Crest Ocean System 4.8 for HDRP

Wave Harmonic Ltd

January 31, 2021

Contents

1 Overview

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

This documentation covers the version of *Crest* targeted at the High Definition Render Pipeline (HDRP).

2 Initial set up

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

To augment / complement this written documentation we published a getting started video available here: `https://youtu.be/FE6l39Lt3js`.

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

2.1 Importing *Crest* files into project

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

1. Prepare a Unity project with the High Definition Render Pipeline (HDRP) installed.

    HDRP version 7.3.1 or later is required by *Crest*.

    Detailed instructions for installing HDRP are outside the scope of this documentation, see the Unity documentation for more information

    `https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.3/manual/index.html`.

    If you are starting from scratch we recommend selecting clicking *NEW* in the Unity Hub Projects view and selecting the *High Definition RP* template.
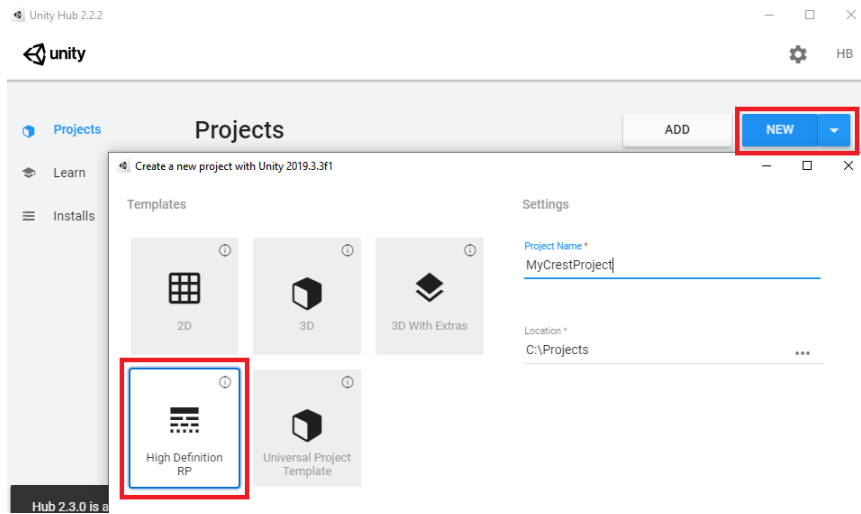
2. After first time installation / setup, restart the Unity Editor to ensure HDRP setup is complete.

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

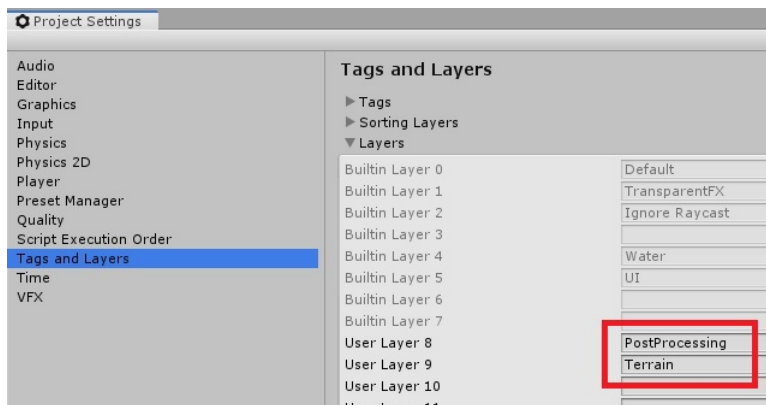   The files under *Crest-Examples* are not required by our core functionality but are provided for illustrative purposes. We recommend first time users import them as they may provide useful guidance.

4. To run the Crest example scenes, add a layer to the project named *Terrain* under *Edit/Project Settings.../Tags and Layers*.

5. HDRP defaults to using Temporal Anti-aliasing (TAA), which does not work well with the water material and makes it look blurry under motion. We recommend switching to a different anti-aliasing method such as SMAA using the *Anti-aliasing* option on the camera component.

6. If you imported the example content, open an example scene such as *Assets/Crest/Crest-Examples/Main/Scenes/main.unity* and press Play and

the ocean will get generated. Otherwise proceed to the next section to add the ocean to an existing scene.

2.2 Adding an ocean to a scene

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

1. Preparation: generate lighting from the *Lighting* window if necessary - the ocean lighting takes the ambient intensity from the baked spherical harmonics.

