Crest

Release 5.6.3

Wave Harmonic

Sep 21, 2025

TABLE OF CONTENTS

Part I

About

CHAPTER

ONE

INTRODUCTION

Crest is a technically advanced water system for Unity.

Architectured for performance, it utilises Level Of Detail (LOD) strategies and GPU acceleration to ensure fast updates and rendering. Crest is also highly flexible, allowing for custom inputs to LODs (such as shape or foam) and featuring an intuitive shape authoring interface.

This documentation is for Crest 5.6.3.

You can view the living documentation online.

1.1 Assets

Crest 5, and several assets which extend it, are available on the Unity Asset Store:

- Crest 5: Asset Store
- CPU Queries: Asset Store
- Paint: Asset Store
- Portals: Asset Store
- Shallow Water: Asset Store
- Shifting Origin: Asset Store
- Splines: Asset Store
- Whirlpool: Asset Store

You can also find *Crest 4* on the *Unity Asset Store* for all render pipelines.

- Crest Water 4 BIRP: Asset Store
- Crest Water 4 HDRP: Asset Store
- Crest Water 4 URP: Asset Store

1.2 Social

- **Website** https://waveharmonic.com
- **Asset Store** https://assetstore.unity.com/publishers/41652
- **YouTube** https://www.youtube.com/@WaveHarmonic
- **Discord** https://discord.gg/JJjx9qcq83
- **X** https://x.com/WaveHarmonicX

CHAPTER

TWO

KNOWN ISSUES

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

- **Issues on GitHub.**
  - BIRP (Built-in Render Pipeline) specific issues on GitHub.
  - HDRP (High Definition Render Pipeline) specific issues on GitHub.
  - URP (Universal Render Pipeline) 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. A list of these issues can be found on our wiki which can be voted on for Unity to prioritise.

As a reminder it is important to always use the latest patched Unity version.

2.2 Prefab Mode Not Supported

Crest does not support running in prefab mode which means dirty state in prefab mode will not be reflected in the scene view. Save the prefab to see the changes.

CHAPTER

THREE

RELEASE NOTES

5.6.3

Fixes

- Fix outline around objects when underwater (except when downsampling is enabled)
- Fix waterline when using viewpoint
- Fix planar TIR (Total Internal Reflection)
- Fix OnGUI editor allocations
- Fix editor-only per-frame allocations
- Fix "_ShadowMapTexture not set" on first frame *BIRP*
- Fix Write Depth option in builds *BIRP*
- Fix missing Water Body validation
- Reduce cases of underwater flickering at high speeds
- Fix *Underwater Environmental Lighting* ignoring *Out-Scattering Factor*
- Fix cases where *Underwater Environmental Lighting* sun attenuation would persist
- Fix Unity 6.3 "Serializable" warnings
- Fix provided meniscus material lighting/refraction options in builds

Optimizations

- Several minor per-frame optimizations
- Reduce Debug GUI per-frame allocations

Documentation

- Add subheading for the Water Body component

5.6.2

Fixes

- Fix exception on recompile if there is no project settings file present

Optimizations

- Use newer profiler marker API

5.6.1

Fixes

- Fix compilation errors if Portals package is not present

5.6.0

With this release we redesigned the underwater effect with a different approach which improves performance, supports a much nicer meniscus, and solves rough/choppy waves triggering the underwater effect above water (a long-standing quality issue).

A toggle has been added to the project settings at *Project Settings* $\rightarrow$ *Crest* $\rightarrow$ *Legacy Underwater* to revert to the previous underwater effect (hereby referred to as legacy) if needed. Several of the fixes below also apply to the legacy underwater effect.

Changes

- Add a more prominent meniscus effect with thickness in world units, and new visuals including a refraction effect
- Move meniscus into separate foldout on the Water Renderer for configuration
- Render underwater before the water surface (and before transparency pass) to improve performance. As a side effect, transparent object no longer receive underwater fog. This excludes the legacy underwater effect
- No longer limit features in batch mode, as it is not relevant to Crest
- Update water surface material defaults to improve reflections and sub-surface scattering
- Validate that underwater material has correct shader

Fixes

- Fix underwater being triggered by choppy waves. This was a long standing quality issue that preventing anyone taking full advantage of our wave system
- Fix meniscus size not being consistent
- Fix water surface above/below water rendering not matching underwater effect
- Fix underwater effect being incorrect due to not appling near plane to distance calculation
- Fix meniscus appearing at screen edge
- Fix several null exceptions when changing render pipeline
- Fix dynamic waves duplicating across LODs when resolution is different from Animated Waves
- Fix some potential issues due to an important uniform not being set as an integer from script
- Fix "truncation" warnings from shadow simulation shader
- Fix color/depth write feature not working *BIRP*
- Fix null exception if there is a terrain with empty data
- Fix some Shader Graph inputs not working in builds *HDRP*
- Fix ugly artifacts caused by *Minimum Reflection Direction Y* not applying to normal maps
- Fix waves potentially not working if there were multiple wave components in the scene
- Fix unnecessary Depth Probe validation logging if populating via script
- Reduce spurious validation in no-graphics mode
- Fix console spam if underwater material has incorrect shader

Optimizations

- Greatly improve performance when underwater is enabled by elimating a pre-pass on the water surface
- Reduce custom screen-sized textures requirements saving performance and memory for underwater effect
- Convert shadow simulation shader to compute which eliminates several draw calls
- Optimize several shader render states
- Reduce work done in no-graphics mode
- Reduce number of subscribers to render pipeline events
- Skip discard fog path completely for transparent objects above water

Full version history has been omitted for brevity. It can be found at Release Notes.

Part II

Getting Started

CHAPTER

FOUR

INITIAL SETUP

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

4.1 Requirements

- Unity 2022.3 or later, latest patched version
- *Shader Graph* package
- Shader compilation target 4.5 or above
- Graphics API that is **not** OpenGL or WebGL

Important: It may be necessary to update to the latest patched version of Unity in order for a problem to be solved. Furthermore, legacy versions of Unity are not supported.

Sample scenes when using BIRP uses the post-processing package. If this is not present in your project, you will see an unassigned script warning which you can fix by removing the offending script. Furthermore, scenes will look overexposed.

4.2 Migrating from *Crest 4*

Please see the *Migration* page.

4.3 Importing Crest files into project

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

4.3.1 Render Pipeline Setup

Ensure that your chosen render pipeline is setup and functioning, either by setting up a new project using the appropriate template or by configuring your current project. This is beyond the scope of this documentation so please see the Unity documentation (BIRP, HDRP, URP) 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.

Tip: If you are starting from scratch we recommend creating a project using a template in the Unity Hub.

4.3.2 Importing Crest

Import the Crest package into the project using the Package Manager window in the Unity Editor.

Crest is a UPM package and thus is imported into the Packages directory. Once imported Crest packages will be listed in the package manager window. Familiarity with the package details pane is important as this is where Samples are imported from.

4.3.3 Importing Sample Content

Sample content is hidden in the file system, and requires importing from the package manager window. The *Importing Crest* step needs to be done first.

1. Open *Window → Package Manager*
2. Set the packages menu to *Packages: In Project* (see C below)
3. Select the Crest package you are interested in from the package list under the **Wave Harmonic** group (see H below). If using Unity 6, Crest will show up twice in the menu, make sure to select the package under the **Wave Harmonic** group towards the bottom.
4. In the main window that shows the package's details, find the Samples section (see J below)
5. To import a Sample into your Project, click the *Import into Project* button. This creates a Samples folder in your Project and imports the Sample you selected into it. This is also where Unity imports any future Samples into.

