Crest

Release 4.9

Wave Harmonic & Contributors

Mar 23, 2021

ABOUT

CHAPTER

ONE

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* 4.9 and targets URP (Universal Render Pipeline).

You can view this documentation online.

1.1 Social

- **Discord** : https://discord.gg/g7GpjDC
- **YouTube** https://www.youtube.com/channel/UC7_ZKKCXZmH64rRZqe-C0WA
- **Twitter** https://twitter.com/@crest_ocean

CHAPTER

TWO

RELEASE NOTES

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 *Water Bodies*.

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- Add validation to inform whether the depth cache is outdated.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 gray ocean by forcing depth and opaque texture when needed in the editor.
- Only feather foam at shoreline if transparency is enabled.

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 stablity 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 ocean depth cache triggered by other cameras or probes
- Fix underwater effect flickering when other cameras are in the scene

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

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

Performance

- Minor underwater performance improvement

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

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
- Fix caustics moving, rotating or warping with camera for URP 7.4+
- Fix caustics breaking for VR/XR SPI (Single Pass Instanced)
- Fix underwater material from breaking on project load or recompile

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 option to clip ocean surface under terrain
- Use local shader keywords

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 effects for URP 7.4+

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

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 underwater shader/material breaking on project load
- Fix shadow sampling running on cameras which isn't the main camera

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
- Make compatible with dynamic batching
- Add option to disable occlusion culling in planar reflections to fix flickering (disabled by default)

Fixed

- Fix *Segment registrar scratch exhausted* error that could appear in editor

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.

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 caustics stereo rendering for single-pass VR

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.
- Bump version to 4.1 to match versioning with *Crest HDRP*.

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.

3.8

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

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

Changed

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

3.5

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

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

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

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

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

CHAPTER

THREE

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:

https://www.youtube.com/watch?v=TpJf13d_-3E

Fig. 3.1: Getting Start with Crest for URP

3.1 Requirements

- Unity Version: 2019.4.9
- Shader compilation target 4.5 or above
- Crest does not support OpenGL or WebGL backends
- The minimum URP package version is 7.3

3.2 Importing *Crest* files into project

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

3.2.1 Pipeline Setup

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.

