waves. Please see [Wave Splines](#) for general spline information.

---

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 *Register Anim Waves Input* script which will tag it for rendering into the shape. This is demonstrated in [Fig. 7.1](#)

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 following input shaders are provided under *Crest/Inputs/Animated Waves*:

- **Add From Texture** allows any kind of shape added to the surface from a texture. Can either be a heightmap texture (1 channel) or a 3 channel XYZ displacement texture. Optionally the alpha channel can be used to write to subsurface scattering which increases the amount of light emitted from the water volume, which is useful for approximating light scattering.
- **Scale By Factor** scales the waves by a factor where zero is no waves and one leaves waves unchanged. Useful for reducing waves.
- **Set Base Water Height Using Geometry** allows the sea level (average water height) to be offset some amount. The top surface of the geometry will provide the water height, and the waves will apply on top.
- **Push Water Under Convex Hull** pushes the water underneath the geometry. Can be used to define a volume of space which should stay 'dry'.
- **Set Water Height Using Geometry** snaps the water surface to the top surface of the geometry. Will override any waves.
- **Wave Particle** is a 'bump' of water. Many bumps can be combined to make interesting effects such as wakes for boats or choppy water. Based loosely on <http://www.cemyuksel.com/research/waveparticles/>.

## 7.2 Dynamic Waves

### 7.2.1 Overview

Crest includes a multi-resolution dynamic wave simulation, which allows objects like boats to interact with the water.

To turn on this feature, enable the *Create Dynamic Wave Sim* option on the *OceanRenderer* script, and to configure the sim, create or assign a *Dynamic Wave Sim Settings* asset on the *Sim Settings Dynamic Waves* option.

The dynamic wave simulation is added on top of the animated FFT waves to give the final shape.

The dynamic wave simulation is not suitable for use further than approximately 10km from the origin. At this kind of distance the stability of the simulation can be compromised. Use the *FloatingOrigin* component to avoid travelling far distances from the world origin.

### 7.2.2 Adding Interaction Forces

Dynamic ripples from interacting objects can be generated by placing one or more spheres under the object to approximate the object's shape. To do so, attach one or more *SphereWaterInteraction* components to children with the object and set the *Radius* parameter to roughly match the shape.

The following settings can be used to customise the interaction:

- **Radius** - The radius of the sphere from which the interaction forces are calculated.
- **Weight** - Strength of the effect. Can be set negative to invert.
- **Weight Up Down Mul** - Multiplier for vertical motion, scales ripples generated from a sphere moving up or down.
- **Inner Sphere Multiplier** - Internally the interaction is modelled by a pair of nested spheres. The forces from the two spheres combine to create the final effect. This parameter scales the effect of the inner sphere and can be tweaked to adjust the shape of the result.
- **Inner Sphere Offset** - This parameter controls the size of the inner sphere and can be tweaked to give further control over the result.
- **Velocity Offset** - Offsets the interaction position in the direction of motion. There is some latency between applying a force to the wave sim and the resulting waves appearing. Applying this offset can help to ensure the waves do not lag behind the sphere.
- **Compensate For Wave Motion** - If set to 0, the input will always be applied at a fixed position before any horizontal displacement from waves. If waves are large then their displacement may cause the interactive waves to drift away from the object. This parameter can be increased to compensate for this displacement and combat this issue. However increasing too far can cause a feedback loop which causes strong 'ring' artifacts to appear in the dynamic waves. This parameter can be tweaked to balance this two effects.

Non-spherical objects can be approximated with multiple spheres, for an example see the *Spinner* object in the *boat.unity* example scene which is composed of multiple sphere interactions. The intensity of the interaction can be scaled using the *Weight* setting. For an example of usages in boats, search for GameObjects with "InteractionSphere" in their name in the *boat.unity* scene.

## 7.2.3 Simulation Settings

All of the settings below refer to the *Dynamic Wave Sim Settings* asset.

The key settings that impact stability of the simulation are the **Damping** and **Courant Number** settings described below.

- **Simulation Frequency** - Frequency to run the dynamic wave sim, in updates per second. Lower frequencies can be more efficient but may limit wave speed or lead to visible jitter. Default is 60 updates per second.
- **Damping** - How much energy is dissipated each frame. Helps sim stability, but limits how far ripples will propagate. Set this as large as possible/acceptable. Default is 0.05.
- **Courant Number** - Stability control. Lower values means more stable sim, but may slow down some dynamic waves. This value should be set as large as possible until sim instabilities/flickering begin to appear. Default is 0.7.
- **Horiz Displace** - Induce horizontal displacements to sharpen simulated waves.
- **Displace Clamp** - Clamp displacement to help prevent self-intersection in steep waves. Zero means unclamped.
- **Gravity Multiplier** - Multiplier for gravity. More gravity means dynamic waves will travel faster.
- **Attenuation in Shallows** - How much waves are dampened in shallow water.

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

## 7.2.4 User Inputs

The recommended approach to injecting forces into the dynamic wave simulation is to use the *SphereWaterInteraction* component as described above. This component will compute a robust interaction force between a sphere and the water, and multiple spheres can be composed to model non-spherical shapes.

However for when more control is required custom forces can be injected directly into the simulation. The following input shader is provided under *Crest/Inputs/Dynamic Waves*:

- **Add Bump** adds a round force to pull the surface up (or push it down). This can be moved around to create interesting effects.

## 7.3 Foam

### 7.3.1 Overview

Crest simulates foam getting generated by choppy water (*pinched*) wave crests) and in shallow water to approximate foam from splashes at shoreline. Each update (default is 30 updates per second), the foam values are reduced to model gradual dissipation of foam over time.

To turn on this feature, enable the *Create Foam Sim* option on the *OceanRenderer* script, and ensure the *Enable* option is ticked in the Foam group on the ocean material.

To configure the foam sim, create a *Foam Sim Settings* asset by right clicking the a folder in the *Project* window and selecting *Create/Crest/Foam Sim Settings*, and assigning it to the *OceanRenderer* component in your scene.

## 7.3.2 User Inputs

### Preview

Splines now support foam. Please see [Wave Splines](#) for general spline information.

Crest supports inputting any foam into the system, which can be helpful for fine tuning where foam is placed. To place foam, add some geometry into the world at the area where foam should be added. Then assign the *RegisterFoamInput* script which will tag it for rendering into the shape, and apply a material with a shader of type *Crest/Inputs/Foam/...*. See the *DepositFoamTex* object in the *whirlpool.unity* scene for an example.

The process for adding inputs is demonstrated in this [Fig. 7.1](#).

The following input shaders are provided under *Crest/Inputs/Foam*:

- **Add From Texture** adds foam values read from a user provided texture. Can be useful for placing ‘blobs’ of foam as desired, or can be moved around at runtime to paint foam into the sim.
- **Add From Vert Colours** can be applied to geometry and uses the red channel of vertex colours to add foam to the sim. Similar in purpose to *Add From Texture*, but can be authored in a modelling workflow instead of requiring a texture.
- **Override Foam** sets the foam to the provided value. Useful for removing foam from unwanted areas.