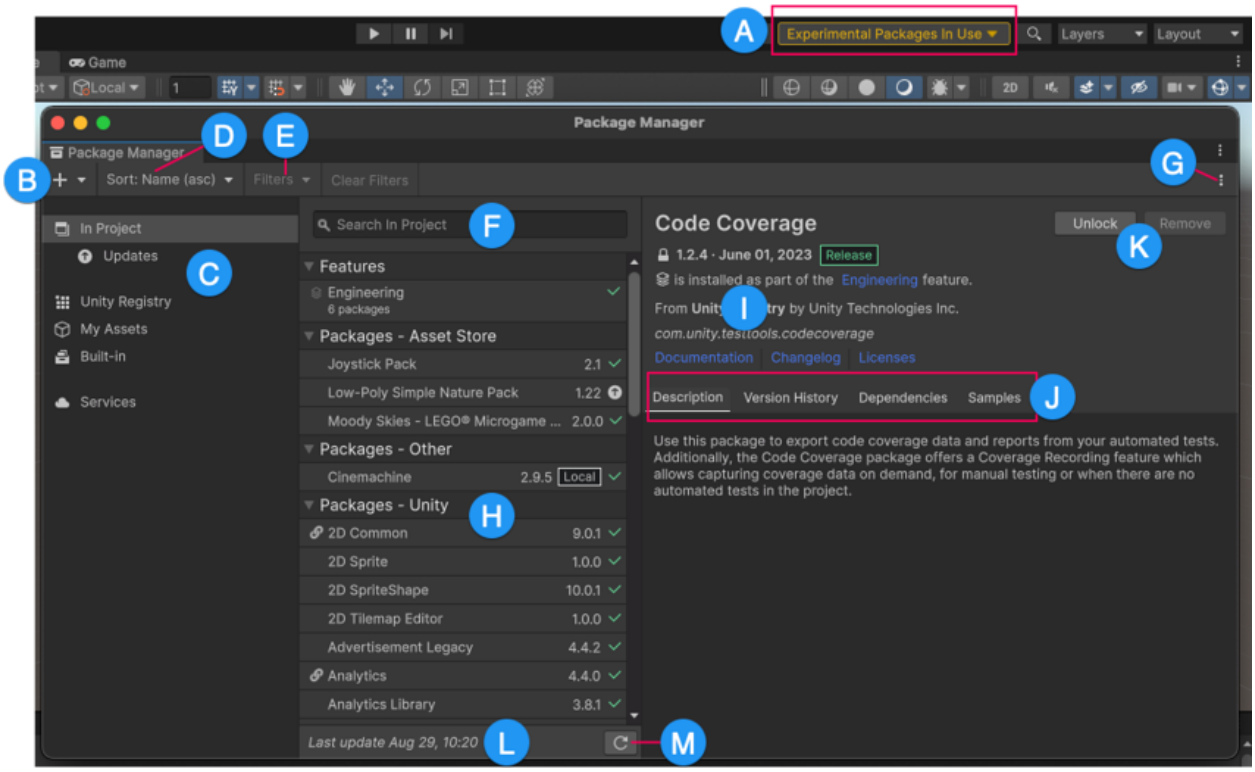


Fig. 4.1: Package Manager window

4.4 Adding Crest to a Scene

The default use case for Crest is an infinite ocean. The steps to adding an ocean to a scene are as follows:

- Create a new *Game Object* for the ocean.
  - Assign the *Water Renderer* component to it. This component will generate the water geometry and do all required initialisation.
  - 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 *WaterRenderer* script. If you need to switch between multiple cameras, update the *ViewCamera* field to ensure the water follows the correct view.
- Be sure to generate lighting if necessary. The water lighting takes the ambient intensity from the generated spherical harmonics regardless of whether you use baked or realtime lighting. 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 *Waves* section for customisation.

Continue on to the *Quick Start Guide*.

4.5 Frequent Setup Issues

The following are issues with the install process which come up frequently.

I am seeing errors in the console and/or visual issues

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 water rendering, magenta materials, scripts unassigned in example scenes, etcetera. Often, restarting the Editor fixes it. Additionally, re-importing broken (ie magenta) materials/shaders may be required. 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.

I am seeing magenta materials after changing render pipelines

Unity has a bug where sometimes it will not correctly switch materials to the current render pipeline. Solutions can vary:

- Restarting Unity
- View the affected material's inspector
- Reimport the affected material/shader

This affects Shader Graph specifically. Find our water Shader Graph, right click and re-import it. It may appear as a blank file icon instead of the usual Shader Graph icon.

I can 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 *Packages/Crest/Runtime/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.

Why aren't my prefab mode edits not reflected in the scene view?

Crest does not support running in prefab mode which means dirty state in prefab mode will not be reflected in the scene view. Save the prefab to see the changes.

I am seeing "Crest does not support OpenGL/WebGL backends." in the editor

It is likely Unity has defaulted to using OpenGL on your platform. You will need to switch to a supported graphics API like Vulkan. You will need to make Vulkan the default by overriding the graphics APIs.

Why I am seeing "The referenced script on this Behaviour is missing!" or similar in Crest's sample scenes and prefabs?

This is normal and can be ignored. The sample scenes support all render pipelines which require render pipeline specific components to be serialized in the scene. If a render pipeline package is missing, then those components will also be missing.

CHAPTER

FIVE

MIGRATION

5.1 Migrating from *Crest 4*

Unfortunately, there is no migration path from Crest 4 to 5. This is due to merging three assets together, moved and renamed files, changes in GUIDs and huge changes to serialized data.

Since all GUIDs and file paths have changed, it is possible to import and run both Crest 4 and 5 in one project for manual migration.

When Crest 4 is installed, it adds CREST_OCEAN scripting symbol. When this symbol is present, all Crest 5 components are prefixed with *Crest 5*.

The following tables document items that have been renamed, removed or changed and their new counterpart.

Table 5.1: Components

Crest 4	Crest 5
BoatAlignNormal	FloatingObject
BoatProbes	FloatingObject
OceanDebugGUI	DebugGUI
OceanDepthCache	DepthProbe
OceanPlanarReflections	WaterRenderer[1]
OceanRenderer	WaterRenderer
OceanSampleHeightEvents	QueryEvents
OceanWaterInteraction	N/A[1]
OceanWaterInteractionAdapter	N/A[1]
RegisterAlbedoInput	AlbedoInput
RegisterAnimatedWavesInput	AnimatedWavesInput
RegisterClipSurfaceInput	ClipInput
RegisterDynamicWavesInput	DynamicWavesInput
RegisterFlowInput	FlowInput
RegisterFoamInput	FoamInput
RegisterHeightInput	WaterLevelInput
RegisterSeaFloorDepthInput	WaterDepthInput
RegisterShadowInput	ShadowInput
RegisterAlphaOnSurface	N/A[1]
ShapeGerstnerBatch	N/A[1]
SimpleFloatingObject	FloatingObject
UnderwaterEffect	N/A[1]
UnderwaterEnvironmentalLighting	WaterRenderer[2]
UnderwaterRenderer	WaterRenderer[2]
VisualiseCollisionArea	CollisionAreaVisualizer
VisualiseRayTrace	RayCastVisualizer

[1] Removed

[2] Merged into

Table 5.2: Shaders

Crest 4	Crest 5
CopyDepthBufferIntoCache	N/A[1]
Inputs/AnimatedWaves/GerstnerBatchGeometry	N/A[1]
Inputs/ShapeWaves/SampleSpectrum	Inputs/Waves/AddFromGeometry
Inputs/DynamicWaves/ObjectInteraction	N/A[1]
Inputs/Flow/AddFlowMap	Inputs/Flow/AddFromTexture
Inputs/Foam/AddFromVertColours	Inputs/Foam/AddFromVertexColors
Inputs/Foam/OverrideFoam	Inputs/All/Override
Inputs/All/ScaleByFactor	Inputs/All/Scale
Inputs/AnimatedWaves/GerstnerGeometry	Inputs/ShapeWaves/AddFromGeometry
Inputs/AnimatedWaves/SetBaseWaterHeightUsingGeom	N/A[1]
Inputs/ClipSurface/ConvexHull	WatertightHull[2]
Inputs/ClipSurface/IncludeArea	Inputs/All/Override
Inputs/ClipSurface/RemoveArea	Inputs/All/Override
Inputs/ClipSurface/RemoveAreaTexture	N/A[3]
Inputs/SeaFloorDepth/SetBaseWaterHeightUsingGeomet	Inputs/Level/WaterLevelFromGeometry
Inputs/Depth/OceanDepthFromGeometry	Inputs/Depth/WaterDepthFromGeometry
Inputs/Depth/CachedDepths	N/A[3]
Inputs/Shadows/OverrideShadows	Inputs/All/Override
Inputs/AnimatedWaves/Whirlpool	Whirlpool[4]
Inputs/Flow/Whirlpool	Whirlpool[4]
Ocean	Water
OceanSurfaceAlpha	N/A[1]
UnderwaterCurtain	N/A[1]
UnderwaterMeniscus	N/A[1]

[1] Removed

[2] Replaced with component

[3] Replaced with *Texture Mode*

Table 5.3: Scripting API

Crest 4	Crest 5
SampleHeightHelper	SampleCollisionHelper

Crest 4 never had an official API, thus only the most important changes have been listed. Furthermore, our namespace has changed from *Crest* to *WaveHarmonic.Crest*.