2. Create a new GameObject for the ocean, give it a descriptive name such as *Ocean.*

   (a) Assign the *OceanRenderer* component to it. On startup this component will generate the ocean geometry and do all required initialisation.

   (b) Assign the desired ocean material to the *OceanRenderer* script - this is a material using the *Crest/Ocean* shader.

   (c) Set the Y coordinate of the position to the desired sea level for your world.

3. Tag a primary camera as *MainCamera* using the Inspector if one is not tagged already, or set the *Viewpoint* property on the *OceanRenderer* script. If you need to switch between multiple cameras, update the *Viewpoint* field at run-time to ensure the ocean follows the correct view.

4. To add waves, create a new GameObject and add the *Shape Gerster Batched* component.

   (a) On startup this script creates a default ocean shape. To edit the shape, create an asset of type *Crest/Ocean Wave Spectrum* and provide it to this script.

   (b) Smooth blending of ocean shapes can be achieved by adding multiple *Shape Gerstner Batched* scripts and cross-fading them using the *Weight* parameter.

5. Any ocean seabed geometry needs mark up to register it with *Crest.* See section 6.

6. If the camera needs to go underwater, the underwater effect must be configured. See section 8 for instructions.

2.3 Frequent setup issues

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

Errors present, or visual issues   Try restarting Unity as a first step.

Compile errors in the log, not possible to enter play mode, visual issues in the scene Verify that HDRP is installed and enabled in the settings. See here for documentation: `https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.3/manual/Upgrading-To-HDRP.html`.

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 and delete the Library folder and restart which will trigger everything to reimport.

Ocean framerate low in edit mode 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:
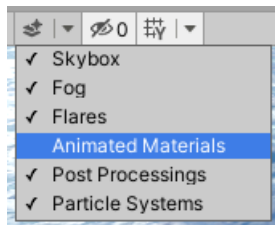
Ocean surface appears blurred under motion HDRP defaults to using Temporal Anti-aliasing (TAA), which does not work well with the water material and makes it look blurry under motion. We recommend switching to a different anti-aliasing method such as SMAA using the *Anti-aliasing* option on the camera component.

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

3 Configuration

3.1 Overview

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

- Edit mode behaviour: by default the ocean is visible in edit mode, and updates infrequently to save power. To enable real-time update enable *Animated Materials* in the Scene View toggles:
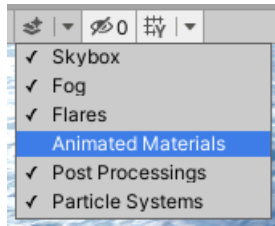
- Ocean surface appearance: The active ocean material is assigned to the *OceanRenderer* script.

  The material parameters are described in section 3.2. Turn off unnecessary features to maximize performance.

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

- Shallow water: Any ocean seabed geometry needs mark up to register it with *Crest*. See section 6.

- 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 8 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 4.2.2.

- A big strength of *Crest* is that you can add whatever contributions you like into the system. You could add your own shape or deposit foam onto the surface where desired. Inputs are generally tagged with the *Register* scripts and examples can be found in the example content scenes.

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

3.2 Material parameters

3.2.1 Normal Mapping

Normals Normal map texture (should be set to Normals type in the properties)
Normals Scale Scale of normal map texture
Normals Strength Strength of normal map influence

3.2.2 Scattering

Scatter Colour Base Base colour when looking straight down into water
Scatter Colour Shadow Base colour in shadow. Requires 'Create Shadow Data' enabled on OceanRenderer script.
Scatter Colour Shallow Colour in shallow water
Scatter Colour Depth Max Max depth that is considered 'shallow'
Scatter Colour Depth Falloff Fall off of shallow scattering

3.2.3 Subsurface Scattering

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 Tint Colour tint for primary light contribution
SSS Sun Falloff Fall-off for primary light scattering to affect directionality

3.2.4 Reflection Environment

Specular Strength of specular lighting response
Occlusion Strength of reflection
Smoothness Smoothness of surface
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

3.2.5 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 Albedo Intensity Scale intensity of diffuse lighting
Foam Emissive Intensity Scale intensity of emitted light
Foam Smoothness Smoothness of foam material
Foam Normal Strength Strength of the generated normals
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

3.2.6 Transparency

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

Depth Fog Density Scattering coefficient within water volume, per channel

3.2.7 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

3.2.8 Underwater

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

3.2.9 Flow

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

3.3 Reflections

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

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

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

HDRP comes with a *Planar Reflection Probe* feature which enables dynamic reflection of the environment at run-time, with a corresponding cost. Documentation is here but is a little sparse: `https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.3/manual/Planar-Reflection-Probe.html`. We could get it working by:

- Create new GameObject

- Set the height of the GameObject to the sea level

- Add the component from the Unity Editor menu using *Component/Rendering/Planar Reflection Probe*

- Set the extents of the probe to be large enough to cover everything that needs to be reflected

- Check the HDRP documentation linked above for details on individual parameters

3.4   Lighting

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

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

Lighting can also be overriden with the *Indirect Lighting Controller*. Please see the HDRP documentation on volume overrides for more information on setup and usage.