## 7.3.3 Simulation Settings

### General Settings

- **Foam Fade Rate** - How quickly foam dissipates. Low values mean foam remains on surface for longer. This setting should be balanced with the generation *strength* parameters below.

### Wave foam / whitecaps

Crest detects where waves are ‘pinched’ and deposits foam to approximate whitecaps.

- **Wave Foam Strength** - Scales intensity of foam generated from waves. This setting should be balanced with the *Foam Fade Rate* setting.
- **Wave Foam Coverage** - How much of the waves generate foam. Higher values will lower the threshold for foam generation, giving a larger area.

### Shoreline foam

If water depth input is provided to the system (see **Sea Floor Depth** section below), the foam sim can automatically generate foam when water is very shallow, which can approximate accumulation of foam at shorelines.

- **Shoreline Foam Max Depth** - Foam will be generated in water shallower than this depth. Controls how wide the band of foam at the shoreline will be. Note that this is not a distance to shoreline, but a threshold on water depth, so the width of the foam band can vary based on terrain slope. To address this limitation we allow foam to be manually added from geometry or from a texture, see the next section.
- **Shoreline Foam Strength** - Scales intensity of foam generated in shallow water. This setting should be balanced with the *Foam Fade Rate* setting.

## Developer Settings

These settings should generally be left unchanged unless one is experiencing issues.

- **Simulation Frequency** - Frequency to run the foam sim, in updates per second. Lower frequencies can be more efficient but may lead to visible jitter. Default is 30 updates per second.

## 7.4 Sea Floor Depth

This simulation stores information that can be used to calculate the water depth. Specifically it stores the terrain height, which can then be differenced with the sea level to obtain the water depth. This water depth is useful information to 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, etcetera.
- 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.

## 7.5 Clip Surface

[https://www.youtube.com/watch?v=jXphUy\\_\\_J0o](https://www.youtube.com/watch?v=jXphUy__J0o)

Fig. 7.2: Water Bodies and Surface Clipping

This data drives clipping of the ocean surface, as in carving out holes. This can be useful for hollow vessels or low terrain that goes below sea level. Data can come from primitives (signed-distance), geometry (convex hulls) or a texture.

To turn on this feature, enable the *Create Clip Surface Data* option on the *OceanRenderer* script, and ensure the *Enable* option is ticked in the *Clip Surface* group on the ocean material.

The data contains 0-1 values. Holes are carved into the surface when the value is greater than 0.5.



## 7.5.1 Simulation Settings

All of the settings below refer to the *Clip Surface Sim Settings* asset.

- **Render Texture Graphics Format** - The render texture format to use for the clip surface simulation. Consider using higher precision (like *R16\_UNorm*) if you are using *Primitive* mode for even more accurate clipping.

## 7.5.2 User Inputs

### Primitive Mode

Clip areas can be added using signed-distance primitives which produces accurate clipping and supports overlapping. Add a *RegisterClipSurfaceInput* script to a *GameObject* and set *Mode* to *Primitive*. The position, rotation and dimensions of the primitive is determined by the *Transform*. See the *FloatingOpenContainer* object in the *boat.unity* scene for an example usage.

### Geometry Mode

Clip areas can be added by adding geometry that covers the desired hole area to the scene and then assigning the *RegisterClipSurfaceInput* script and setting *Mode* to *Geometry*. See the *RowBoat* object in the *main.unity* scene for an example usage.

To use other available shaders like *ClipSurfaceRemoveArea* or *ClipSurfaceRemoveAreaTexture*: create a material, assign to renderer and disable *Assign Clip Surface Material* option. For the *ClipSurfaceRemoveArea* shaders, the geometry should be added from a top-down perspective and the faces pointing upwards.

The following input shaders are provided under *Crest/Inputs/Clip Surface*:

- **Convex Hull** - Renders geometry into clip surface data taking all dimensions into account. An example use case is rendering the convex hull of a vessel to remove the ocean surface from within it.

---

### Example

See the *RowBoat* object in the *main.unity* scene for an example usage.

---



---

**Note:** Overlapping or adjacent meshes will not work correctly in most cases. There will be cases where one mesh will overwrite another resulting in the ocean surface appearing where it should not. The mesh is rendered from a top-down perspective. The back faces add clip surface data and the front faces remove from it which creates the convex hull. With an overlapping mesh, the front faces of the sides of one mesh will clear the clipping data creating by the other mesh. Overlapping boxes which are not rotated on the X or Z axes will work well whilst spheres will have issues. Consider using *Primitive* mode which supports overlapping.

---

- **Include Area** - Removes clipping data so the ocean surface renders.
- **Remove Area** - Adds clipping data to remove the ocean surface.
- **Remove Area Texture** - Adds clipping data using a texture to remove the ocean surface.

## 7.6 Shadows

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

This data is captured from the shadow maps Unity renders before the transparent pass. 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.

### BIRP

To turn on this feature, enable the *Create Shadow Data* option on the *OceanRenderer* script, and ensure the *Shadowing* option is ticked on the ocean material.

### HDRP

To turn on this feature, enable the *Create Shadow Data* option on the *OceanRenderer* script.

Specular (direct) lighting on the ocean surface is not shadowed by this data. It is shadowed by the pipeline. But we still use the data to shadow anything not covered by the pipeline like caustic shadows.

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

1. On the HDRP asset (either the asset provided with *Crest Assets/Crest/CrestExampleHDRPAsset*, or the one used in your project), ensure that *Custom Pass* is enabled.
2. Shadow maps must be enabled in the frame settings for the camera.
3. Enable shadowing in Crest. Enable *Create Shadow Data* on the *OceanRenderer* script.
4. 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.
5. If desired the shadow sim can be configured by assigning a *Shadow Sim Settings* asset (*Create/Crest/Shadow Sim Settings*).

### URP

To turn on this feature, enable the *Create Shadow Data* option on the *OceanRenderer* script, and ensure the *Shadowing* option is ticked on the ocean material.

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

1. In the [shadow settings of the URP asset](#), ensure that shadow cascades are enabled. *Crest* requires cascades to be enabled to obtain shadow information.
2. Enable shadowing in Crest. Enable *Create Shadow Data* on the *OceanRenderer* script.
3. 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.
4. If desired the shadow sim can be configured by assigning a *Shadow Sim Settings* asset (*Create/Crest/Shadow Sim Settings*).
5. 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.

There will be times when the shadow jitter settings will cause shadows or light to leak. An example of this is when trying to create a dark room during daylight. At the edges of the room the jittering will cause the ocean on the inside of the room (shadowed) to sample outside of the room (not shadowed) resulting in light at the edges. Reducing the

*Jitter Diameter Soft* setting can solve this, but we have also provided a *Register Shadow Input* component which can override the shadow data. This component bypasses jittering and gives you full control.

## 7.7 Flow

### 7.7.1 Overview

Flow is the horizontal motion of the water volumes. It does not affect wave directions, but transports the waves horizontally. This horizontal motion also affects physics.

---

#### Example

See the *whirlpool.unity* example scene where flow is used to rotate the waves and foam around the vortex.

---

### 7.7.2 User Inputs

---

#### Preview

Splines now support flow. Please see [Wave Splines](#) for general spline information.