CHAPTER

SIX

QUICK START GUIDE

This section provides a summary of common steps for setting up water with links for further reading:

- **Add water**: Add Crest water to your scene as described in section *Adding Crest to a Scene*.
- **Water surface appearance**: The active water material is displayed below the *Water Renderer* component. See *Water Appearance* for documentation on water appearance.
- **Add Waves**: Add *Shape FFT* component to a *Game Object* and assign a *Wave Spectrum* asset. Waves can be generated everywhere, or in specific areas. See section *Waves*.
- **Shallow Water**: Reduce waves in shallow water. See section *Shorelines and Shallows*.
- **Oceans, Rivers and Lakes**: Crest supports setting up networks of connected water bodies. See section *Water Bodies*.
- **Underwater**: If the camera needs to go underwater, the underwater effect must be enabled. See section *Underwater*.
- **Dynamic wave simulation**: Simulates dynamic effects like object-water interaction. See section *Dynamic Waves*.
- **Watercraft**: Several components combined can create a convincing watercraft. See page *Floating Objects*.
- **Networking**: Crest is built with networking in mind and can synchronise waves across the network. It also has limited support for headless servers. See section *Network Synchronisation*.
- **Open Worlds**: Crest comes with "shifting origin" support to enable large open worlds. See section *Shifting Origin*.

CHAPTER

SEVEN

FREQUENTLY ASKED QUESTIONS

Where is the sample content?

Please see *Importing Sample Content*.

How can I migrate from Crest 4?

Please see *Migrating from Crest 4*.

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

Yes, see *Water Bodies* for documentation.

Can I push the water below the terrain?

Yes, this is demonstrated in Fig. 10.1.

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

Yes, see /API/WaveHarmonic.Crest/SampleCollisionHelper, and for more context and examples, see *Queries*. The *WaterRenderer* 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 water surface without any scripting knowledge?

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

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 water 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.

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 Water Surface*.

Can I render transparent objects behind the water surface?

See *Transparent Object Behind The Water Surface*.

7.1 Compatibility

Which platforms does Crest support?

Testing occurs primarily on MacOS and Windows. iOS and Android are also periodically tested.

Firstly, make sure your target platform adheres to the *Requirements*.

We have users targeting the following platforms:

- Windows
- MacOS
- Linux *
- PlayStation *
- Xbox *
- Switch * **
- iOS **
- Android/Quest **

* We do not have access to these platforms ourselves.
** Performance is a challenge on these platforms. Please see the previous question.

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

For additional platform notes, see Platform Support.

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 water solution such as the one in the *Boat Attack* project. Therefore we target Crest at PC/console platforms.

That being said, developers have had success with Crest on lower powered platforms like the *Nintendo Switch*. *Apple* devices are good targets as their processors are quite capable all round. *Meta Quest 2* is one device where it will be a struggle to take advantage of Crest.

Can Crest work with networked multiplayer?

Yes, the animated waves are deterministic and can be synchronised across the network. For more information see *Network Synchronisation*.

Note however that the dynamic wave simulation is not synchronized over the network and should not be relied upon in networked situations.

Does Crest support multiple viewpoints like for split-screen multiplayer?

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

Providing both both players are near each other, split-screen multiplayer can be accomplished by setting the viewpoint between them. Your mileage may vary depending on your settings and expectations.

Does Crest support third-party assets?

Please see /About/Integrations for third-party support.

Does Crest support orthographic projection?

Yes. Please see orthographic_projection.

Does Crest support motion vectors?

Yes. Please see motion-vectors.

Does Crest support Depth of Field?

Yes. Please see post-processing.

Does Crest support Soft Particles?

Yes. Please see soft-particles.

7.2 Troubleshooting

I am seeing errors in the console and/or visual issues

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 water rendering, magenta materials, scripts unassigned in example scenes, etcetera. Often, restarting the Editor fixes it. Additionally, re-importing broken (ie magenta) materials/shaders may be required. 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.

I am seeing magenta materials after changing render pipelines

Unity has a bug where sometimes it will not correctly switch materials to the current render pipeline. Solutions can vary:

- Restarting Unity
- View the affected material's inspector
- Reimport the affected material/shader

This affects Shader Graph specifically. Find our water Shader Graph, right click and re-import it. It may appear as a blank file icon instead of the usual Shader Graph icon.

I can 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 *Packages/Crest/Runtime/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.

Why aren't my prefab mode edits not reflected in the scene view?

Crest does not support running in prefab mode which means dirty state in prefab mode will not be reflected in the scene view. Save the prefab to see the changes.

I am seeing "Crest does not support OpenGL/WebGL backends." in the editor

It is likely Unity has defaulted to using OpenGL on your platform. You will need to switch to a supported graphics API like Vulkan. You will need to make Vulkan the default by overriding the graphics APIs.

Why I am seeing "The referenced script on this Behaviour is missing!" or similar in Crest's sample scenes and prefabs?

This is normal and can be ignored. The sample scenes support all render pipelines which require render pipeline specific components to be serialized in the scene. If a render pipeline package is missing, then those components will also be missing.

Part III

Basics

CHAPTER

EIGHT

WATER APPEARANCE

8.1 Material

If you need to change the absorption value via script, then do the following:

```
var material = WaterRenderer.Instance.Material;
var color = Color.red; // Replace with your color.
material.SetColor(Shader.PropertyToID("_Crest_AbsorptionColor"), color);
material.SetVector(Shader.PropertyToID("_Crest_Absorption"), WaterRenderer.
↪CalculateAbsorptionValueFromColor(color));
```

8.2 Lighting

As other shaders would, the water will get most its lighting from the primary directional light (ie sun or moon).

See *Chunk Template* for advanced configuration.

8.2.1 Reflections

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

Crest uses Shader Graph which makes it by default ready to receive all the various sources of reflections Unity provides.

Furthermore, Crest provides Planar Reflections via the Reflections foldout on the Water Renderer for both above and below the water (known as TIR).

8.2.2 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 Water Surface* for issues with Crest and other refractive materials.

8.2.3 Chunk Template

Crest uses Mesh Renderers to render chunks of water throughout the scene. Mesh Renderers have many settings for configuring how a mesh responds to different lighting components like probes.

The Chunk Template allows you to provide a pre-configured Mesh Renderer to override Crest's defaults. Some settings cannot be overriden as they make no sense or require support from Crest.

To use this feature, create a prefab with a Mesh Renderer. The only other requirement is to **not** have a Water Chunk Renderer present in the prefab.

Tip: Clicking "New" next to *Water Renderer* → *Surface Renderer* → *Chunk Template* will create a new one for you.

Settings which cannot be overriden are:

- Receive Shadows
- Motion Vectors

Other settings may not have any effect depending on the level of support with the render pipeline.

Alternatively, there is an event which is raised after chunk creation: /API/WaveHarmonic.Crest/SurfaceRenderer/OnCreateChunkRendere

8.3 Foam

8.3.1 Overview

Crest simulates foam generation by choppy water (ie *pinched* wave crests) and in shallow water to approximate foam from splashes at the 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 *Water Renderer* → *Simulations* → *Foam* → *Enabled*.

To configure the foam simulation, create a Foam Lod Settings with *Assets* → *Create* → *Crest* → *Simulation Settings* → *Foam Sim Settings*, and assigning it to the Water Renderer component in your scene.

Tip: Clicking "New" next to *Water Renderer* → *Simulations* → *Foam* → *Settings* will create a new one for you.

8.3.2 User Inputs

Crest supports inputting foam data into the system, which can be helpful for fine tuning where foam is placed.

The Foam Input component can write data to the simulation and supports the *Texture Mode*, *Spline Mode*, *Paint Mode* and *Renderer Mode*. The *Texture Mode* is the most efficient mode.

The following shaders are available under *Crest/Inputs/Foam* if using the *Renderer Mode*:

- **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 simulation. This is an alternative to the *Texture Mode* as it can provide more properties to adjust.
- **Add From Vertex Colors** can be applied to geometry and uses the red channel of vertex colours to add foam to the simulation. Similar in purpose to *Add From Texture*, but can be authored in a modelling workflow instead of requiring a texture.

8.3.3 Simulation Settings

Simulation properties are covered with tooltips. This are overviews of each category.

Whitecaps: Crest detects where waves are 'pinched' and deposits foam to approximate whitecaps. **Shoreline Foam**: If water depth input is provided to the system (see *Depth Simulation*), the foam simulation can automatically generate foam when water is very shallow, which can approximate accumulation of foam at shorelines.

8.4 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 approximate 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 simulation then reads these shadow maps and copies shadow information into its LOD textures.

To turn on this feature, enable *Water Renderer* → *Simulations* → *Shadows* → *Enabled*.

8.4.1 User Inputs

The Shadow Input component can write data to the simulation and supports the *Renderer Mode*. There are currently no shadow specific shaders, but the general purpose shaders under *Crest/Inputs/All* will accomplish most things (especially Override).

8.4.2 Simulation Settings

The shadow simulation can be configured under *Water Renderer* → *Simulations* → *Shadows*.

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 water 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 *Shadow Input* component which can override the shadow data. This component bypasses jittering and gives you full control.

8.5 Absorption & Scattering

8.5.1 Overview

The Absorption and Scattering targets allows changing the water volume color at the texel level. These are separate simulations, but are often used together.