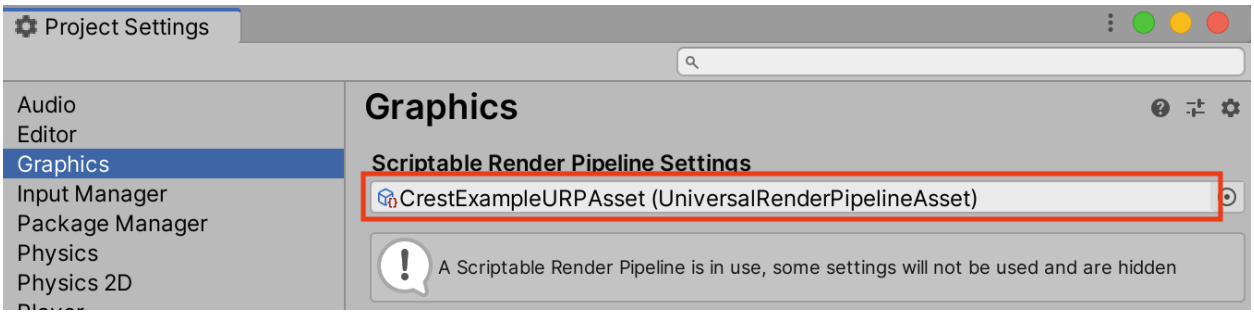
3.2.2 Importing Crest

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

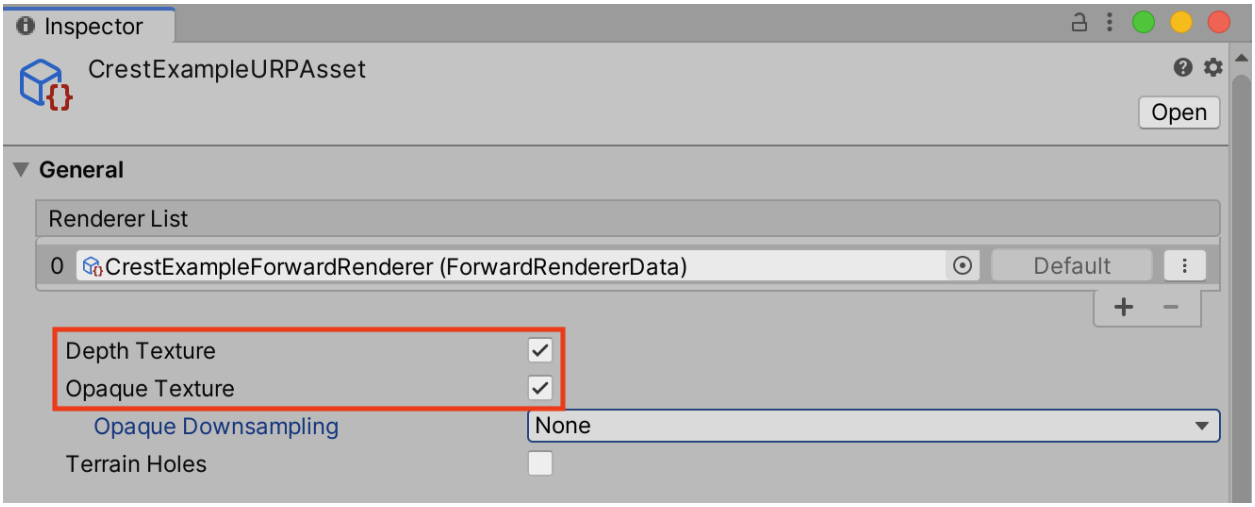
Note: The files under Crest-Examples are not required by our core functionality, but are provided for illustrative purposes. We recommend first time users import them as they may provide useful guidance.

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*:



Read Unity's documentation on the URP Asset for more information on these options.

Shadowing

To enable shadowing of the water surface which will darken the appearance in shadows, add the *Sample Shadows* Render Feature by following How to add a Renderer Feature to a Renderer.

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

3.3 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 Gerstner Batched* 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.

3.4 Frequent Setup Issues

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

3.4.1 Errors present, or visual issues

Try restarting Unity as a first step.

3.4.2 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:

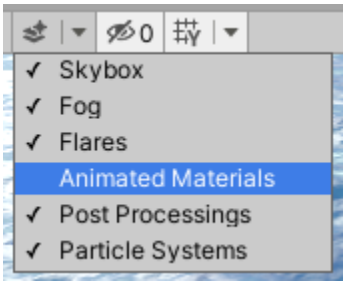
Installing URP into a project

3.4.3 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.

3.4.4 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.

CHAPTER

FOUR

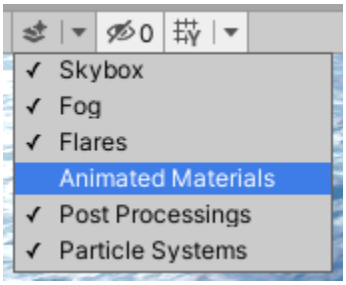
CONFIGURATION

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 *ShapeGerstnerBatched* script by providing an *Ocean Wave Spectrum* asset. This asset has an equalizer-style interface for tweaking different scales of waves, and also has some parametric wave spectra from the literature for comparison. 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.

4.1 Material Parameters

4.1.1 Normal Mapping

Enable Whether to add normal detail from a texture. Can be used to add visual detail to the water surface
Normals Normal map and caustics distortion texture (should be set to Normals type in the properties)
Normals Scale Scale of normal map texture
Normals Strength Strength of normal map influence

4.1.2 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.
Enable Shadowing Changes colour in shadow. Requires 'Create Shadow Data' enabled on OceanRenderer script.
Scatter Colour Shadow Base colour in shadow. Requires 'Create Shadow Data' enabled on OceanRenderer script.