---

Crest supports adding any flow velocities to the system. To add flow, add some geometry into the world which when rendered from a top down perspective will draw the desired displacements. Then assign the *RegisterFlowInput* script which will tag it for rendering into the flow, and apply a material using one of the following shaders.

The following input shaders are provided under *Crest/Inputs/Flow*:

The *Crest/Inputs/Flow/Add Flow Map* shader writes a flow texture into the system. It assumes the x component of the flow velocity is packed into 0-1 range in the red channel, and the z component of the velocity is packed into 0-1 range in the green channel. The shader reads the values, subtracts 0.5, and multiplies them by the provided scale value on the shader. The process of adding ocean inputs is demonstrated in [Fig. 7.1](#).

## 7.8 Albedo

---

#### Preview

This feature is currently in preview.

---

## 7.8.1 Overview

The Albedo feature allows a colour layer to be composited on top of the water surface. This is useful for projecting colour onto the surface.

This is somewhat similar to decals, except the colour only affects the water.

HDRP has a [Decal Projector](#) feature that works with the water, and the effect is more configurable and may be preferred over this feature. When using this feature be sure to enable [Affects Transparent](#).

URP 2022 has a decal system but it does not support transparent surfaces like water.

## 7.8.2 User Inputs

---

**Note:** Inputs only execute the first shader pass (pass zero). It is recommended to use unlit shader templates or unlit *Shader Graph* (URP only) if not using one of ours. Shaders provided by *Unity* generally will not work as their primary pass is not zero - even for unlit shaders.

---

Any geometry or particle system can add colour to the water. It will be projected from a top down perspective onto the water surface.

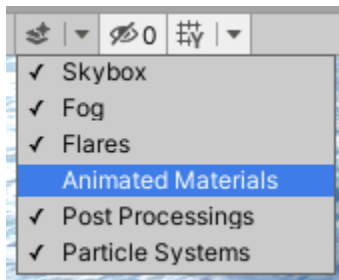
To tag `GameObjects` to render onto the water, attach the *RegisterAlbedoInput* component.

## WAVE CONDITIONS

The following sections describe how to define the wave conditions.

**Tip:** It is useful to see the animated ocean surface while tweaking the wave conditions.

By default, the update speed is intentionally throttled by Unity to save power when in edit mode. To enable real-time update, enable *Animated Materials* in the Scene View toggles:



See the [Unity Documentation](#) for more information.

---

### 8.1 Wave Systems

The *ShapeFFT* component is used to generate waves in Crest.

For advanced situations where a high level of control is required over the wave shape, the *ShapeGerstner* component can be used to add specific wave components.

### 8.2 Authoring

To add waves, add the *ShapeFFT* component to a GameObject (comparison of the two options above).

The appearance and shape of the waves is determined by a *Wave Spectrum*. A default wave spectrum will be created if none is specified. To author wave conditions, click the *Create Asset* button next to the *Spectrum* field. The resulting spectrum can then be edited by expanding this field.

The spectrum can be freely edited in Edit mode, and is locked by default in Play mode to save evaluating the spectrum every frame (this optimisation can be disabled using the *Spectrum Fixed At Runtime* toggle). The spectrum has sliders for each wavelength to control contribution of different scales of waves. To control the contribution of 2m wavelengths, use the slider labelled '2'. Note that the wind speed may need to be increased on the *OceanRenderer* component in order for large wavelengths to be visible.

There is also control over how aligned waves are to the wind direction. This is controlled via the *Wind Turbulence* control on the *ShapeFFT* component.

Another key control is the *Chop* parameter which scales the horizontal displacement. Higher chop gives crisper wave crests but can result in self-intersections or ‘inversions’ if set too high, so it needs to be balanced.

To aid in tweaking the spectrum, we provide a standard empirical wave spectrum model from the literature, called the ‘Pierson-Moskowitz’ model. To apply this model to a spectrum, select it in the *Empirical Spectra* section of the spectrum editor which will lock the spectrum to this model. The model can be disabled afterwards which will unlock the spectrum power sliders for hand tweaking.

---

**Tip:** Notice how the empirical spectrum places the power slider handles along a line. This is typical of real world wave conditions which will have linear power spectrums on average. However actual conditions can vary significantly based on wind conditions, land masses, etc, and we encourage experimentation to obtain visually interesting wave conditions, or conditions that work best for gameplay.

---

Together these controls give the flexibility to express the great variation one can observe in real world seascapes.

## 8.3 Wave Splines

---

### Preview

This feature is in preview and may change in the future.

---

<https://www.youtube.com/watch?v=JRzPcUP5aaA>

Fig. 8.1: Wave Splines

Wave Splines allow flexible and fast authoring of how waves manifest in the world. A couple of use cases are demonstrated in the video above.

If the *Spline* component is attached to the same *GameObject* as a *ShapeFFT* component, the waves will be generated along the spline. This allows for quick experimentation with placing and orienting waves in different areas of the environment.

The *Spline* component can also be combined with the *RegisterHeightInput* to make the water level follow the spline, and with the *RegisterFlowInput* to make water move along the spline.

## SHORELINES AND SHALLOWS

Changed in version 4.9: *Layer Names* (String[]) has changed to *Layers* (LayerMask).

*Crest* requires water depth information to attenuate large waves in shallow water, to generate foam near shorelines, and to provide shallow water shading. The way this information is typically generated is through the *OceanDepthCache* component, which takes one or more layers, and renders everything in those layers (and within its bounds) from a top-down orthographic view to generate a heightfield for the seabed. These layers could contain the render geometry/terrains, or it could be geometry that is placed in a non-rendered layer that serves only to populate the depth cache. By default this generation is done at run-time during startup, but the component exposes other options such as generating offline and saving to an asset, or rendering on demand.

The seabed affects the wave simulation in a physical way - the rule of thumb is *waves will be affected by the seabed when the water depth is less than half of their wavelength*. So for example when the water is 250m deep, this will start to dampen 500m wavelengths from the spectrum, so it is recommended that the seabed drop down to at least 500m 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 in the surface waves and/or a line of foam. Alternatively, there is *Shallows Max Depth* on the *Sim Settings Animated Waves* asset which smooths the attenuation to a provided maximum depth where waves will be at full strength.

### 9.1 Setup

<https://www.youtube.com/watch?v=jcmqUlboTUK>

Fig. 9.1: Depth Cache usage and setup

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.

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 *Layers* field contains the layer that the island is assigned to (*Terrain* in our project). Only objects in these layer(s) will render into the cache.

- Both the transform scale (white rectangle) and the *Layers* property determine what will be rendered into the cache.

By default the cache is populated in the *Start()* function. It can instead be configured to populate from script by setting the *Refresh Mode* to *On Demand* and calling the *PopulateCache()* method on the component from script.

Once populated the cache contents can be saved to disk by clicking the *Save cache to file* button that will appear in the Inspector in play mode. Once saved, the *Type* field can be set to *Baked* and the saved data can be assigned to the *Saved Cache* field.

## 9.2 Shoreline Waves

Modelling realistic shoreline waves efficiently is a challenging open problem. We discuss further and make suggestions on how to set up shorelines with *Crest* in the following video.