They both contain depth-based absorption/scattering, which can simulate suspended matter, dissolved matter and other scattering elements close to shore.

8.5.2 User Inputs

The Absorption/Scattering Input component can write data to the simulation and supports the *Texture Mode*, *Spline Mode*, *Paint Mode* and *Renderer Mode*.

Texture

Be awary that the absorption simulation stores a computed absorption value, not the absorption color you will see on materials and scripts. This is an issue when using this mode, as the input texture needs to be in the computed value. To calculate the absorption value, see /API/WaveHarmonic.Crest/WaterRenderer/CalculateAbsorptionValueFromColor.

Renderer

The following shaders are available under *Crest/Inputs/Absorption* and *Crest/Inputs/Scattering* if using the *Renderer Mode*:

- **Color** a basic shader for writing a color.

8.6 Custom Albedo

8.6.1 Overview

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

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

Note: HDRP has a Decal Projector feature that works with the water shader, 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.

8.6.2 User Inputs

The Albedo Input component can write data to the simulation and supports the *Renderer Mode*.

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

The following shaders are available under *Crest/Inputs/Albedo* if using the *Renderer Mode*:

- **Color** a basic shader for writing a color or texture to the Albedo data including blend modes.

Note: If using a shader with multiple passes (or Shader Graph), if the output looks incorrect then change the Shader Pass Index.

8.7 Level of Detail

There are a small number of parameters that control the construction of the water shape and geometry. All the information is in tooltips under *Water Renderer* → *Level of Detail*.

CHAPTER

NINE

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.

For performance reasons, the underwater effect is disabled if the viewpoint is not underwater.

Tip: Use opaque or alpha test materials for underwater surfaces. Transparent materials do not receive the underwater effect. This can be solved with *Integrate Water Volume (Shader Graph)*.

9.1 Underwater Renderer

The Underwater Renderer is built into the Water Renderer and executes a fullscreen underwater effect before the transparent pass. Transparent objects by default do not receive the underwater effect.

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

9.1.1 Setup

- Configure the water material for underwater rendering. Under *Surface Options* make sure both sides are enabled with either *Render Face* to *Both* or *Double-Sided* enabled.
- Enable *Crest Water* → *Underwater* → *Enabled*.
- Make sure that *Crest* is set to the underwater material.

Note: If you are using the underwater effect in URP, 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.

9.1.2 Appearance

The underwater effect gets its appearance from two sources: the water material and the underwater material.

The water material can be overriden by setting *Water Renderer* → *Volume Material* to a material variant of the water material. Not every property on the water material affects the underwater effect. To see what can be overriden, see the read-only properties on the underwater material.

The underwater material has more properties which can further adjust the underwater effect.

9.1.3 Integrate Water Volume (Shader Graph)

Transparent objects do not receive the underwater effect due to the effect rendering before the transparent pass. We provide a Shader Graph node, *Integrate Water Volume*, to conditionally apply underwater fog.

Additionally, this can exclude Unity's own fog when underwater (for transparent and opaque).

9.2 Meniscus Renderer

The Meniscus Renderer is integrated into the Water Renderer under the Meniscus foldout. It renders a meniscus effect where the surface intersects the near plane (or in case of portals, the edge of the portal).

To configure the meniscus effect, there are options under foldout, and the material inspector appears at the bottom of the Water Renderer component.

9.3 Detecting Above or Below Water

The Water Renderer component has the *Viewer Height Above Water* property which can be accessed with `WaterRenderer.Instance.ViewerHeightAboveWater`. It will return the signed height from the water surface of the camera rendering the water.

There is also the *Query Events* component which uses UnityEvents to provide a scriptless approach to triggering changes.

9.4 Portals & Volumes

Underwater rendering can be restricted to a mesh.

To help with this feature, the Portals package is available as a separate purchase and its offline manual is available in the package.

9.5 Underwater Only

The underwater effect can render without the surface to save performance. Simply disable *Water Renderer* → *Surface* → *Enabled*.

Furthermore, disable all of the simulations to save performance further.

9.6 Legacy Underwater

For those who need the older implementation of the underwater/meniscus effect, enable *Project Settings* → *Crest* → *Legacy Underwater*. It may take a minute for recompilation to begin.

The legacy effect typically has worst performance, and there should not be a need to use it. It renders after the transparent pass, which means transparents will receive the underwater effect without an integration, albeit incorrectly.

CHAPTER

TEN

WATER INPUTS

Inputs provides a means for developers to control the various simulations powering Crest. The following video covers the basics:

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

Fig. 10.1: Crest 4: Basics of Adding Inputs (outdated but still useful)

10.1 Inputs

Each simulation target has an associated generic input:

- *Albedo Input*
- *Animated Waves Input*
- *Clip Input*
- *Dynamic Waves Input*
- *Flow Input*
- *Foam Input*
- *Shadow Input*
- *Water Depth Input*
- *Water Level Input*

There are also inputs which are more specific like the Sphere Water Interaction.

10.2 Input Modes

Inputs (eg Flow Input) can have multiple authoring modes. Support for modes will vary across inputs. More complicated inputs will have their own component like the Sphere Water Interaction.

10.2.1 Geometry Mode

This mode is the same as using the *Renderer Mode* with a Mesh Renderer, except much easier to set up. All you need to provide is a mesh and change the Transform to your needs.

10.2.2 Global Mode

Global is applied everywhere.

10.2.3 Texture Mode

This mode uses a compute shader to apply your texture to the target simulation. It performs better than shader equivalents by reducing draw calls as typically a shader needs to do one draw call per LOD (eg if LOD Count is seven then that is seven draw calls per input).

This a great utility for customization and integration with Crest, as it abstracts away the complexity of the system if you treat the texture as an intermediary between your own simulations/effects and Crest's.

10.2.4 Spline Mode

To help with this feature, the Splines package is available as a separate purchase and its offline manual is available in the package.

10.2.5 Renderer Mode

This is the most advanced type of input and allows rendering any geometry/shader into the water system data. One could draw foam directly into the foam data, or inject a flow map baked from an offline simulation.

The geometry can come from a Mesh Renderer, or it can come from any Renderer component such as a Trail Renderer, Line Renderer or Particle System. This geometry will be rendered from a orthographic top down perspective to "print" the data onto the water. For simple cases, it is recommended to use an upwards facing quad for the best performance.

Note: It is recommended to use unlit shader templates or unlit *Shader Graph* for data if not using one of ours. If using Shader Graph with HDRP, then you must use the Built-In target in your Shader Graph.

The following shaders can be used with any input:

- **Override** sets the data to a value.
- **Scale** scales the water data between zero and one inclusive. It is multiplicative, which can be inverted, so zero becomes no data and one leaves the data unchanged.
- **Utility** a very configurable shader exposing blend modes and operations.

10.2.6 Paint Mode

To help with this feature, the Paint package is available as a separate purchase and its offline manual is available in the package.

CHAPTER

ELEVEN

WAVES

The Animated Waves simulation contains the animated surface shape. This typically contains the waves from shape components, but can also contain waves from the Dynamic Waves simulation. All waves will eventually be combined into this simulation so the water shader only needs to sample once to animate vertices.

11.1 Environmental Waves

The Shape FFT component is used to generate waves in Crest.

For advanced situations where a high level of control is required over the wave shape, the Shape Gerstner component can be used to add specific wave components. It can be especially useful for Trochoidal waves and shoreline waves. See the *Shoreline Waves* section for more information on the latter.

11.1.1 Wave Conditions

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 *New* or *Clone* 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 *Water Renderer* 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 Shape FFT 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.

The waves will be dampened/attenuated in shallow water if a Depth simulation is used (see *Depth Simulation*). The amount that waves are attenuated is configurable using the Attenuation In Shallows setting.

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

Wind

There are global wind controls on the Water Renderer which includes WindZone support. Global wind can be overriden on each Shape* component.

11.1.2 User Inputs

Waves can be applied everywhere in the world, placed along or orthogonal to a spline, or injected via a custom shader. The Shape FFT/Gerstner components supports the *Global Mode*, *Texture Mode*, *Spline Mode*, *Paint Mode* and *Renderer Mode*.

Global

This is the default mode and places waves globally. Global waves are additive and multiple global wave components can be used together. Furthermore, transition between wave conditions by interpolating between the Weight property of two Shape FFT/Gerstner components.

Renderer

The following shaders are available under *Crest/Inputs/Shape Waves* if using the *Renderer Mode*:

- **Add From Geometry** places waves from the spectrum into the world confined to the provided mesh.

11.2 Animated Waves

The environmental waves are termed "Animated Waves" in Crest. The Animated Waves simulation is also the displacement target where any data for vertex displacement is stored.

11.2.1 User Inputs

The Animated Waves Input component can write data to the simulation and supports the *Renderer Mode*.