4.1.3 Subsurface Scattering

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

4.1.4 Shallow Scattering

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.
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).

4.1.5 Reflection Environment

Specular Strength of specular lighting response.

Smoothness Smoothness of surface.

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

Smoothness Far Material smoothness at far distance from camera.

Smoothness Far Distance Definition of far distance.

Smoothness Falloff How smoothness varies between near and far distance.

Softness Acts as mip bias to smooth/blur reflection.

Fresnel Power Controls harshness of Fresnel behaviour.

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

Refractive Index of Water Index of refraction of water. Typically left at 1.333.

Planar Reflections Dynamically rendered 'reflection plane' style reflections. Requires OceanPlanarReflection script added to main camera.

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

4.1.6 Procedural Skybox

Enable Enable a simple procedural skybox. Not suitable for realistic reflections, but can be useful to give control over reflection colour - especially in stylized/non realistic applications.

Base Base sky colour.

Towards Sun Colour in sun direction.

Directionality Direction fall off.

Away From Sun Colour away from sun direction.

4.1.7 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.

Light Scale Scale intensity of lighting.

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

4.1.8 Foam 3D Lighting

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

Foam Normal Strength Strength of the generated normals.

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

Specular Boost Strength of specular response.

4.1.9 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.

4.1.10 Transparency

Enable Whether light can pass through the water surface.
Refraction Strength How strongly light is refracted when passing through water surface.
Depth Fog Density Scattering coefficient within water volume, per channel.

4.1.11 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 Strength How much the caustics texture is distorted.
Caustics Distortion Scale The scale of the distortion pattern used to distort the caustics.

4.1.12 Underwater

Enable Whether the underwater effect is being used. This enables code that shades the surface correctly from underneath.
Cull Mode Ordinarily set this to *Back* to cull back faces, but set to *Off* to make sure both sides of the surface draw if the underwater effect is being used.

4.1.13 Flow

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

4.2 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 settings control how much reflection contributes for different view angles.

The base reflection comes from a one of these sources:

- Unity's specular cubemap. This is the default and is the same as what is applied to glossy objects in the scene. It will support reflection probes, as long as the probe extents cover the ocean tiles, which enables real-time update of the reflection environment (see Unity documentation for more details).
- 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 wavey 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.

4.3 Lighting

TODO

Work in progress.

Crest URP currently does not support additional lights.

4.4 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.

4.5 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.

CHAPTER

FIVE

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. 5.1: Basics of Adding Ocean Inputs

5.1 Animated Waves

5.1.1 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.

5.1.2 User Inputs

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. 5.1

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 ether 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.
- **Add Water Height From 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 To 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/.

5.2 Dynamic Waves

5.2.1 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.

One use case for this is boat wakes. In the *boat.unity* scene, the geometry and shader on the *WaterObjectInteraction-Sphere0* GameObject will apply forces to the water. It has the *RegisterDynWavesInput* component attached to register it with the system.

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

5.2.2 Simulation Settings

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

- **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.

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

5.2.3 User Inputs

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

The following input shaders are 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.
- **Object Interaction** can be used in conjunction with the *ObjectWaterInteraction* script to simulate the interaction of an object with the water. Can be used for boat wakes. See the boat example scenes.
- **Sphere-Water Interaction** is a more specialized and accurate version of the *Object Interaction* input. It models the interaction between a sphere and takes into account how submerged the sphere is. Multiple spheres can be composed into compound shapes. See the *Spinner* object in the *boat.unity* example scene for an example.

5.3 Foam

5.3.1 Overview

Crest simulates foam getting generated by choppy water (*pinched*) wave crests) and in shallow water to approximate foam from splashes at shoreline. Each frame, 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.