<https://www.youtube.com/watch?v=Y7ny8pKzWMk>

Fig. 9.2: Tweaking Shorelines

## 9.3 Troubleshooting

*Crest* runs validation on the depth caches - look for warnings/errors in the Inspector, and in the log at run-time, where many issues will be highlighted.

To inspect the contents of the cache, look for a child *GameObject* parented below the cache with the name 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. The cache will appear black for dry land and red for water that is at least 1m deep.



## OCEANS, RIVERS AND LAKES

---

### Preview

The features described in this section are in preview and may evolve in future versions.

---

### 10.1 Oceans

By default Crest generates an infinite body of water at a fixed sea level, suitable for oceans and very large lakes.

### 10.2 Lakes

Crest can be configured to efficiently generate smaller bodies of water, using the following mechanisms.

- The waves can be generated in a limited area - see the *Wave Splines* section.
- The *WaterBody* component, if present, marks areas of the scene where water should be present. It can be created by attaching this component to a *GameObject* and setting the X/Z scale to set the size of the water body. If gizmos are enabled an outline showing the size will be drawn in the Scene View.
- The *WaterBody* component turns off tiles that do not overlap the desired area. The *Clip Surface* feature can be used to precisely remove any remaining water outside the intended area. Additionally, the clipping system can be configured to clip everything by default, and then areas can be defined where water should be included. See the *Clip Surface* section.
- If the lake altitude differs from the global sea level, create a spline that covers the area of the lake and attach the *RegisterHeightInput* component which will set the water level to match the spline (or click the *Set Height* button in the *Spline* inspector). It is recommended to cover a larger area than the lake itself, to give a protective margin against LOD effects in the distance.

---

### Example

The *LakesAndRivers.unity* scene contains an example of a lake connected by a river.

---

Another advantage of the *WaterBody* component is it allows an optional override material to be provided, to change the appearance of the water. Since this feature cannot be applied partially to an ocean tile, and an ocean tile can overlap two water bodies, this feature does not work well with bordering water bodies. If you use this feature and want to still have an ocean, then disable *Water Body Culling* on the *Ocean Renderer*.

## 10.3 Rivers

Splines can also be used to create rivers, by creating a spline at the water surface of the river, and attaching the following components:

- *RegisterHeightInput* can be used to set the water level to match the spline.
- *RegisterFlowInput* can be used to make the water move along the spline.
- *ShapeFFT* can be used to generate waves that propagate along the river.

The *Add Feature* section of the *Spline* inspector has helper buttons to quickly add these components.

---

### Example

The *LakesAndRivers.unity* scene contains an example of a river connecting two lakes.

---

## 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.

---

### Research

We use a technique called *Fixed Point Iteration* to calculate the water height. We gave a talk at GDC about this technique which may be useful to learn more: <http://www.huwbowles.com/fpi-gdc-2016/>.

---

The *Visualise Collision Area* debug component is useful for visualising the collision shape for comparison against the render surface. It draws debug line crosses in the Scene View around the position of the component.

## 11.1 Collision API Usage

The collision providers built into our system perform queries asynchronously; queries are offloaded to the GPU or to spare CPU cores for processing. This has a few non-trivial impacts on how the query API must be used.

Firstly, queries need to be registered with an ID so that the results can be tracked and retrieved later. This ID needs to be globally unique, and therefore should be acquired by calling *GetHashCode()* on an object/component which will be guaranteed to be unique. A primary reason why *SampleHeightHelper* is useful is that it is an object in itself and there can pass its own ID, hiding this complexity from the user.

---

**Important:** Queries should only be made once per frame from an owner - querying a second time using the same ID will stomp over the last query points.

---

Secondly, even if only a one-time query of the height is needed, the query function should be called every frame until it indicates that the results were successfully retrieved. See *SampleHeightHelper* and its usages in the code - its *Sample()* function should be called until it returns true. Posting the query and polling for its result are done through the same function.

Finally due to the above properties, the number of query points posted from a particular owner should be kept consistent across frames. The helper classes always submit a fixed number of points this frame, so satisfy this criteria.

## 11.2 Compute Shape Queries (GPU)

This is the default and recommended choice for when a GPU is present. 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 wave components and depth caches and other Crest features.

## 11.3 Baked FFT Data (CPU)

---

### Preview

This feature is in preview and may change in the future.

---

In scenarios where a GPU is not present such as for headless servers, a CPU option is available.

To use this feature, select a *Shape FFT* component that is generating the waves in a scene and enable the **Enable Baked Collision**. Next configure the following options:

- **Time Resolution** - Frames per second of baked data. Larger values may help the collision track the surface closely at the cost of more frames and increase baked data size.
- **Smallest Wavelength Required** - Smallest wavelength required in collision. To preview the effect of this, disable power sliders in spectrum for smaller values than this number. Smaller values require more resolution and increase baked data size.
- **Time Loop Length** - FFT waves will loop with a period of this many seconds. Smaller values decrease data size but can make waves visibly repetitive.

Next click **Bake to asset and assign to current settings** and select a path and filename for the result. After the bake completes the current active *Animated Waves Sim Settings* will be configured to use this data.

---

**Important:** There are currently a few key limitations of this approach:

- Only a single set of waves from one *Shape FFT* component is supported. This collision does not support multiple sets of waves.
  - The *Depth Cache* components are not supported. In order to get a one to one match between the visuals and the collision data, depth caches should not be used.
  - Varying water levels such as rivers flowing down a gradient or lakes at different altitudes is not supported. This feature assumes a fixed sea level for the whole scene.
- 

---

### Sponsor

Sponsoring us will help increase our development bandwidth which could work towards solving the aforementioned limitations.

[Trello Card](#)

[Sponsor Us](#)

---

## 11.4 Gerstner Waves CPU (deprecated)

---

### Deprecated

The Gerstner wave system in Crest is now deprecated. A CPU query path for the FFT waves is being worked on.

---

This collision option is serviced directly by the *GerstnerWavesBatched* component which implements the *IColl-Provider* 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.

A drawback of this approach is the CPU performance cost of evaluating the waves. 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.



## UNDERWATER



*Crest* supports seamless transitions above/below water. It can also have a meniscus which renders a subtle line at the intersection between the camera lens and the water to visually help the transition. 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.

Out-scattering is provided as an example script which reduces environmental lighting with depth underwater. See the *UnderwaterEnvironmentalLighting* component.

For performance reasons, the underwater effect is disabled if the viewpoint is not underwater. Only the camera rendering the ocean surface will be used.

---

**Tip:** Use opaque or alpha test materials for underwater surfaces. Transparent materials may not render correctly underwater. See [Transparent Object Underwater](#) for possible workarounds.

---

## 12.1 Underwater Renderer

---

**Note:** You can enable/disable rendering in the scene view by toggling fog in the [scene view control bar](#).

---

The *Underwater Renderer* component executes a fullscreen underwater effect between the transparent pass and post-processing pass.

It is similar to a post-processing effect, but has the benefit of allowing other renderers to execute after it and still receive post-processing. An example is to add underwater fog correctly to semi-transparent objects.

This is the current underwater solution used for the example scenes, and is the simplest to setup.