If wanting to add waves, you should use the Shape FFT/Gerstner components as detailed above. This input is more for wave manipulation, manipulating displacement or custom waves via a custom shader.

For the Animated Waves Input's *Renderer Mode*, the following shaders are provided under the shader category *Crest/Inputs/Animated Waves*:

- **Push Water Under Convex Hull** pushes the water underneath the geometry. Can be used to define a volume of space which should stay 'dry'.
- **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/.

Under *Crest/Inputs/All* there is the following:

- **Scale By Factor** scales the waves by a factor where zero is no waves and one leaves waves unchanged. Useful for reducing waves.

11.2.2 Advanced Settings

The environmental waves are termed "Animated Waves" in the Crest system and can be configured under *Water Renderer → Simulations → Animated Waves*. Properties have detailed tooltips.

11.3 Dynamic Waves

11.3.1 Overview

Environmental/animated waves are 'static' in that they are not influenced by objects interacting with the water. 'Dynamic' waves are generated from a multi-resolution simulation that can take such interactions into account.

To turn on this feature, enable *Water Renderer → Simulations → Dynamic Waves → Enabled*. To configure the simulation, create or assign a *Dynamic Wave Lod Settings* asset to Settings.

The dynamic wave simulation is added on top of the animated 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 Shifting Origin to avoid travelling far distances from the world origin.

11.3.2 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 Sphere Water Interaction components to children on the object and set the *Radius* parameter to roughly match the shape.

Non-spherical objects can be approximated with multiple spheres, for an example see the *Spinner* object in the Boat sample 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, see the provided prefabs with the Boat sample.

11.3.3 Simulation Settings

The simulation can be configured with a *Dynamic Wave Lod Settings* asset.

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

The Debug GUI overlay reports the number of simulation steps taken each frame.

11.3.4 User Inputs

The Dynamic Waves Input component can write data to the simulation and supports the *Renderer Mode*.

The recommended approach to injecting forces into the dynamic wave simulation is to use the Sphere Water Interaction 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 using the *Renderer Mode*. The following input shader is provided under *Crest/Inputs/Dynamic Waves*:

- **Dampen Circle** dampens dynamic waves within a circle.

11.4 Shoreline Wave Simulation

The shallow water simulation is a new, next-generation feature for Crest and is not required for shoreline waves.

To help with this feature, the Shallow Water package is available as a separate purchase and its offline manual is available in the package.

CHAPTER

TWELVE

WATER EXCLUSION

Features detailed here can either exclude or include the surface and/or volume.

12.1 Clip Surface

https://www.youtube.com/watch?v=jXphUy__J0o

Fig. 12.1: Crest 4: Water Bodies and Surface Clipping (outdated but still useful)

This data drives clipping of the water surface (has no effect on the volume), as in carving out holes. It is similar to Unity's Terrain Holes feature. 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 *Surface Clipping* simulation on the *WaterRenderer* script, and ensure the *Alpha Clipping* is enabled on the water material.

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

12.1.1 User Inputs

The Clip Input component can write data to the simulation and supports the *Primitive Mode*, *Texture Mode*, *Spline Mode*, *Paint Mode* and *Renderer Mode*.

Primitive

Clip areas can be added using signed-distance primitives which produce accurate clipping and supports overlapping. The position, rotation and dimensions of the primitive is determined by the Transform.

Renderer

The following input materials are provided:

- **Clip Include Area**: Removes clipping data so the water surface renders.
- **Clip Remove Area**: Adds clipping data to remove the water surface.

12.2 Displacement

Displacing the water surface is another option for removing water from an area. Simply push the water below the surface using an Animated Waves Input or Water Level input.

When using the Animated Waves Input, it is possible to displace the water surface and have it affect underwater rendering:

- Enable the Displacement collision layer (see *Collision Layers*)
- Set Displacement Pass to *Lod Independent (Last)*.
- Set Collision Layer on Floating Objects to anything earlier to Everything

---

Tip: Displacement is not as precise as the clip simulation. It requires thicker walls to hide the edge of the effect.

It is also possible to nest buoyant objects, in addition to above:

- Set Collision Layer to not Everything on Floating Objects that you want to nested

12.3 Watertight Hull

The Watertight Hull component uses the clip simulation and/or displacement to remove water from a convex hull. Simply assign a convex hull mesh.

12.4 Mask Underwater

The *Portals & Volumes* feature can remove both the water surface and the underwater volume. Otherwise, enable/disable the Underwater Renderer where needed.

CHAPTER

THIRTEEN

WATER LEVEL

The Water Level (ie height) simulation can simulate water bodies of varied height and tides. The default water level is sea level, which is the height of the Water Renderer.

13.1 Tides

It is possible to move the entire water surface on the Y axis to simulate tides. The Water Level simulation can be used to localize the tide.

13.2 User Inputs

13.2.1 Geometry

Sets the water level using the provided geometry.

See *Geometry Mode* for more.

13.2.2 Spline

The spline input mode will use the spline mesh to set the water level.

See *Spline Mode* for more.

13.2.3 Paint

The paint input mode will enable paint tools to paint add/set the water level.

The water level can be raised or lowered with `LeftClick` and `Control-LeftClick` respectively. Use `Shift-LeftClick` to remove water level.

See *Paint Mode* for more.

13.2.4 Texture

Set water level using a height map.

See *Texture Mode* for more.

13.2.5 Renderer

Set the water level using a mesh using the *Water Level From Geometry* material.

See *Renderer Mode* for more.

13.3 Troubleshooting

There is staircasing and/or small bumps on the water surface

This is due to lack of precision. Set *Water Renderer* → *Simulations* → *Animated Waves* → *Texture Format Mode* to Precision, and set *Water Renderer* → *Simulations* → *Water Level* → *Texture Format Mode* to Automatic.

If the problem persists, and you are targetting a mobile platform, then enable *Project Settings* → *Crest* → *Full Precision Displacement On Half Precision Platforms*.

CHAPTER

FOURTEEN

WATER BODIES

14.1 Oceans

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

14.2 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 *User Inputs* section.
- If the lake altitude differs from the global sea level, use a *Water Level Input* to change the height of the water to match. It is recommended to cover a larger area than the lake itself, to give a protective margin against lower mesh densities from LODs in the distance.
- For advanced lake authoring, see the *Water Body Component*

14.2.1 Water Body Component

The Waterbody component can cull water chunks outside its bounds, clip water outside its bounds at the texel level, and override the material of chunks inside its bounds. The typical use case is for creating Closed Lakes, but it is not required.

Clipping

The Water Body 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.

Material Override

The Water Body can override the water material on the water chunks. This can be used to give closed lakes a distinct appearance.

Important: It is important to understand that since this feature cannot be applied partially to a water chunk, and a water chunk can overlap two water bodies, this feature does not work well with bordering water bodies - including bording an ocean. Typically it works best with only a single lake in the scene.

Tip: Crest has many simulations which can change the appearance of water at the texel level. We encourage everyone to consider those options instead.

Usage

The Water Body component, if present, marks areas of the scene where water should be present. It can be created by attaching this component to a Game Object 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.

1. Add WaterBody component to a GameObject
2. Position and expand the scale (XZ only) of the GameObject so the bounds covers the desired area (bounds visible with gizmos)
3. Configure clip and material overrides as desired
4. If clipping is enabled, set *Water Renderer → Simulations → Surface Clipping → Default Clipping State* to *Everything Clipped*

Tip: If you only want the material override feature, then disable *Water Renderer → Surface → Culling → Water Body Culling*.

14.3 Streams

There are several ways to accomplish streams. The most pertinent data is flow for the current and height for the stream gradient (optional).

14.3.1 Flow & Height Map

A river being a large stream, often without a visible gradient, can be implemented with just a *Flow Map*. The water level can also be adjusted to make a stream gradient by using a height map or a mesh with the Water Level Input.

14.3.2 Splines

Splines are a great option for creating streams. They can direct flow and create gradients.

To help with this feature, the Splines package is available as a separate purchase and its offline manual is available in the package.

14.3.3 Shallow Water Simulation

The Shallow Water Simulation can simulate streams and bake the final output to a texture.

To help with this feature, the Shallow Water package is available as a separate purchase and its offline manual is available in the package.

CHAPTER

FIFTEEN

SHORELINES AND SHALLOWS

Crest uses water depth information for various purposes, from attenuating large waves in shallow water to generating foam near shorelines. The way this information is typically generated is through the Depth Probe 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 probe. 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 that waves will be affected by the seabed when the water depth is less than half of their wavelength. 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 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 *Water Renderer* → *Simulations* → *Water Depth* → *Shallows Maximum Depth* which smooths the attenuation to a provided maximum depth where waves will be at full strength.

