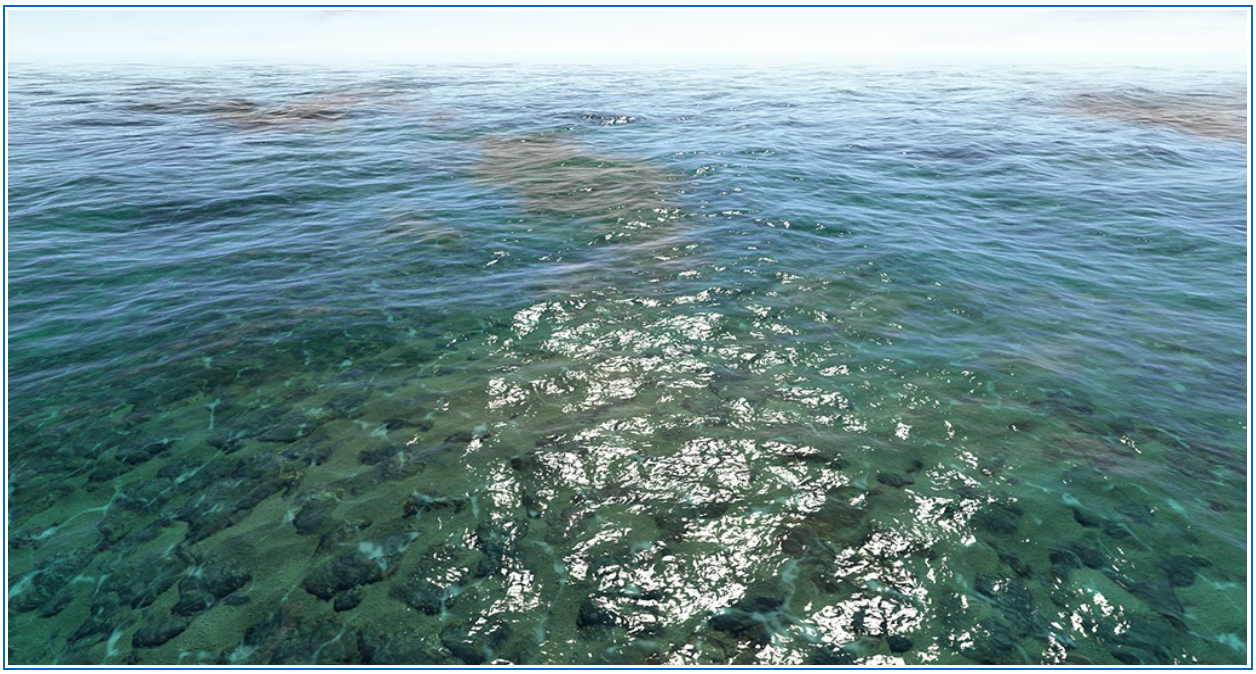
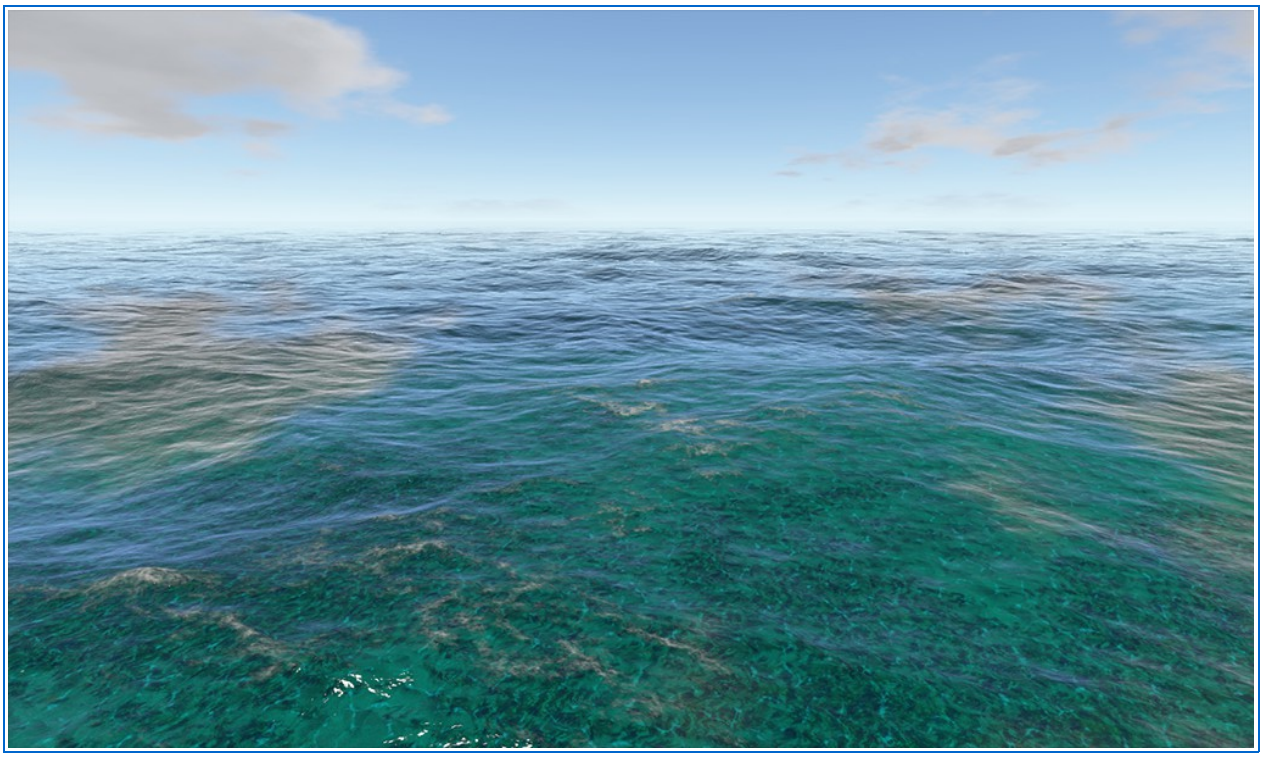
Ceto