### 12.1.1 Setup

#### BIRP

- Configure the ocean material for underwater rendering. In the *Underwater* section of the material params, ensure *Enabled* is turned on and *Cull Mode* is set to *Off* so that the underside of the ocean surface renders. See *Ocean-Underwater.mat* for an example.

#### HDRP

- Configure the ocean material for underwater rendering. Ensure that *Double-Sided* is enabled under *Surface Options* on the ocean material so that the underside of the ocean surface renders. See *Ocean-Underwater.mat* for an example.

#### URP

- Configure the ocean material for underwater rendering. In the *Underwater* section of the material params, ensure *Enabled* is turned on and *Cull Mode* is set to *Off* so that the underside of the ocean surface renders. See *Ocean-Underwater.mat* for an example.
- Add the *Underwater Renderer* component to your camera game object.

### 12.1.2 Parameters

- **Mode:** How the underwater effect (and ocean surface) is rendered:
  - **Full-Screen:** Full screen effect.
  - **Portal:** Renders the underwater effect and ocean surface from the geometry's front-face and behind it.
  - **Volume:** Renders the underwater effect and ocean surface from the geometry's front-face to its back-face.
  - **Volume (Fly-Through):** Renders the underwater effect and ocean surface from the geometry's front-face to its back-face - even from within the volume.
- **Depth Fog Density Factor:** Reduces the underwater depth fog density by a factor. Useful to reduce the intensity of the fog independently from the ocean surface.



### 12.1.3 Detecting Above or Below Water

The *OceanRenderer* component has the *ViewerHeightAboveWater* property which can be accessed with `OceanRenderer.Instance.ViewerHeightAboveWater`. It will return the signed height from the ocean surface of the camera rendering the ocean. Internally this uses the *SampleHeightHelper* class which can be found in *SamplingHelpers.cs*.

There is also the *OceanSampleHeightEvents* example component (requires example content to be imported) which uses [UnityEvents](#) to provide a scriptless approach to triggering changes. Simply attach it to a game object, and it will invoke a `UnityEvent` when the attached game object is above or below the ocean surface once per state change. A common use case is to use it to trigger different audio when above or below the surface.

### 12.1.4 Portals & Volumes

---

#### Preview

This feature is in preview and may change in the future.

---

The underwater effect can be rendered from a provided mesh which will effectively become a portal (2D) or volume (3D). Change the *Mode* property to one of your choosing and set the *Volume Geometry* to a *Mesh Filter* (it will use its transform). This feature also clips the ocean surface to match. A common use case would be a window on a boat.

### 12.1.5 Underwater Shader API

---

#### Preview

This feature is in preview and may change in the future.

---

The underwater effect uses opaque depth and thus will not render correctly for transparent objects. Too much fog will be applied as it is as if the transparent object does not exist.

The most effective approach is to render the transparent objects after the underwater effect and apply the underwater effect as part of the shader for the transparent object (basically the same way Unity fog is applied).

The *Shader API* needs to be enabled on the *Underwater Renderer* (located under the *Shader API* heading).

#### BIRP

Once the *Shader API* is enabled, the underwater effect will be rendered before the transparent pass instead of after it, and the global shader properties will be populated. This means that when a transparent object is rendered, it will already have underwater fog behind it. It is then just a matter of applying the underwater fog to the transparent object.

---

#### Example

We have an example *Surface Shader* which you can use as a reference:

*Crest/Crest-Examples/Shared/Shaders/ExampleUnderwaterTransparentSurfaceShader.shader*

Furthermore, you can view the shader in action in the *Transparent Object Underwater* example in the *Examples* scene.

---

Setting up a shader can be broken down to the following:

1. Including our includes file:  
*Crest/Crest/Shaders/Underwater/UnderwaterEffectIncludes.hlsl*
2. Adding optional keywords (see example shader)
3. Use the *CrestApplyUnderwaterFog* function to apply the fog to the final color

Here is the important part from *ExampleUnderwaterTransparentSurfaceShader.shader*:

```
float2 positionNDC = IN.screenPos.xy / IN.screenPos.w;
float deviceDepth = IN.screenPos.z / IN.screenPos.w;

if (!CrestApplyUnderwaterFog(positionNDC, IN.worldPos, deviceDepth, _FogMultiplier,
    ↪color.rgb))
{
    UNITY_APPLY_FOG(IN.fogCoord, color);
}
```

## HDRP

Once the *Shader API* is enabled, any transparent object in the correct layer and using a modified shader (more on that later) will have its above water pixels rendered in the transparent pass and below water pixels rendered after the underwater pass.

In a perfect world, we would render the underwater pass before the transparent pass, and then apply the underwater effect to the final color of each transparent object using the *CrestNodeApplyUnderWaterFog* node. But *Shader Graph* does not allow modification of the final color.

The workaround is in the example node *CrestNodeApplyUnderwaterFogExample*. This node uses the *CrestNodeApplyUnderWaterFog* node and does a few things to get around this problem:

- Apply the underwater effect only to the Emission input to bypass *Unity*'s lighting
- Reduce the alpha and the color by distance from the camera

The end result is that the effect is inconsistent with the underwater pass. Despite that we believe it is a decent enough approximation until *Unity* improves this area.

---

## Example

We have an example *Surface Shader* which you can use as a reference:

*Crest/Crest-Examples/Shared/Shaders/LitTransparentWithUnderwaterFog.shadergraph*

Furthermore, you can view the shader in action in the *Transparent Object Underwater* example in the *Examples* scene.

---

Setting up a graph can be broken down to the following:

1. Add optional keywords (see example graph)
2. Add the *CrestNodeApplyUnderwaterFogExample* node
3. Connect *Fogged Color* (and alpha) and *Fogged Emission* outputs to the [Master Stack](#)
4. Multiply *Factor* output with any properties except *Ambient Occlusion*
5. Enable *Alpha Clipping*

For best results using the [Lit Shader](#) graph:

- Keep *Preserve Specular Lighting* disabled as this will cause the object to be visible from any distance
- Do not enable *Receive Fog* as this will write over the emission and thus underwater fog
- Be mindful of what features on the *Shader Graph* you enable as it might affect the underwater fog

## URP

Once the *Shader API* is enabled, any transparent object in the correct layer and using a modified shader (more on that later) will have its above water pixels rendered in the transparent pass and below water pixels rendered after the underwater pass.

In a perfect world, we would render the underwater pass before the transparent pass, and then apply the underwater effect to the final color of each transparent object using the *CrestNodeApplyUnderWaterFog* node. But *Shader Graph* does not allow modification of the final color.

The workaround is in the example node *CrestNodeApplyUnderwaterFogExample*. This node uses the *CrestNodeApplyUnderWaterFog* node and does a few things to get around this problem:

- Apply the underwater effect only to the Emission input to bypass *Unity*'s lighting
- Reduce the alpha and the color by distance from the camera

The end result is that the effect is inconsistent with the underwater pass. Despite that we believe it is a decent enough approximation until *Unity* improves this area.