5.3.2 User Inputs

User provided foam contributions can be added similar to the Animated Waves. In this case the *RegisterFoamInput* script should be applied to any inputs. There is no combine pass for foam so this does not have to be taken into consideration - one must simply render 0-1 values for foam as desired. See the *DepositFoamTex* object in the *whirlpool.unity* scene for an example. This is also demonstrated in Fig. 5.1.

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 canbe 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 at texture.
- **Override Foam** sets the foam to the provided value. Useful for removing foam from unwanted areas.

5.3.3 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.

Adding custom foam areas

Crest supports inputing 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/...*. The process for adding inputs is demonstrated in this Fig. 5.1.

Foam can be masked/removed by using the *FoamOverride* material.

5.4 Sea Floor Depth

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

The following will contribute to ocean depth:

- Objects that have the *RegisterSeaFloorDepthInput* component attached. These objects will render every frame. This is useful for any dynamically moving surfaces that need to generate shoreline foam, 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.

5.5 Clip Surface

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

Fig. 5.2: 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 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 values is greater than 0.5.

Overlapping meshes will not work correctly in all cases. There will be cases where one mesh will overwrite another resulting in ocean surface appearing where it should not. Overlapping boxes aligned on the axes will work well whilst spheres may have issues.

Clip areas can be added by adding geometry that covers the desired hole area to the scene and then assigning the *RegisterClipSurfaceInput* script. See the *FloatingOpenContainer* object in the *boat.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.

5.6 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.

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.

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

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

1. On the URP asset (either the asset provided with Crest *Assets/Crest/CrestExampleURPAsset*, or the one used in your project), ensure that shadow cascades are enabled. *Crest* requires cascades to be enabled to obtain shadow information.
2. Create a new renderer which will have the sample shadows feature enabled. Right click a folder under Assets and select *Create/Rendering/Universal Render Pipeline/Forward Renderer*. Then see How to add a Renderer Feature to a Renderer.

3. Add the new renderer to *General/Renderer List* of the URP asset created in the previous step.
4. Enable shadowing in Crest. Enable *Create Shadow Data* on the OceanRenderer script.
5. On the same script, assign a *Primary Light* for the shadows. This light needs to have shadows enabled, if not an error will be reported accordingly.
6. If desired the shadow sim can be configured by assigning a *Shadow Sim Settings* asset (*Create/Crest/Shadow Sim Settings*).
7. Enable *Shadowing* on the ocean material to compile in the necessary shader code

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

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.

5.7 Flow

5.7.1 Overview

Flow is the horizontal motion of the water volumes. It is used in the *whirlpool.unity* example scene to rotate the waves and foam around the vortex. It does not affect wave directions, but transports the waves horizontally. This horizontal motion also affects physics.

5.7.2 User Inputs

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. 5.1.

CHAPTER

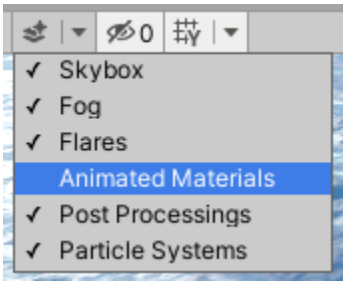
SIX

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.

6.1 Authoring

To add waves, add the *ShapeGerstnerBatched* component to a GameObject.

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 change the waves, right click in the Project view and select *Create/Crest/Ocean Wave Spectrum*, and assign the new asset to the *Spectrum* property of the *ShapeGerstnerBatched* script.

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'.

The *Wave Direction Variance* controls the spread of wave directions. This controls how aligned the waves are to the wind direction.

The *Chop* parameter 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 values we provide implementations of common wave spectra from the literature. Select one of the spectra by toggling the button, and then tweak the spectra inputs, and the spectrum values will be set according to the selected model. When done, toggle the button off to stop overriding the spectrum.

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

6.2 Local Waves

By default the Gerstner waves will apply everywhere throughout the world, so 'globally'. They can also be applied 'locally' - in a limited area of the world.