Version 1.1.3

27 JUL 2017





Contents

- How to cache spectrum conditions
- How to use asynchronous wave query tasks
- How to set Ceto up for SteamVR

6. Common Problems
    - Where are my reflections?
    - Why does this object not have the underwater effect applied to it?
    - Why is this wave overlay not working?
    - Why cant I get global fog to be applied to the ocean?
    - Why cant I get shadows to show on the ocean?
    - Why cant I get point and spot lights to show on the ocean?
    - Why is there a strange artefact on the ocean behind a Speed Tree?
    - Why is there a strange artefact on the tips of the waves?
    - Why is my object not floating on the waves?
    - Why is a object showing in the reflections when its not seen by the main camera?
    - Why are my particles not showing correctly underwater?
    - Why are the caustics not showing up in my scene?
    - I am getting a error after I upgraded to a new Ceto version. How do I fix it?
    - Why is the ocean not rendered correctly in reflection probes?
    - Why is the ocean not showing up in Unity's DepthNormal texture?
    - Unity 5.4 upgrade notes
    - Why Does the ocean drawn over the tree Billboards on the terrain?
    - Why does changing the transparent ocean render queue to “Transparent-500” break the ocean?
    - Unity 5.5 upgrade notes.
    - How to add support for single pass rendering in VR to Ceto.

1. Quick Guide

Follow these steps to quickly set Ceto up in your scene. You must run the scene for the ocean to show up. Ceto does not run in the editor.