---

## Example

We have an example *Surface Shader* which you can use as a reference:

*Crest/Crest-Examples/Shared/Shaders/LitTransparentWithUnderwaterFog.shadergraph*

Furthermore, you can view the shader in action in the *Transparent Object Underwater* example in the *Examples* scene.

---

Setting up a graph can be broken down to the following:

1. Add optional keywords (see example graph)
2. Add the *CrestNodeApplyUnderwaterFogExample* node
3. Connect *Fogged Color* (and alpha) and *Fogged Emission* outputs to the [Master Stack](#)
4. Multiply *Factor* output with any properties except *Ambient Occlusion*
5. Enable *Alpha Clipping*

## 12.2 Underwater Curtain *BIRP URP*

---

### Deprecated

The *Underwater Curtain* will be removed in a future Crest version. It has been replaced by the *Underwater Renderer*.

---

### 12.2.1 Setup

- Configure the ocean material for underwater rendering. In the *Underwater* section of the material params, ensure *Enabled* is turned on and *Cull Mode* is set to *Off* so that the underside of the ocean surface renders. See *Ocean-Underwater.mat* for an example.
- Place *UnderWaterCurtainGeom* and *UnderWaterMeniscus* prefabs under the camera (with cleared transform).

## 12.3 Underwater Post-Process HDRP

---

### Deprecated

The *Underwater Post-Process* will be removed in a future Crest version. It has been replaced by the *Underwater Renderer*.

---

Renders the underwater effect at the beginning of the post-processing stack.

### 12.3.1 Setup

Steps to set up underwater:

1. Ensure Crest is properly set up and working before proceeding.
2. Enable [Custom Pass on the HDRP Asset](#) and ensure that [Custom pass on the camera's Frame Settings](#) is not disabled.
3. Add the custom post-process (*Crest.UnderwaterPostProcessHDRP*) to the *Before TAA* list. See the [Custom Post Process documentation](#).
4. Add the *Crest/Underwater Volume Component*.
  - Please learn how to use the *Volume Framework* before proceeding as covering this is beyond the scope of our documentation:

<https://www.youtube.com/watch?v=vczkfjLoPf8>

Fig. 12.1: Adding Volumes to HDRP (Tutorial)

5. Configure the ocean material for underwater rendering. Ensure that *Double-Sided* is enabled under *Surface Options* on the ocean material so that the underside of the ocean surface renders. See *Ocean-Underwater.mat* for an example.

## TIME CONTROL

By default, *Crest* uses the current game time given by `Time.time` when simulating and rendering the water. In some situations it is useful to control this time, such as an in-game pause or to synchronise wave conditions over a network. This is achieved through what we call *TimeProviders*, and a few use cases are described below.

---

**Note:** The *Dynamic Waves* simulation must progress frame by frame and can not be set to use a specific time, and also cannot be synchronised accurately over a network.

---

### 13.1 Supporting Pause

One way to pause time is to set `Time.timeScale` to 0. In many cases it is desirable to leave `Time.timeScale` untouched so that animations continue to play, and instead pause only the water. To achieve this, attach a *TimeProviderCustom* component to a `GameObject` and assign it to the *Time Provider* parameter on the *OceanRenderer* component. Then time can be paused by setting the `_paused` variable on the *TimeProviderCustom* component to `false`.

The *TimeProviderCustom* also allows driving any time to the system which may give more flexibility for specific use cases.

A final alternative option is to create a new class that implements the *ITimeProvider* interface and call *OceanRenderer.Instance.PushTimeProvider()* to apply it to the system.

### 13.2 Network Synchronisation

A requirement in networked games is to have a common sense of time across all clients. This can be specified using an offset between the clients `Time.time` and that of a server.

This is supported by attaching a *TimeProviderNetworked.cs* component to a `GameObject`, assigning it to the *Time Provider* parameter on the *OceanRenderer* component, and at run-time setting *TimeProviderNetworked.TimeOffsetToServer* to the time difference between the client and the server.

If using the *Mirror* network system, set this property to the `network time offset`.

## 13.3 Timelines and Cutscenes

One use case for this is for cutscenes/timelines when the waves conditions must be known in advance and repeatable. For this case you may attach a *Cutscene Time Provider* component to a `GameObject` and assign it to the *Ocean Renderer* component. This component will take the time from a *Playable Director* component which plays a cutscene *Timeline*. Alternatively, a *Time Provider Custom* component can be used to feed any time into the system, and this time value can be keyframed, giving complete control over timing.

## OTHER FEATURES

### 14.1 Floating Origin

*Crest* has support for ‘floating origin’ functionality, based on code from the *Unity Community Wiki*. See the [original Floating Origin wiki page](#) for an overview and original code.

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.

### 14.2 Buoyancy

---

**Note:** Buoyancy physics for boats is not a core focus of *Crest*. For a professional physics solution we recommend the [DWP2 \(Dynamic Water Physics 2\)](#) asset which is compatible with *Crest*.

With that said, we do provide rudimentary physics scripts.

---

*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.

#### 14.2.1 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 *BoatProbes*.
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 Width* and *Boat Length* params 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 *SphereWaterInteraction*. 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.



## RENDERING

### 15.1 Transparency

*Crest* is rendered in a standard way for water shaders - in the transparent pass and refracts the scene. The refraction is implemented by sampling the camera's colour texture which has opaque surfaces only. It writes to the depth buffer during rendering to ensure overlapping waves are sorted correctly to the camera. The rendering of other transparent objects depends on the case, see headings below. Knowledge of render pipeline features, rendering order and shaders is required to solving incompatibilities.

#### 15.1.1 Transparent Object In Front Of Ocean Surface

Normal transparent shaders should blend correctly in front of the water surface. However this will not work correctly for refractive objects. *Crest* will not be available in the camera's colour texture when other refractive objects sample from it, as the camera colour texture will only contain opaque surfaces. The end result is *Crest* not being visible behind the refractive object.

#### 15.1.2 Transparent Object Behind The Ocean Surface

Alpha blend and refractive shaders will not render behind the water surface. Other transparent objects will not be part of the camera's colour texture when *Crest* samples from it. The end result is transparent objects not being visible behind *Crest*.

On the other hand, alpha test / alpha cutout shaders are effectively opaque from a rendering point of view and may be usable in some scenarios.

#### 15.1.3 Transparent Object Underwater

---

**Note:** See the [Underwater Shader API](#) (in preview) for a more accurate solution.

---

This is tricky because the underwater effect uses the opaque scene depths in order to render the water fog, which will not include transparents.

The following only applies to the *Underwater Renderer*.

##### BIRP

Transparents will need to be rendered after the underwater effect. The underwater effect is rendered at the [CameraEvent.AfterForwardAlpha](#) event. They can be rendered after the underwater effect using [Command Buffers](#). Transparents rendered after the underwater effect will not have the underwater water fog shading applied to them. The effect

of the fog either needs to be faked by simply ramping the opacity down to 0 based on distance from the camera, or the water fog shader code needs to be included and called from the transparent shader.