This is done by setting the *Mode* to *Geometry*. In this case the system will look for a *MeshFilter/MeshRenderer* on the same GameObject and it will generate waves over the area of the geometry. The geometry must be 'face up' - it must be visible from a top-down perspective in order to generate the waves. It must also have a material using the *Crest/Inputs/Animated Waves/Gerstner Batch Geometry* shader applied.

For a concrete example, see the *GerstnerPatch* object in *boat.unity*. It has a *MeshFilter* component with the *Quad* mesh applied, and is rotated so the quad is face up. It has a *MeshRenderer* component with a material assigned with a Gerstner material.

The material has the *Feather at UV Extents* option enabled, which will fade down the waves where the UVs go to 0 or 1 (at the edges of the quad). A more general solution is to scale the waves based on vertex colour so weights can be painted - this is provided through the *Weight from vertex colour (red channel)* option. This allows different wave conditions in different areas of the world with smooth blending.

6.3 ShapeGerstnerBatched

Deprecated since version 4.9: *ShapeGerstnerBatched* will be replaced by the much improved *ShapeGerstner*.

6.4 ShapeGerstner (preview)

A new Gerstner wave system has been added, intended to replace the current system. It can be tested by adding a *ShapeGerstner* component to a GameObject. The settings and behaviour are quite similar to the current system (described above). The new system has the following advantages:

- Much lower ocean update CPU cost per-frame (35% reduction in our tests for just one Gerstner component). Part of this efficiency comes from not recalculating the wave spectrum at run-time by default as toggled by the *Spectrum is static* option on the *ShapeGerstner* component. When this optimisation is enabled, *waves must be edited in edit mode (not in play mode)*.
- Lower GPU cost (0.12ms saved for wave generation in our tests)
- Wave foam generation works much better in background thanks to a new wave variance statistic
- Support for wave splines (see below)

After more testing we will switch over to this new system and deprecate the *ShapeGerstnerBatched* component.

6.5 Wave Splines (preview)

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

Fig. 6.1: Wave Splines

While it is possible to use the above steps to place localised waves in the world, we added a new system we call *Wave Splines* to make it easier and faster.

As part of this system, we added a generic *Spline* component which is in itself a useful spline tool which could be re-used for other purposes.

If the *Spline* component is attached to the same GameObject as a *ShapeGerstner* 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.

CHAPTER

SEVEN

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.

7.1 Setup

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

Fig. 7.1: 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.

7.2 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. 7.2: Tweaking Shorelines

7.3 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.

CHAPTER

EIGHT

WATER BODIES

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

Fig. 8.1: Water Bodies and Surface Clipping

Note: *Water Bodies* as a complete feature is a work-in-progress.

By default the system generates a water surface that expands our to the horizon in every direction. There are mechanisms to limit the area:

- The waves can be generated in a limited area - see the *Local Waves* 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.

8.1 Wizard (preview)

We recently added a 'wizard' to help create this setup. It can be used as follows:

- Open the wizard window by selecting *Window/Crest/Create Water Body*
- Click *Create Water Body*. A white plane should appear in the Scene View visualising the location and size
- Set the position using the translation gizmo in the Scene View, or using the *Center position* input
- Set the size using the *Size X* and *Size Z* inputs
- Each of the above components are available via the *Create ...* toggles
- Click *Create* and a water body should be created in the scene
- Click *Done* to close the wizard

CHAPTER

NINE

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/.

9.1 Compute Shape Queries

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

Using the GPU to perform the queries is efficient, but the results can take a couple of frames to return to the CPU. This has a few non-trivial impacts on how it must be used.

Firstly, queries need to be registered with an ID so that the results can be tracked and retrieved from the GPU 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.

9.2 Gerstner Waves CPU

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

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.

CHAPTER

TEN

UNDERWATER

Crest supports seamless transitions above/below water. It can also has 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.