15.1 Depth Simulation

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 and to generate foam near shorelines. It is calculated by rendering the 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 water depth:

- Objects that have the *Depth Input* 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 probes as described above. The scene objects will be rendered into this probe once, and the results saved. Once the probe is populated, it is then copied into the Water Depth LOD Data. The probe has a gizmo that represents the extents of the probe (white outline) and the near plane of the camera that renders the depth (translucent rectangle). The probe 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.

15.1.1 Setup

https://www.youtube.com/watch?v=jcmqUlboTUk

Fig. 15.1: Crest 4: Depth Probe (formerly Ocean Depth Cache) usage and setup (outdated but still useful)

Tip: By default, Crest will include terrain height automatically. This can be disabled with *Water Renderer → Simulations → Water Depth → Include Terrain Height*. The aforementioned option is less accurate than a DepthProbe, and will not include details like rocks, but reduces friction for beginners.

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

The Main sample scene has an example of a probe set up around the island. The probe Game Object is called *Island-DepthCache* and has a Depth Probe 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 probe. If gizmos are visible and the probe is selected, the area is demarcated with a white rectangle.
- The Capture Range is the minimum and maximum height of any surfaces above the sea level that will render into the probe. If gizmos are visible and the probe 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 probe.
- Both the transform scale (white rectangle) and the Layers property determine what will be rendered into the probe.

By default the probe 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 /API/WaveHarmonic.Crest/DepthProbe/Populate method on the component from script.

Once populated the probe contents can be saved to disk by clicking the *Bake* button.

Overhangs

Since the Depth Probe captures from a top-down perspective, overhangs can cause problems by being captured instead of capturing what is below.

To solve this use the Capture Range to remove the overhang. The next potential issue is the hole left behind by cutting the top of the captured object. The Fill Holes feature can alleviate this by filling only the hole left behind. Set the *Fill Holes Capture Height* which relative to the Capture Range (ie 100 is 100 above the second)

15.1.2 Shoreline Foam

Once the Water Depth is running, shoreline foam can be configured. See *Simulation Settings* section for more information.

15.1.3 Troubleshooting

Crest runs validation on the depth probes - look for warnings/errors in the Inspector, and in the log at run-time, where many issues will be highlighted. To run validation, click the Validate Setup button at the bottom of the Water Renderer component inspector.

To inspect the contents of the probe, there is a preview pane available at the bottom.

15.2 Shoreline Waves

15.2.1 Attenuation

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

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

Fig. 15.2: Crest 4: Tweaking Shorelines (outdated but still useful)

15.2.2 Wave Map

Shape FFT/Gerstner components can accept a texture using the *Texture Mode*. The X/Y values map to the X/Z direction of the waves and their magnitude is the intensity of the waves.

15.2.3 Splines

Alternatively, using Shape Gerstner with a spline is an effective way to create shoreline waves. You will need to set Reverse Wave Weight to zero to avoid waves also going in the opposite direction and set Blend to Blend which effectively overwrites existing waves to prevent global waves from interferring.

To help with this feature, the Splines package is available as a separate purchase and its offline manual is available in the package.

15.2.4 Simulation

To help with this feature, the Shallow Water package is available as a separate purchase and its offline manual is available in the package.

CHAPTER

SIXTEEN

FLOW

16.1 Overview

Flow is the horizontal motion of the water volume. It does not affect wave directions, but transports the waves horizontally. This horizontal motion also affects physics. Furthermore, flow also affects foam, normals and caustics.

This can be used to simulate water currents and other waterflow.

16.2 User Inputs

Crest supports adding any flow velocities to the system. The Flow Input supports the following modes:

16.2.1 Spline

Flow will be generated along the spline direction. Reverse the spline if going in the wrong direction. The velocity of the flow can be adjusted per point with the *Spline Point Flow Data*.

See *Spline Mode* for more.

16.2.2 Paint

The paint input mode will enable paint tools to paint flow in the direction and magnitude of the brush stroke.

Use `LeftClick` and `Shift-LeftClick` to add and remove flow respectively.

See *Paint Mode* for more.

16.2.3 Texture

Writes a flow texture (ie flow map) into the system. It expects the XZ velocity to be packed into the RG components respectively. Furthermore if Normalized is enabled then it expects the values to be in the 0-1 range where 0.5 is zero velocity.

See *Texture Mode* for more.

16.2.4 Renderer

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

- **Add Flow Map** - The same as *Texture* except with more options.
- **Fixed Direction** - Adds flow in a fixed direction for the entire input.

See *Renderer Mode* for more.

CHAPTER

SEVENTEEN

FLOATING OBJECTS

Crest is capable of supporting various floating objects from barrels to watercraft.

17.1 Physics

Floating Object handles all physics interactions including buoyancy and drag. It has two models available: Align Normal and Probes.

Align Normal is a simple buoyancy model that attempts to match the object position and rotation with the surface height and normal. This can work well enough for small watercraft that do not need perfect floating behaviour, or floating objects such as buoys, barrels, etc. Furthermore, this model is less affected by physics, like mass, making authoring much easier like adding additional colliders.

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

17.1.1 Usage

Steps to get started:

1. Add a Floating Object to your Rigidbody.
2. Choose a model on the Floating Object.
    1. Add probes if using the Probes model.
3. Configure the Floating Object.
    - Force Strength is the most important value for keeping the object afloat. This value will differ greatly depending on the chosen model. Probes will take the rigid body mass into account. Align Normal will not and requires balancing Force Strength with Height Offset to get desired results.
    - Drag and Object Width/Length are important for keeping the object stable.

17.1.2 Troubleshooting

If a floating object is behaviing eratically, then there could be an internal collision. Review your collider hierarchy and see if it can be simplified.

17.2 Movement

Several scripts exist for adding movement to floating objects to make watercraft.

17.2.1 Usage

- Add a *Controller* to a *Floating Object*
- Add a control to the Controller

17.2.2 Controller

Controller is a general purpose watercraft controller script. It can control thrust, steer and dive mechanics by adding forces to the rigid body directly.

17.2.3 Controls

A control is an abstraction to allow input from various sources. It simply provides an XYZ value which maps to steer, dive and thrust respectively. We provide a few controls in the package.

Example

For a player control example, import the Boats sample. The Player Control works with both old and new Unity input systems. There is also a dive capable control in the Submarine sample. These are only examples because they are not the ideal usage of Unity's input systems.

The idea is that developers can easily make their own controls by extending the Control class. Input could come from the Unity input system or from some mechanic in the game world like a lever.

Fixed Control

Simply provdes a constant input which is configurable from the editor.

17.3 Interactions

The Sphere Water Interaction component is used to add interactions to floating objects like wakes. See *Adding Interaction Forces* section for more information on this component.

17.4 Watertightness

There are various methods to removing water from Crest detailed on the *Water Exclusion* page.

17.5 Troubleshooting

Wakes are causing Watercraft to jitter or behaving slightly erratically

When using the Sphere Water Interaction for wakes, it can create a feedback loop which causes watercraft to jitter. Excluding dynamic waves for this object is currently the only solution to solving these jitters. Please see *Collision Layers* for more information.

CHAPTER

EIGHTEEN

EVENTS

18.1 Query Events

The purpose of this component is to provide Unity Events for a scriptless approach to triggering changes using water information from queries.

It can perform the following:

- **Above/Below Water:** invoke a UnityEvent when the attached game object is above or below the water surface - once per state change. A common use case is to use it to trigger different audio when above or below the surface.
- **Distance From Surface**: Constantly sends the vertical distance from the water surface. Can be used to fade in and out audio.
- **Distance From Edge**: Constantly sends the horizontal distance from the water's edge. Can be used to fade in and out shoreline audio.

Part IV

Advanced

CHAPTER

NINETEEN

QUERIES

19.1 Usage

All providers which can be queried use a similar interface. They also have an accompanying helper class (known as sample helpers) to make basic queries simpler.

The 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 sample helpers are useful is that they are objects themselves and therefore can pass their own ID, hiding this complexity from the user.

Secondly, even if only a one-time query is needed, the query function should be called every frame until it indicates that the results were successfully retrieved. Querying a second time, in the same frame, using the same ID, will stomp over the last query points. 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 for the current frame, so satisfy this criteria.

Important: For each unique query ID:

- Queries should only be made once per frame.
- Number of query points posted should be kept consistent across frames.

The following code example uses the SampleCollisionHelper to query the water height at the current position. Since all the helpers have a similar interface, it can be applied to others with only some parameter changes.

```
using UnityEngine;
using WaveHarmonic.Crest;

public class ScriptExample : MonoBehaviour
{
   SampleCollisionHelper helper = new();

   // To avoid a one frame delay, call this before the WaterRenderer.LateUpdate call.
   void Update()
   {
      // Call only once per frame.
      if (helper.SampleHeight(transform.position, out float height))
```

(continues on next page)

(continued from previous page)

```
        {
            Debug.Log($"Height is {height}.");
        }
    }
}
```

Tip: For the most user-friendly, use-case driven approach, which requires not scripting, there is the *Query Events* component.