### HDRP

The Submarine example scene demonstrates an underwater transparent effect - the bubbles from the propellers when the submarine is in motion. This effect is from the *Bubbles Propellor* GameObject, which is assigned a specific layer *TransparentFX*. The particles need to be rendered between the underwater and post-processing passes which is achieved using a *Custom Pass Volume* component attached to the *CustomPassForUnderwaterParticles* GameObject. It is configured to render the *TransparentFX* layer in the *Before Post Process* injection point with a priority of “-1” (which orders it to render after the underwater pass). Transparents rendered after the underwater effect will not have the underwater water fog shading applied to them. The effect of the fog either needs to be faked by simply ramping the opacity down to 0 based on distance from the camera, or the water fog shader code needs to be included and called from the transparent shader. The shader *UnderwaterEffectPassHDRP.shader* is a good reference for calculating the underwater effect. This will require various parameters on the shader like fog density and others.

### URP

Transparents will need to be rendered after the underwater effect. The underwater effect is rendered at the *BeforeRenderingPostProcessing* event. They can be rendered after the underwater effect using the [Render Objects Render Feature](#) set to *BeforeRenderingPostProcessing*. Transparents rendered after the underwater effect will not have the underwater water fog shading applied to them. The effect of the fog either needs to be faked by simply ramping the opacity down to 0 based on distance from the camera, or the water fog shader code needs to be included and called from the transparent shader.

## 15.2 Render Order *BIRP URP*

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 renders early in the transparent queue (queue = 2510).
  - Queue = Geometry+510 *BIRP*. Queue = Transparent-100 *URP*.
  - 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 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.
  - If enabled, underwater postprocess constructs a screenspace mask for the ocean and uses it to draw the underwater effect over the screen.

## 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.

### 16.1 Quality parameters

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

- See *Ocean Construction Parameters* 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 *Material Parameters*). A potential idea would be to change materials on the fly from script, for example to switch to a deep water material when out at sea to avoid shallow water calculations
- Our wave system uses an inefficient approach to generate the waves to avoid an incompatibility in older hardware. If you are shipping on a limited set of hardware which you can test the waves on, you may try disabling the *Ping pong combine* option in the *Animated Wave Settings* asset.



## TECHNICAL DOCUMENTATION

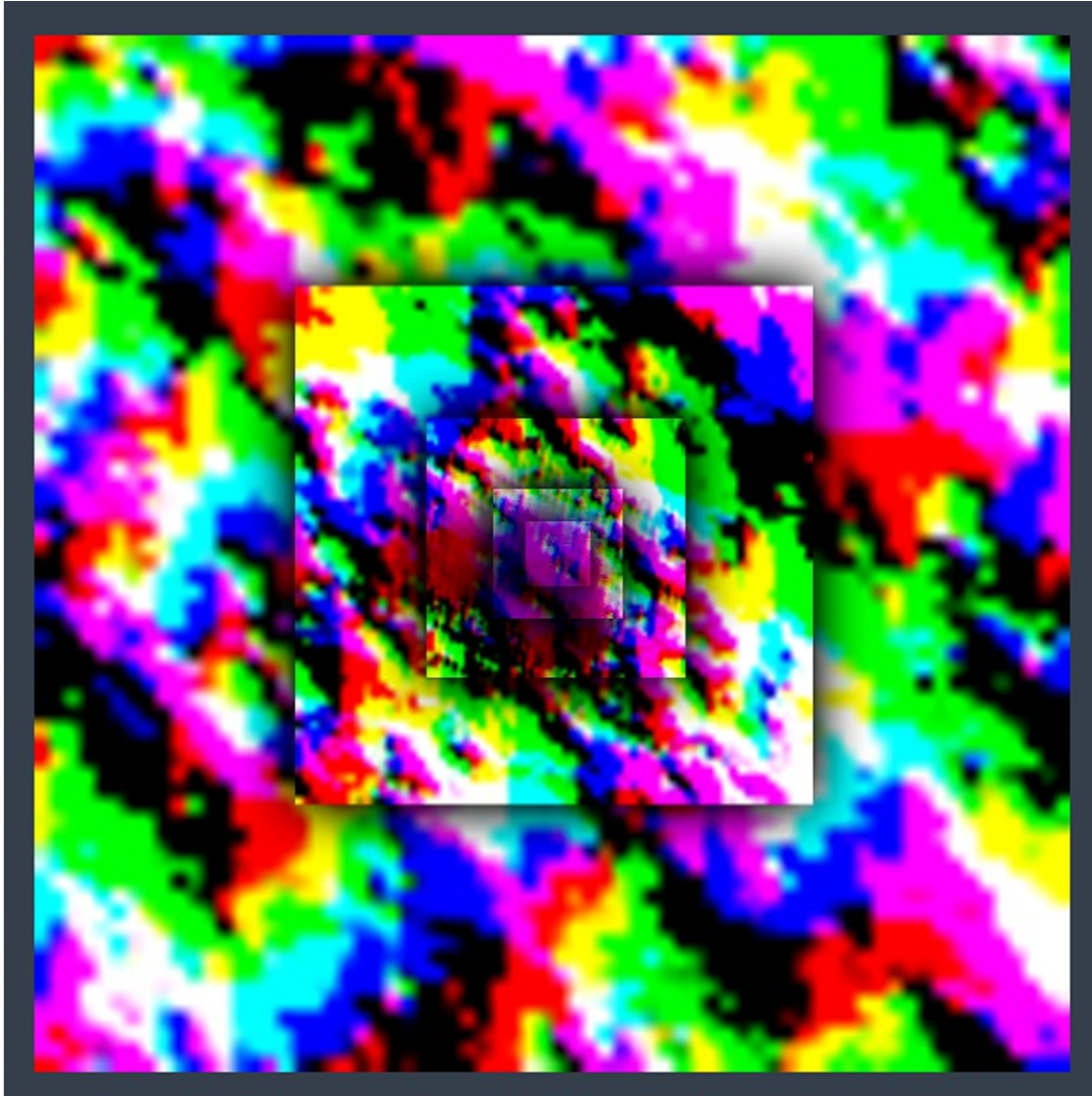
We have published details of the algorithms and approaches we use. See the following publications:

- 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>

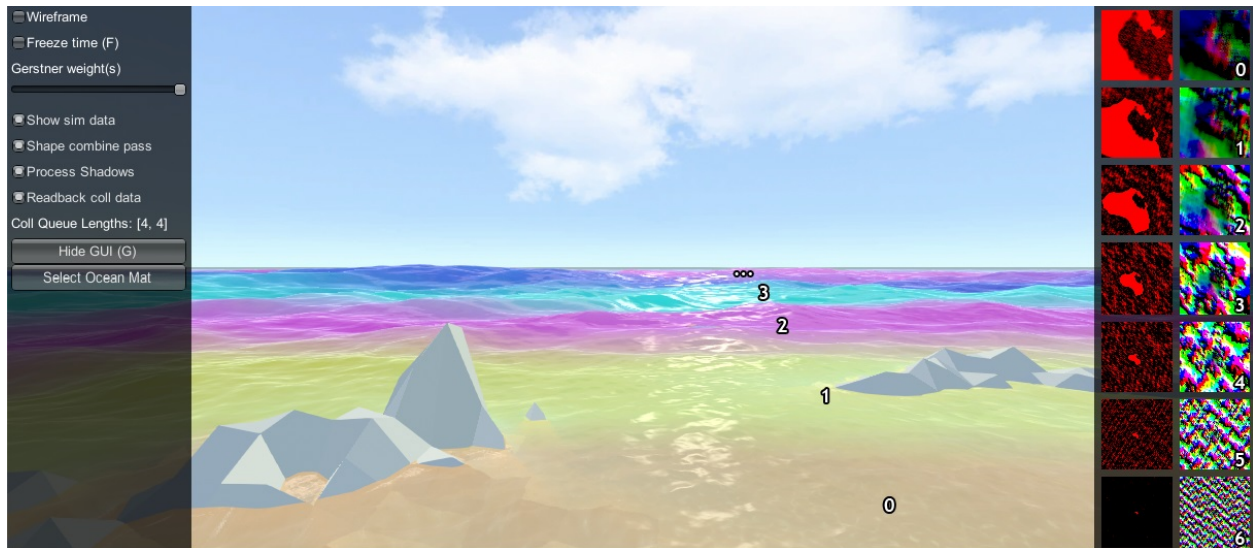
### 17.1 Core Data Structure

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:

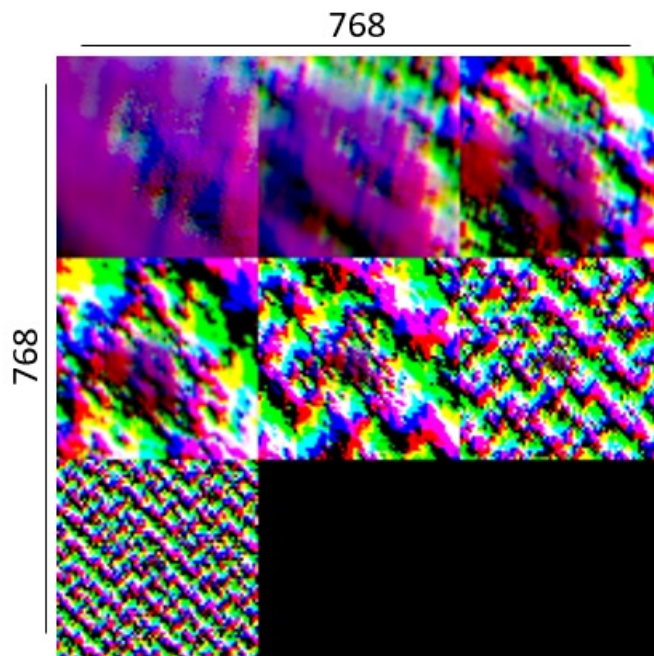


Each LOD is the same resolution (256x256 here), configured on the *OceanRenderer* script. In this example the largest LOD covers a large area (4km squared), and the most detail LOD provides plenty of resolution close to the viewer. These textures are visualised in the Debug GUI on the right hand side 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 #96.

## 17.2 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 *OceanDebugGUI* 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 using an FFT and copied 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 FFT 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 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.

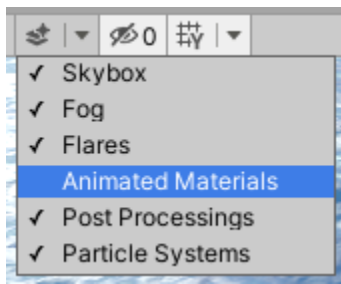
The fragment shader samples normal and foam maps at 2 different scales, both proportional to the current and next LOD scales, and then interpolates the result using *lodAlpha* for a smooth transition. It combines the normal map with surface normals computed directly from the displacement texture.



## Q & A

### Why does the ocean not update smoothly in edit mode?

By default, the update speed is intentionally throttled by Unity to save power when in edit mode. To enable real-time update, enable *Animated Materials* in the Scene View toggles:



See the [Unity Documentation](#) for more information.

### 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 the following platforms:

- Windows
- Mac
- Linux
- PS4 \*\*
- XboxOne \*\*
- Switch \* \*\*
- iOS/Android \* \*\* *BIRP URP*

- Quest \* \*\* *BIRP URP*

\* Performance is a challenge on these platforms. Please see the previous question.

\*\* We do not have access to these platforms ourselves.

*Crest* also supports VR/XR Multi-Pass and Single Pass Instanced rendering.

For additional platform notes, see [Platform Support](#).

## Is *Crest* well suited for localised bodies of water such as lakes?

Currently *Crest* is targeted towards large bodies of water. This area is being actively developed. Please see [Oceans](#), [Rivers and Lakes](#) for current progress.

## Can *Crest* work with multiplayer?

Yes, the animated waves are deterministic and easily synchronized. See discussion in [#75](#). However, the dynamic wave sim is not synchronized over the network and can not currently be relied upon in networked situations.

For more information, see the following sections relevant to networking and server environments:

- [Network Synchronisation](#)
- [Baked FFT Data \(CPU\)](#)
- [Gerstner Waves CPU \(deprecated\)](#)

## Errors are present in the log that report *Kernel ‘xxx.yyy’ not found*

Unity sometimes gets confused and needs assets reimported. This can be done by clicking the *Crest* root folder in the Project window and clicking *Reimport*. Alternatively the *Library* folder can be removed from the project root which will force all assets to reimport.

## Can I push the ocean below the terrain?

Yes, this is demonstrated in [Fig. 7.1](#).

## Does *Crest* support multiple viewpoints?

Currently only a single ocean instance can be created, and only one viewpoint is supported at a time. We hope to support multiple simultaneous views in the future.

## Can I sample the water height at a position from C#?

Yes, see `SampleHeightHelper` class in `SamplingHelpers.cs`. The `OceanRenderer` uses this helper to get the height of the viewer above the water, and makes this viewer height available via the `ViewerHeightAboveWater` property.

## Can I trigger something when an object is above or under the ocean surface without any scripting knowledge?

Yes. Please see *Detecting Above or Below Water*.

## Does Crest support orthographic projection?

Yes. Please see *Orthographic Projection*.

## How do I disable underwater fog rendering in the scene view?

You can enable/disable rendering in the scene view by toggling fog in the `scene view control bar`.

## Can the density of the fog in the water be reduced?

The density of the fog underwater can be controlled using the *Fog Density* parameter on the ocean material. This applies to both above water and underwater. The *Depth Fog Density Factor* on the *Underwater Renderer* can reduce the density of the fog for the underwater effect.

## Does Crest support third party sky assets? *BIRP URP*

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'.

Please see the `Community Contributions` section in our [Wiki](#) for some integration solutions.

## Can I remove water from inside my boat?

Yes, this is referred to as 'clipping' and is covered in section *Clip Surface*.

## How to implement a swimming character?

As far as we know, existing character controller assets which support swimming do not support waves (they require a volume for the water or physics mesh for the water surface). We have an efficient API to provide water heights, which the character controller could use instead of a physics volume. Please request support for custom water height providers to your favourite character controller asset dev.

## Can I render transparent objects underwater?

See *Transparent Object Underwater*.

## Can I render transparent objects in front of water?

See *Transparent Object In Front Of Ocean Surface*.

## Can I render transparent objects behind the ocean surface?

See *Transparent Object Behind The Ocean Surface*.