Bug

Underwater effects do *not* support orthographic projection.

10.1 Underwater Curtain

In the *main.unity* scene, the *UnderWaterCurtainGeom* and *UnderWaterMeniscus* prefabs are parented to the camera which renders the underwater effects.

Checklist for using underwater:

- 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).

CHAPTER

ELEVEN

OTHER FEATURES

11.1 Decals

Use the *Render Alpha On Surface* component with a material using the *Crest/Ocean Surface Alpha* shader. For an example, please see the *CrestLogo* game object floating in the *Main* example scene.

11.2 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.

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

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

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

Bug

Surface details like foam and normals can pop on teleports.

11.3 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.

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

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

CHAPTER

TWELVE

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.

12.1 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
- The number of wave components will affect the update cost. This can be reduced by turning down sliders in the wave spectrum, and by reducing the *Components per Octave* setting on the *OceanGerstnerBatched* script.
- Our Gerstner 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.

CHAPTER

THIRTEEN

TECHNICAL DOCUMENTATION

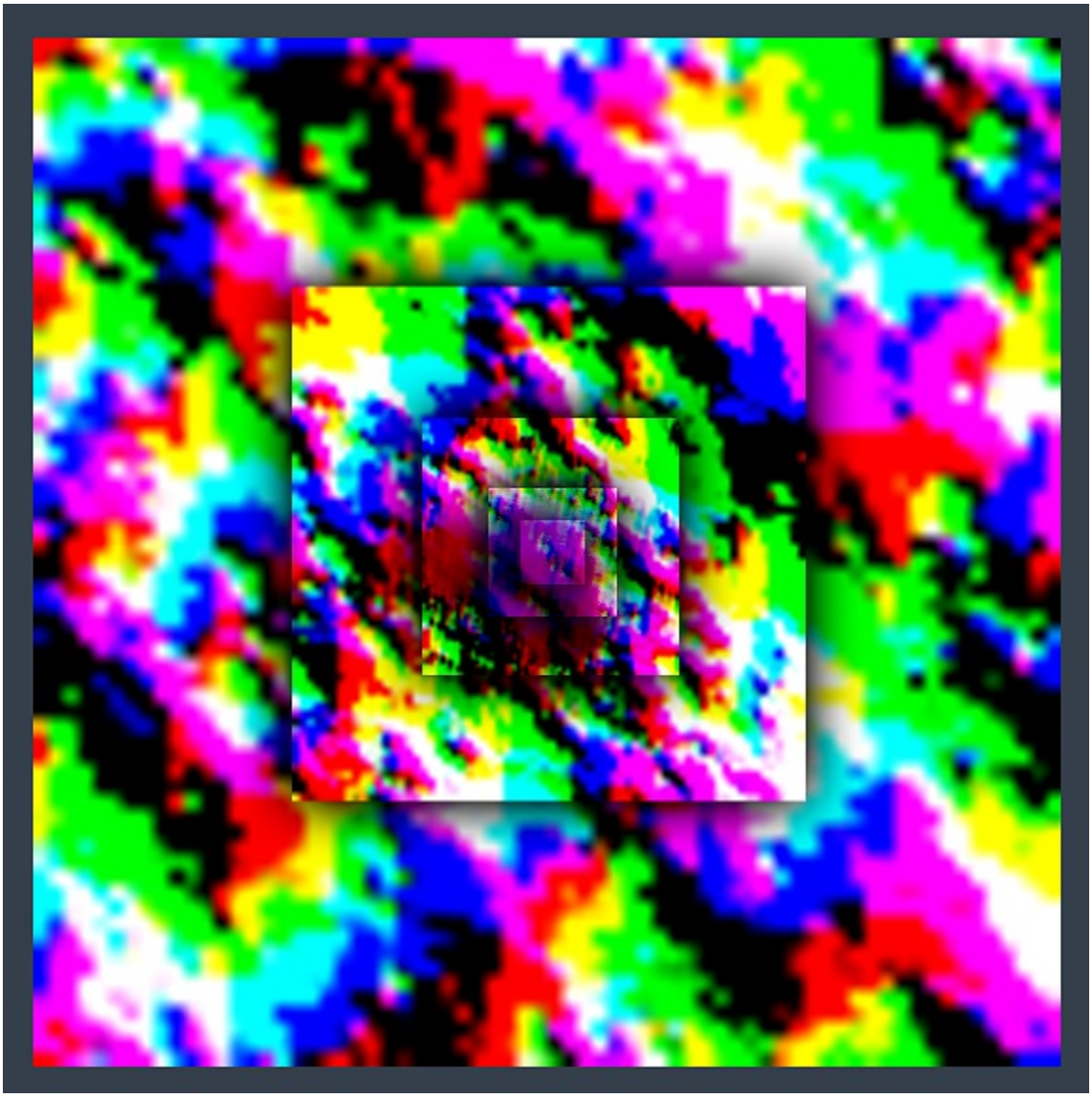
We have published details of the algorithms and approaches we use. See the following publications:

- 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

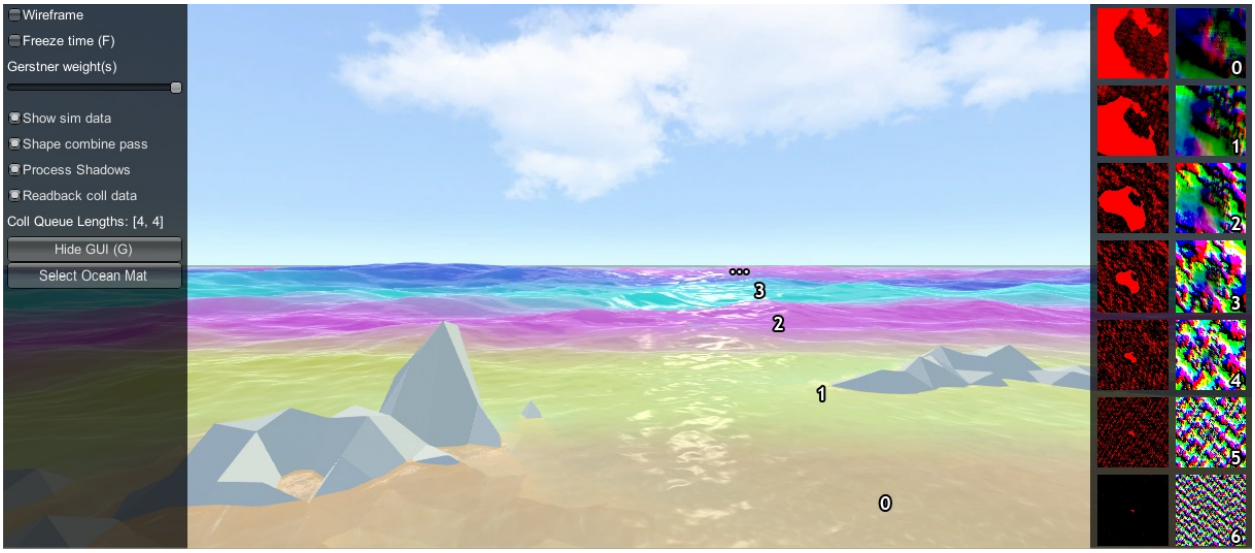
13.1 Core Data Structure

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:

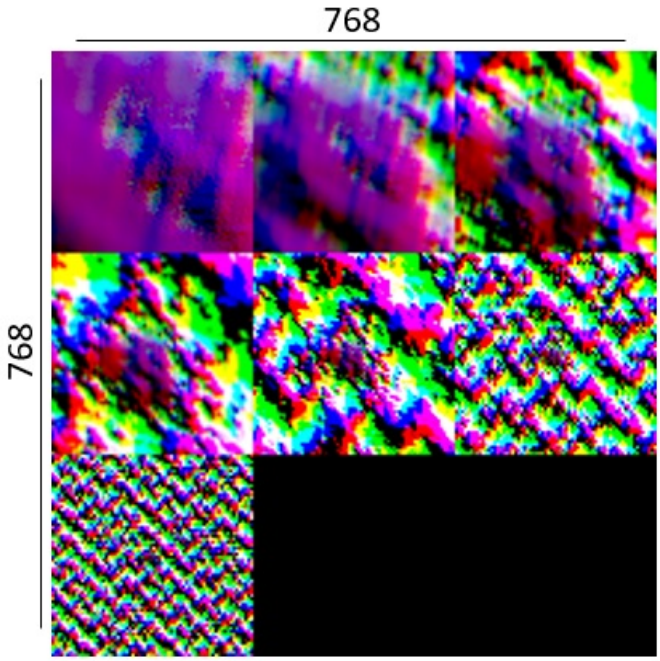


Each LOD is the same resolution (256x256 here), configured on the *OceanRenderer* script. In this example the largest LOD covers a large area (4km squared), and the most detail LOD provides plenty of resolution close to the viewer. These textures are visualised in the Debug GUI on the right hand side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

13.2 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 by rendering Gerstner wave components into the animated waves data. Each wave component is rendered into the shape LOD that is appropriate for the wavelength, to prevent over- or under- sampling and maximize efficiency. A final pass combines the shape results from the different Gerstner components together. Disable the *Shape combine pass* option on the *OceanDebugGUI* to see the shape contents before this pass.

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

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

13.3 Render Order

A typical render order for a frame is the following:

- Opaque geometry is rendered, writes to opaque depth buffer.
- Sky is rendered, probably at zfar with depth test enabled so it only renders outside the opaque surfaces.
- Frame colours and depth are copied out for use later in postprocessing.
- Ocean 'curtain' renders, draws underwater effect from bottom of screen up to water line.
  - Queue = Geometry+510 *BIRP*. Queue = Transparent-110 *URP*.
  - It is set to render before ocean in UnderwaterEffect.cs.
  - Sky is at zfar and will be fully fogged/obscured by the water volume.
- Ocean renders early in the transparent queue (queue = 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 the underwater curtain to make a correct final result.
  - It stomps over sky - sky is at zfar and will be fully fogged/obscured by the water volume.
- Particles and alpha render. If they have depth test enabled, they will clip against the surface.
- Postprocessing runs with the postprocessing depth and colours.

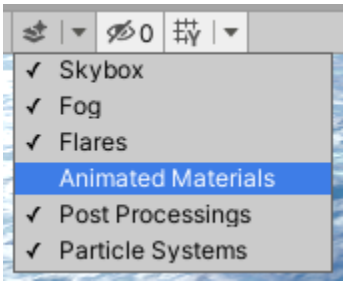
CHAPTER

FOURTEEN

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.

Crest also supports VR/XR Multi-Pass, Single 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 *Water Bodies* 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. Additionally, *Crest* does not currently support being run as a CPU-only headless instance. We hope to improve this in the future.

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. 5.1.

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?

The OceanSampleHeightEvents can be used for this purpose. It will invoke a UnityEvent when the attached game object is above or below the ocean surface once per state change.

Does Crest support orthographic projection?

Yes. Please see *Orthographic Projection*.

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.

Does Crest support third party sky assets?

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

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 shaders/materials underwater?

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 requires shader knowledge. One could create a new Unlit shader, create a material using this shader, and set the *Render Queue* property on the material to *Transparent*. This will draw the material after the underwater effect has been drawn, which will make it visible, but it will not have the underwater effect applied. The most obvious issue will be that the water fog is not applied. 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 *UnderwaterCurtain.shader* is a good reference for calculating the underwater effect. This will require various parameters on the shader like fog density and others.