Alternatively, if you wish to query the provider directly, then this is how:

```
using UnityEngine;
using WaveHarmonic.Crest;

public class ScriptExample : MonoBehaviour
{
    readonly Vector3[] _query = new Vector3[1];
    readonly Vector3[] _height = new Vector3[1];

    // To avoid a one frame delay, call this before the WaterRenderer.LateUpdate call.
    void Update()
    {
        if (WaterRenderer.Instance == null) return;

        var provider = WaterRenderer.Instance.AnimatedWavesLod.Provider;

        _query[0] = transform.position;

        // Call only once per frame.
        var status = provider.Query(GetHashCode(), 0, _query, _height, null, null);

        if (provider.RetrieveSucceeded(status))
        {
            Debug.Log($"Height is {_height[0]}.");
        }
    }
}
```

The advantage with the direct method is if you to query multiple points at a time (we are only querying a single point in the example). This is more efficient using this approach.

19.2 Collision Shape

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 with *Water Renderer* → *Simulations* → *Animated Waves* → *Collision Source* dropdown.

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

To simplify the code required to get the water height, or other data via scripting, the /API/WaveHarmonic.Crest/SampleCollisionHelper is provided.

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: https://www.gdcvault.com/play/1023011/Fixed-Point-Iteration-A-Simple.

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

19.2.1 GPU Queries

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 water data and returns the desired results. The result of the query accurately tracks the height of the surface, including all wave components, depth caches and other Crest features. Set *Water Renderer* → *Simulations* → *Animated Waves* → *Collision Source* to GPU.

Collision Layers

GPU queries support collision layers. Collision layers allow excluding collision sources by querying the displacement data at certain points.

For example, using the After Animated Waves layer includes all waves, but excludes Dynamic Waves. This has often been requires as when using the Sphere Water Interaction to produce wakes, it can cause a feedback loop and jitters.

Collision layers can be enabled with *Water Renderer* → *Simulations* → *Animated Waves* → *Collision Layers*. When enabling a layer it will allow that layer to be included or excluded. On any object which queries collisions, like Floating Object, set the layer to what you want to include. The layers are in order (except Everything), and selecting a layer will include everything preceding it (eg After Dynamic Waves also includes Animated Waves).

Example

To have a Floating Object exclude dynamic waves, make sure Dynamic Waves layer is enabled on Collision Layers, and then set *Floating Object* → *Collision Layer* to After Animated Waves.

See the boats in the Boats sample.

There is also the Displacement layer which is a layer at the end. It can be used to render inputs which affect underwater and have nested buoyant objects (eg floating barrel in a ship).

19.2.2 CPU Queries

On platforms where a GPU is not present, or for authoritative servers, CPU queries are available.

To help with this feature, the Portals package is available as a separate purchase and its offline manual is available in the package.

19.3 Flow

Flow can also be queried, which is currently done on the GPU only. Query against the *Flow Provider* or use the /API/WaveHarmonic.Crest/SampleFlowHelper.

19.4 Water Depth

Currently, the water depth cannot be sampled, but the distance to water's edge can. See /API/WaveHarmonic.Crest/SampleDepthHelper/SampleDistanceToWaterEdge.

CHAPTER

TWENTY

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.

20.1 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 Custom Time Provider component to a GameObject and assign it to *Water Renderer* → *General* → *Time Provider*. Then time can be paused by setting the *_paused* variable on the Custom Time Provider component to *false*.

The Custom Time Provider 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 *WaterRenderer.Instance.PushTimeProvider()* to apply it to the system.

20.2 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 *Water Renderer* → *General* → *Time Provider*, 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. Crest expects that if the client time is ahead then the offset needs to be a negative value which may mean that you need to invert the sign of the offset first.

If the server needs the water shape to run physics but does not have a GPU then we have a CPU path, see *CPU Queries*. Different server conditions can be emulated in Editor using the Force Batch Mode and Force No GPU toggles on the Water Renderer.

Note that *Dynamic Waves* are not synchronised across the network and should not be relied upon in multiplayer projects.

20.3 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 Game Object and assign it to *Water Renderer* → *General* → *Time Provider*. This component will take the time from a Playable Director component which plays a cutscene Timeline. Alternatively, a Custom Time Provider component can be used to feed any time into the system, and this time value can be keyframed, giving complete control over timing.

CHAPTER

TWENTYONE

OPEN WORLDS

Crest follows the camera so it is inheritently suitable for open worlds, but there will be engine issues to account for.

21.1 Shifting Origin

To help with this feature, the Shifting Origin package is available as a separate purchase and its offline manual is available in the package.

CHAPTER

TWENTYTWO

SCRIPTING

22.1 Resources

Crest does not use the Resources folder. Instead a scriptable object which holds the references to all our compute and internal shaders is stored on the Water Renderer. This will tell Unity to include these files in a standalone build.

If instantiating the Water Renderer via scripting in builds, and there is no reference to a Water Renderer in the scene (including via prefab), then you must add a reference to the Resources asset in a scene to include all the necessary assets in the build. The Resources asset will not include shaders that are exposed in the shader selection for materials.

22.2 Scripting API

There is a separate API document provided in the same folder as this manual. Each package will have its own manual and API document.

22.2.1 Water Singleton

If you need a reference to the WaterRenderer, it can be obtained via the singleton WaterRenderer.Instance.

22.2.2 Assemblies

When using Assembly Definitions, there are two extra assemblies that need to be referenced: Shared and Scripting. The former is required to compile correctly, while the latter is optional.

The Scripting assembly provides extension methods necessary to adding input components:

```
// Add the foam input with the texture mode.
var input = gameObject.AddComponent<FoamLodInput>(LodInputMode.Texture);
// Retrieve the texture mode data and assign a texture.
var data = input.GetData<FoamTextureLodInputData>();
data.Texture = texture;
```

Part V

Guides

CHAPTER

TWENTYTHREE

PERFORMANCE GUIDE

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.

Tip: Tooltips will often mention if a feature has a significant performance cost.

23.1 Quality Parameters

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

- See parameters under *Water Renderer* → *Level of Detail* which directly control how much detail is in the water, and therefore the work required to update and render it. These are the primary quality settings from a performance point of view.
- Adjust resolution of simulations on a per-simulation basis.
- Persistent simulations can have their frequency reduced to improve performance.

23.2 Features

- The water shader has accrued a number of features, and has become a reasonably heavy shader. Where possible these are on toggles which can be disabled to will help the rendering cost.
- Disable any unused simulations under *Water Renderer* → *Simulations* will help.
- By default, there are several extra passes to accomodate the *Collision Layers* feature. If this feature is not needed, performance can be improved by disabling it.
- Rendering features under *Water Renderer* → *Rendering* like Motion Vectors, Write Color and Write Depth can be expensive (especially for BIRP).

23.3 Mobile Performance

Mobile is not the primary target for Crest, but the following are some hints on getting better performance:

- Crest can be draw call heavy which mobile platforms can be sensitive to. Together, reducing the *Water Renderer → Level of Detail → Levels* and increasing the *Water Renderer → Level of Detail → Scale* minimum can significantly reduce draw calls.
- For demanding platforms that use tile rendering (like the Meta Quest), consider setting the water to opaque, as it will net a significant performance gain. Transparency does not add a lot of value to open ocean scenes.

CHAPTER

TWENTYFOUR

RENDERING NOTES

24.1 Transparency

By default Crest is rendered in a typical 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.

24.1.1 Transparent Object In Front Of Water 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.

Preview

Please see the in-preview feature, injection-point , for a proper solution to this problem.

24.1.2 Transparent Object Behind The Water 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.

24.1.3 Transparent Object Underwater

Obsolete

This advice is only applicable to the *Legacy Underwater*. Please see *Integrate Water Volume (Shader Graph)*.

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

Rendering the transparent object after the underwater pass (occurs right after the transparent pass) with reduced transparency can often be good enough.