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

3.5   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* support orthographic projection.

3.6   Ocean construction parameters

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

- **Lod Data Resolution** - the resolution of the various ocean LOD data including displacement textures, foam data, dynamic wave sims, etc. Sets the 'detail' present in the ocean - larger values give more detail at increased run-time expense.

- **Geometry Down Sample Factor** - geometry density - a value of 2 will generate one vert per 2x2 LOD data texels. A value of 1 means a vert is generated for every LOD data texel. Larger values give lower fidelity surface shape with higher performance.

- **Lod Count** - the number of levels of detail / scales of ocean geometry to generate. The horizontal range of the ocean surface doubles for each added LOD, while GPU processing time increases linearly. It can be useful to select the ocean in the scene view while running in editor to inspect where LODs are present.

- **Max Scale** - the ocean is scaled horizontally with viewer height, to keep the meshing suitable for elevated viewpoints. This sets the maximum the ocean will be scaled if set to a positive value.

- **Min Scale** - this clamps the scale from below, to prevent the ocean scaling down to 0 when the camera approaches the sea level. Low values give lots of detail, but will limit the horizontal extents of the ocean detail.

4 Ocean 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`.

4.1 Animated waves

4.1.1 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 section 4.4). The amount that waves are attenuated is configurable using the *Attenuation In Shallows* setting.

4.1.2 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 *RegisterAnimWavesInput* script which will tag it for rendering into the shape. This is demonstrated in this video: `https://www.youtube.com/watch?v=sQIakAjSq4Y`.

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** 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/`.

4.2 Dynamic waves

4.2.1 Overview

This LOD data is a multi-resolution dynamic wave simulation, which gives dynamic interaction with the water. To turn on this feature, enable the *Create Dynamic Wave Sim* option on the *OceanRenderer* script.

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

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

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

4.2.2 Simulation setup

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

1. Set the *Gravity Multiplier* to the lowest value that is satisfactory. Higher values will make the simulated waves travel faster, but make the simulation more unstable and require more update steps / expense.

2. Increase *Damping* as high as possible. Higher values make the sim easier to solve, but makes the waves fade faster and limits their range.

3. Set the *Courant Number* to the highest value which still yields a stable sim. Higher values reduce cost but reduce stability. Put the camera low down near the water while testing as the most detailed waves are the most unstable.

4. Reduce *Max Sim Steps Per Frame* as much as possible to reduce the simulation cost. This may slow down waves in the lower LOD levels, which are the most detailed waves. Hopefully this slight slow down in just the smallest wavelengths is not noticeable/objectionable for the player. If waves are visible travelling too slow, increase it.

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

4.2.3 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

4.3 Foam

4.3.1 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.

4.3.2 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 this video: `https://www.youtube.com/watch?v=sQIakAjSq4Y`.

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

- **Add From Texture** adds foam values read from a user provided texture. Can be useful for placing 'blobs' of foam as desired, or can be moved around at runtime to paint foam into the sim.

- **Add From Vert Colours** can be applied to geometry and uses the red channel of vertex colours to add foam to the sim. Similar in purpose to *Add From Texture*, but can be authored in a modelling workflow instead of requiring at texture.

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

4.3.3 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.

4.3.4 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.

4.3.5 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.

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

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

4.4   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, etc.

- It is also possible to place world space depth caches. The scene objects will be rendered into this cache once, and the results saved. Once the cache is populated it is then copied into the Sea Floor Depth LOD Data. The cache has a gizmo that represents the extents of the cache (white outline) and the near plane of the camera that renders the depth (translucent rectangle). The cache should be placed at sea level and rotated/scaled to encapsulate the terrain.

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

4.5   Clip Surface

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.

4.6   Shadows

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

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

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

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

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

1. On the scriptable render pipeline asset (either the asset provided with Crest *Assets/Crest/CrestExampleHDRPAsset*, or the one used in your project), ensure that *Custom Pass* is enabled.