CHAPTER

TWENTYFIVE

SYSTEM NOTES

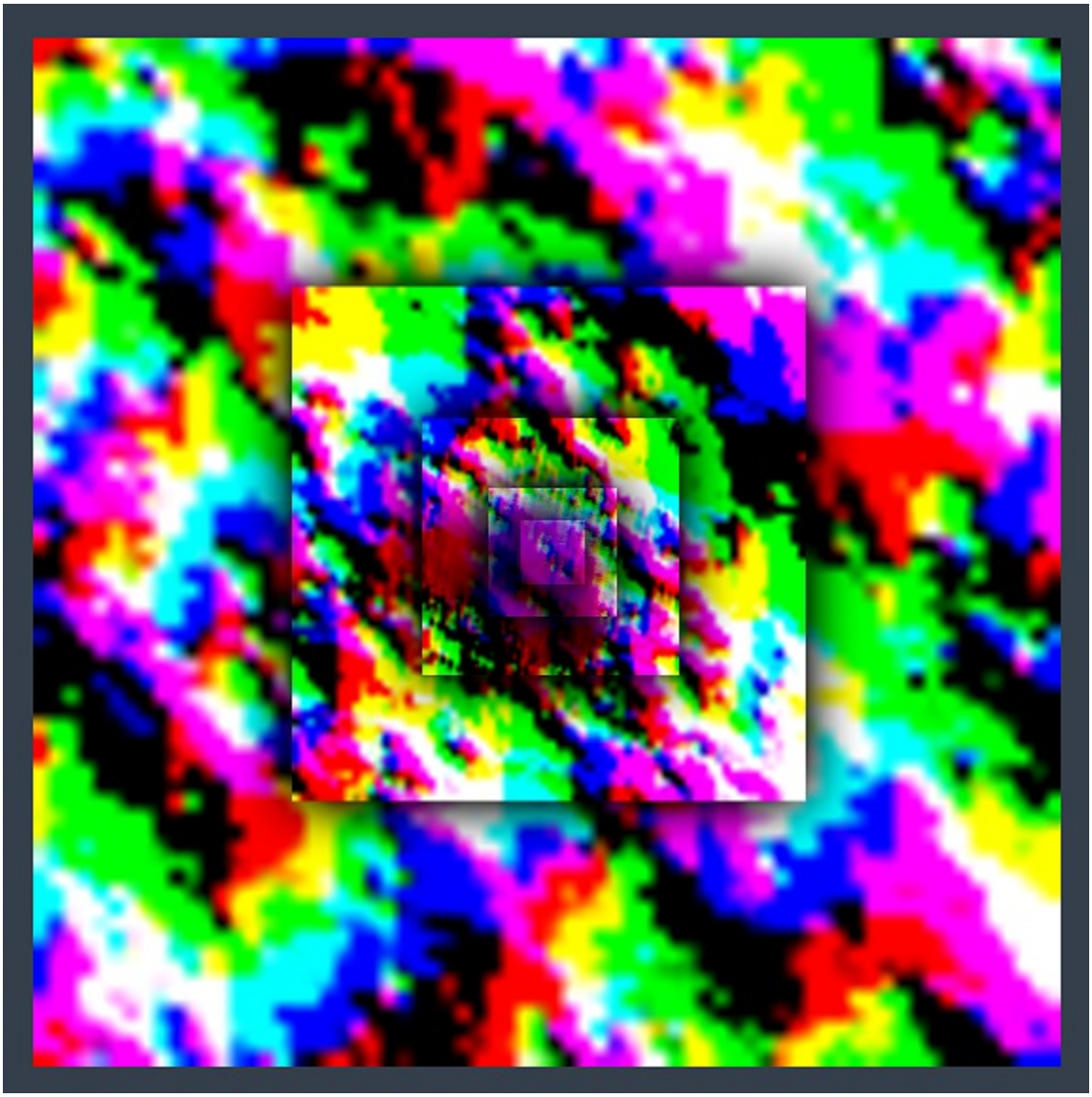
We have published details of the algorithms and approaches we use. See the following publications:

- Crest: *Novel Ocean Rendering Techniques in an Open Source Framework*, Advances in Real-Time Rendering in Games, ACM SIGGRAPH 2017 courses https://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

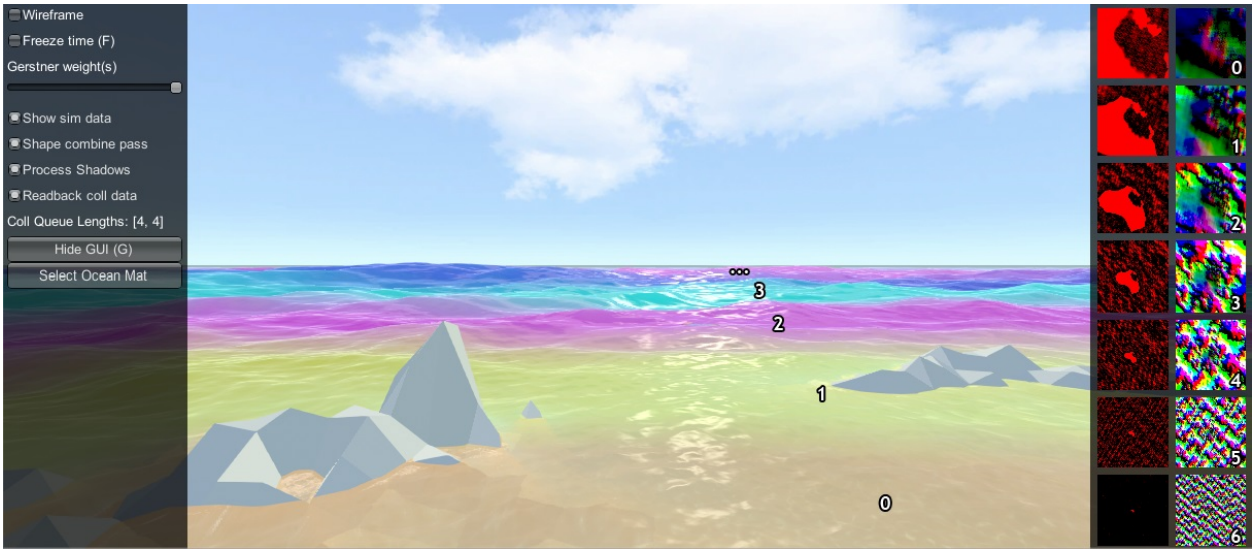
25.1 Core Data Structure

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 water 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:

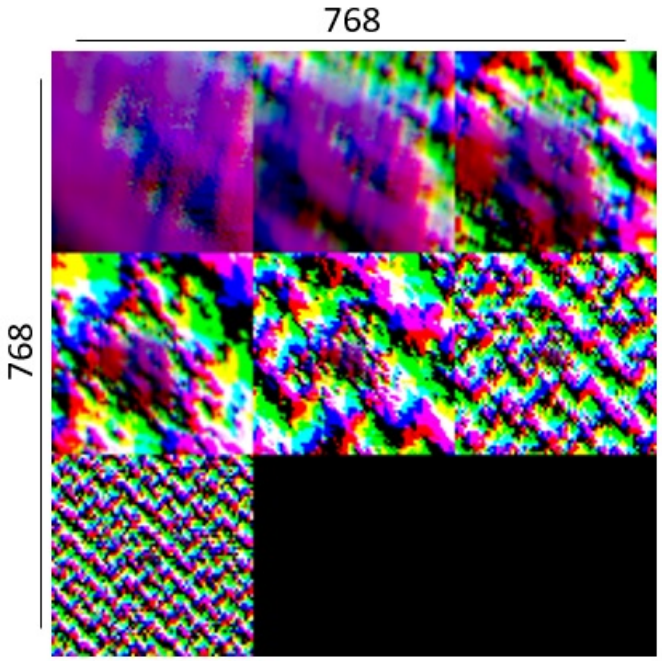


Each LOD is the same resolution (256x256 here), configured on the *Water Renderer* script. In this example the largest LOD covers a large area (4km squared), and the most detail LOD provides plenty of resolution close to the viewer. These textures are visualised in the Debug GUI on the right hand side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



In the above screenshot the foam data is also visualised (red textures), and the scale of each LOD is clearly visible by looking at the data contained within. In the rendering each LOD is given a false colour which shows how the LODs are arranged around the viewer and how they are scaled. Notice also the smooth blend between LODs - LOD data is always interpolated using this blend factor so that there are never pops are 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 *Water Renderer* has min and max scale settings to set limits on this dynamic range.

When rendering the water, the various LOD data is sampled for each vertex and the vertex is displaced. This means that the data is carried with the waves away from its rest position. For some data like foam this is fine and desirable.

For other data such as the depth to the water 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.

25.2 Implementation Notes

On startup, the *Water Renderer* script initialises the water system and asks the *WaterBuilder* script to build the water surface. As can be seen by inspecting the water 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 water system updates its state in *LateUpdate*, after game state update and animation, etc. *Water Renderer* updates before other scripts and first calculates a position and scale for the water. The water GameObject is placed at sea level under the viewer. A horizontal scale is computed for the water 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 water data are updated, such as animated waves, simulated foam, simulated waves, etc. The data can be visualised on screen if the *Debug GUI* 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 water data types, the water 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 *Debug GUI* to see the shape contents before this pass.

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

The water geometry is rendered by Unity as part of the graphics queue, and uses the *Crest/Water* 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 water 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 water 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.