2. Shadow maps must be enabled in the frame settings for the camera.

3. Enable shadowing in Crest. Enable *Create Shadow Data* on the OceanRenderer script.

4. On the same script, assign a *Primary Light* for the shadows. This light needs to have shadows enabled, if not an error will be reported accordingly.

5. If desired the shadow sim can be configured by assigning a *Shadow Sim Settings* asset (*Create/Crest/Shadow Sim Settings*).

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.

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

4.7 Flow

4.7.1 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.

4.7.2 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 the following video: `https://www.youtube.com/watch?v=sQIakAjSq4Y`.

5 Wave conditions

It is useful to see the animated ocean surface while tweaking the wave conditions. By default, the ocean update is throttled by Unity to save power. To enable real-time update enable *Animated Materials* in the Scene View toggles:
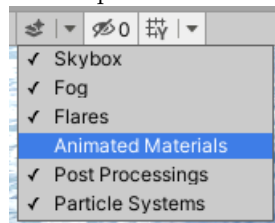
The following sections describe how to define the wave conditions.

5.1 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.

5.2   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.

5.3   ShapeGerstner - new system (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.

5.4   Wave Splines - new system (preview)

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.

6   Shallow water and shorelines

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

Crest requires water depth information to attenuate large waves in shallow water, to generate foam near shorelines, and to provide shallow water shading. The way this information is typically generated is through the *OceanDepthCache* component, which takes one or more layer names, and renders everything in those layers 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.

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 *Layer Names* field contains the layer that the island is assigned to - *Terrain*. Only objects in these layer(s) will render 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.

6.1 Troubleshooting

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

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

7 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.

7.1 Compute Shader Queries

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

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. **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**. 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.

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.

7.2 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.

8 Underwater

8.1 Overview

Crest supports seamless transitions above/below water. This is demonstrated in the *main.unity* scene in the example content. The ocean in this scene uses the material *Ocean-Underwater.mat* which enables rendering the underside of the surface.
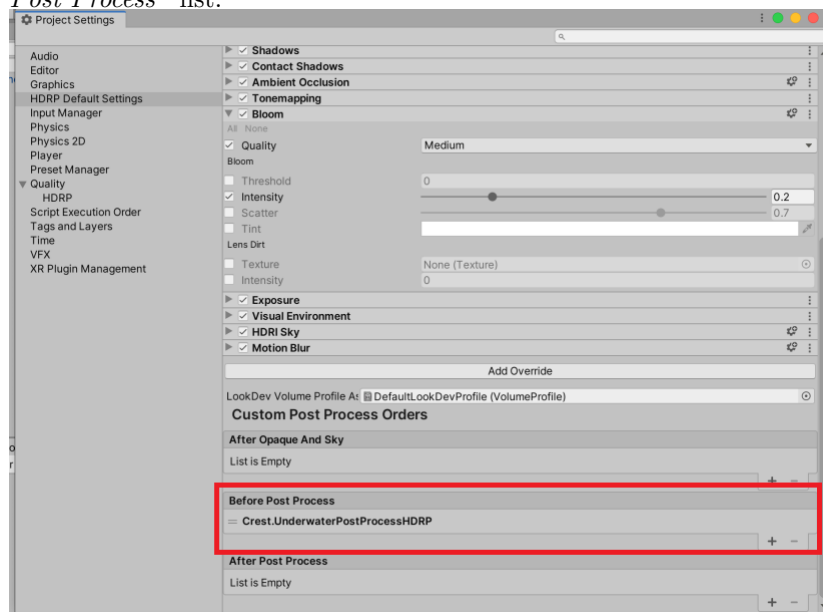
The underwater effect is implemented as a Custom HDRP Post Process effect.

8.2 Setup steps

Steps to set up underwater:

1. Ensure Crest is properly set up and working before proceeding

2. Add the underwater postprocess - go to *Edit/Project Settings/HDRP Default Settings* and add *Crest.UnderwaterPostProcessHDRP* to the *Before Post Process** list:

* For Unity 2020.2+/HDRP 10+, use *Before TAA*. This will fix the outline on objects undewater that use alpha clipping.

3. Configure the ocean material for underwater rendering - in the *Underwater* section of the material params, ensure *Cull Mode* is set to *Off* so that the underside of the ocean surface renders. See *Ocean-Underwater.mat* for an

23

example.
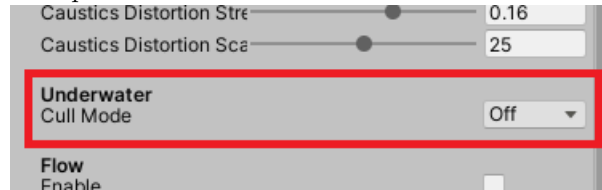
4. Use opaque or alpha test materials for underwater surfaces. Transparent materials may not render correctly underwater.

5. For performance reasons, the underwater effect is disabled if the viewpoint is not underwater. If there are multiple cameras, the *Viewpoint* property of the *OceanRenderer* component must be set to the current active camera.

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

9 Other features

9.1 Floating origin

Crest has support for 'floating origin' functionality, based on code from the Unity community wiki. See the original wiki page for an overview and original code: `http://wiki.unity3d.com/index.php/Floating_Origin`.

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

- Set the floating origin threshold to a power of 2 value such as 4096.

- Set the size/scale of any world space textures to be a smaller power of 2. This way the texture tiles an integral number of times across the threshold, and when the origin moves no change in appearance is noticeable. This includes the following textures:

  - Normals - set the Normal Mapping Scale on the ocean material
  - Foam texture - set the Foam Scale on the ocean material
  - Caustics - also should be a power of 2 scale, if caustics are visible when origin shifts happen

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

9.2 Buoyancy / floating physics

Buoyancy physics for boats is not a core focus of *Crest*. For a professional physics solution we recommend the Dynamic Water Physics 2 (DWP2) asset

which is compatible with *Crest*. With that said, we do provide the following 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.

9.2.1 Adding boats

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

1. Pick an existing boat to replace. Only use *BoatAlignNormal* if good floating behaviour is not important, as mentioned above. The best choice is usually boat probes.

2. Duplicate the prefab of the one you want to replace, such as *crest/Assets/Crest/Crest-Examples/BoatDev/Data/BoatProbes.prefab*

3. Remove the render meshes from the prefab, and add the render mesh for your boat. We recommend lining up the meshes roughly.

4. Switch out the collision shape as desired. Some people report issues if the are multiple overlapping physics collision primitives (or multiple rigid-bodies 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.

10 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`

10.1 Core data structure

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

Let's study one of the LOD data types in more detail. The surface shape is generated by the Animated Waves LOD Data, which maintains a set of *displacement textures* which describe the surface shape. A top down view of these textures laid out in the world looks as follows:
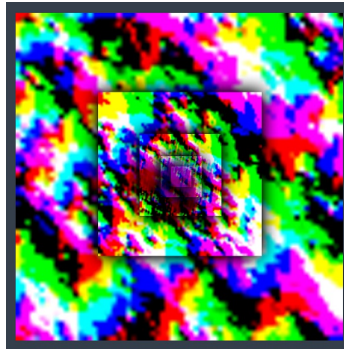
Each LOD is the same resolution (256x256 here), configured on the *Ocean-Renderer* script. In this example the largest LOD covers a large area (4km squared), and the most detail LOD provides plenty of resolution close to the viewer. These textures are visualised in the Debug GUI on the right hand side of the screen:
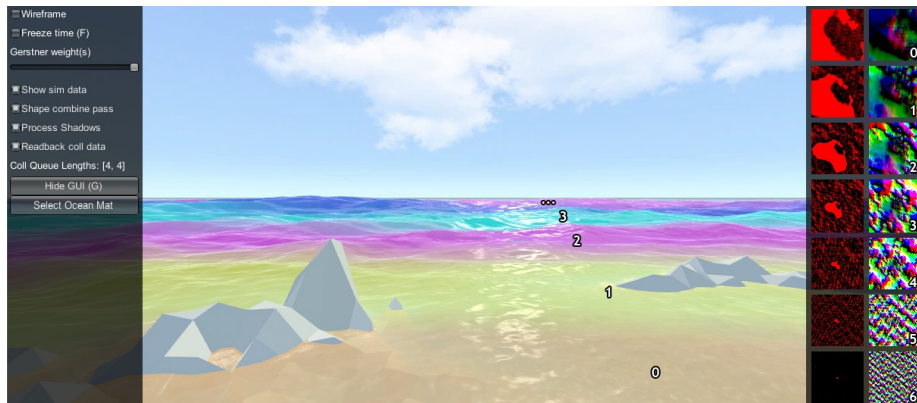
In the above screenshot the foam data is also visualised (red textures), and the scale of each LOD is clearly visible by looking at the data contained within. In the rendering each LOD is given a false colour which shows how the LODs are arranged around the viewer and how they are scaled. Notice also the smooth blend between LODs - LOD data is always interpolated using this blend factor so that there are never pops 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:
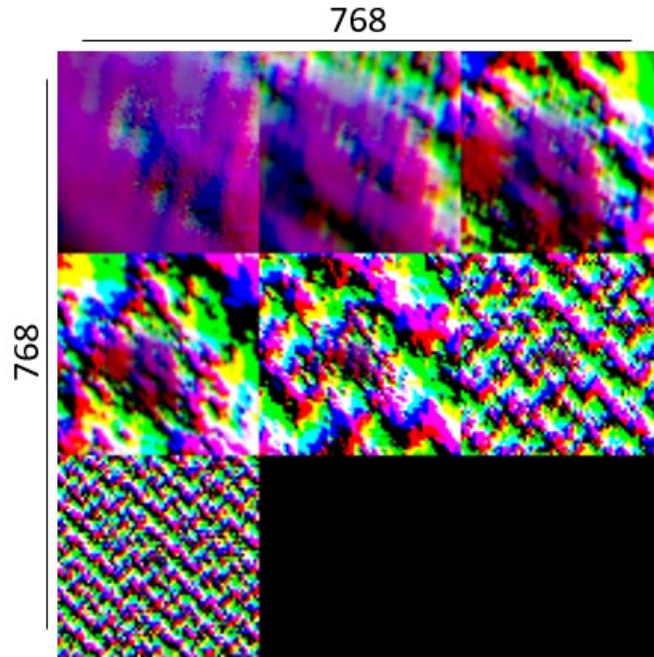
A final feature of the LOD system is that the LODs change scale with the viewpoint. From an elevated perspective, horizontal range is more important than fine wave details, and the opposite is true when near the surface. The *OceanRenderer* has min and max scale settings to set limits on this dynamic 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 issue 96.

10.2 Implementation notes

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

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

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

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

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

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

11 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.

11.1 Quality parameters

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

- See section 3.5 for parameters that directly control how much detail is in the ocean, and therefore the work required to update and render it. These are the primary quality settings from a performance point of view.

- The ocean shader has accrued a number of features and has become a reasonably heavy shader. Where possible these are on toggles and can be disabled, which will help the rendering cost (see section 3.2).

- 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.

11.2   Potential optimisations

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

- The ocean update runs as a command buffer, portions of which could potentially be ran asynchronously which may improve GPU utilisation. For example dynamic waves and Gerstner waves could be computed in parallel before the combine pass merges them together. We attempted this but found platform support is currently too limited to allow us to implement this in core *Crest*.

- Create multiple ocean materials with different sets of features and switch between them at run-time. An obvious use case would be one for shallow water and one for deep water and perhaps switch between them on a per-tile basis.

- Limit resolution range of LOD data. There is currently a min/max grid size option on the dynamic wave sim to limit what resolutions it runs at, this could be rolled out to other sim types.

- Set LOD data resolution per LOD type - e.g. the ocean depth and foam sim could potentially be stored at half resolution or lower.

- Pre-rendered wave displacements - sample waves from texture instead of computing them on the fly. We did some initial experimentation in the branch *feature/baked-waves* and found it challenging to get good shape without interpolation artifacts. Given that the baking step is also inconvenient, there are no plans to explore this further.

12   Q&A

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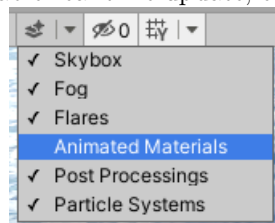
Is *Crest* well suited for medium-to-low powered mobile devices?   *Crest* is built to be performant by design and has numerous quality/performance levers. However it is also built to be very flexible and powerful and as such can not compete with a minimal, mobile-centric ocean renderer such as the one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

Which platforms does *Crest* support?   Testing occurs primarily on Windows. We have users targeting Windows, Mac, Linux, PS4, XboxOne and Switch. Performance is a challenge on Switch - see the previous question.

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

Can *Crest* work with multiplayer?   Yes the animated waves are deterministic and easily synchronized. See discussion in `https://github.com/wave-harmonic/crest/issues/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 this video: `https://www.youtube.com/watch?v=sQIakAjSq4Y`.

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.

Does Crest support orthographic projection?   It does. Please see section ??.