- First add Ceto to your scene by dragging the Ceto/Prefabs/Ocean prefab into the scene. You will see there is a transparent and opaque prefab. See the usage decisions section to see the pros and cons of each. The ocean should look the sample for both prefabs. Its just the render queue that's different. Is you are unsure about which to choose use the transparent prefab.
- On the ocean component set the level to the value your sea level should be.
- On the ocean component drag a directional light onto the sun slot. This step is optional but if used the suns direction and colour will be used for certain effects like subsurface scatter and the underwater post effect.
- On the Wave Spectrum component set the wind speed to a value based on how large you want the waves to be. A higher wind speed creates larger waves. A value of 2-4 is good for calm conditions and 10-12 for rough conditions. You can also use the grids scale to adjust the overall scale of the waves.
- On the Planar Reflection component click on the reflection mask and select which layers should be in the reflections. Keep the number of layers to a minimum. You can adjust if the reflections are blurred and by how much by using the blur mode and blur iterations.
- On the Underwater component click on the ocean depths mask and select the layers that will be underwater. You may need to organise your layers so you can easily filter out anything that will never be underwater. Whatever layer Speed Trees are on cannot be selected. They will not render correctly if they are. You may need to create a layer for your trees and unselect them on this mask.

  If you are using the transparent prefab with the depth mode set to USE_DEPTH_BUFFER then this step is not needed. Its only applies if the USE_OCEAN_DEPTH_PASS mode is selected. See the usage decisions section to see the pros and cons of each option.
- If your camera will only ever be looking down on the ocean and will not go under it then set the underwater mode to ABOVE_ONLY.
- If the camera will go under the water and you want the ‘under water fog’ effect you have to set the underwater mode to ABOVE_AND_BELOW and you must also place the UnderWaterPostEffect script on the camera.

You may only have one ocean object in the scene at a time. If more than one prefab is added only the first one will be used. Any other prefabs will report an error and shut themselves down.

The Ocean is on the water layer by default. If you do not want the camera to render the ocean then deselect the water layer in the camera culling mask. You can change what layer the ocean is on in the Ocean script.

2. Usage Decisions

Ceto provides you with many options so you can get the effect you want for the best performance possible. Some of these options however have draw backs. Choosing the correct one for your needs is important at may mean giving up on other features.

Transparent or opaque prefab?

Ceto provides you with two prefabs. One will use materials that render in the transparent queue and the other in the opaque queue. Both oceans should look the same its just the position in the render queue that is different. The transparent material will render in the Transparent-1 queue and the opaque in the AlphaTest+50 queue.

Transparent queue pros

- The shader will support blending which makes features like the edge fade much easier.
- The depth info used to apply the underwater effect can be calculated directly from depth buffer.
- Certain image effects like SSAO will not be applied to the ocean which dont tend to look good on the ocean.

Transparent queue cons

- The ocean can not have shadows on it as shadows on transparent objects are not supported in Unity.
- Certain image effects like global fog will not be applied to the ocean. Most sky assets also apply their fog effect here meaning it will also not be applied to the ocean. This is because they are applied after the opaque queue but before the transparent one. Unity's normal fog will still effect the ocean.

It should be mentioned that you can still apply the fog to the ocean by shifting when it gets applied. This requires removing the 'ImageEffectOpaque ' tag above the skys OnRenderImage function in the post effect script. This may have side effects and the fog maybe incorrectly applied to other transparent objects.

Opaque queue pros

- The ocean will have all image effects applied to it like global fog .
- The ocean can receive shadows.

Opaque queue cons

- When under the water looking up the sky will not be in the refractions as the sky box gets filled after the opaque has run.
- The depth information used to apply the underwater effect has to be created using a replacement shader. Replacement shaders can the quite expensive in complex scenes.
- Applying the edge fade is much complicated as blending is not supported for opaque objects. While Ceto will apply the edge fade for the opaque ocean artefacts (a dark patch when the edge fade is) may occur in some situations. You may need to disable the edge fade feature in the shader if this occurs. This artefact seems to only occur on the terrain when shadows are enabled.

CPU or GPU Fourier transform?

To generate the wave data Cetos uses Fast Fourier Transform. FFT can be quite demanding to calculate. To handle larger Fourier sizes Ceto can run the FFT on the GPU where it will run much faster. The issue is that the data then must be read back from the GPU into the main system RAM to query the wave height of the ocean. The read back is very costly and wont be practical for most applications. This means you will need to make a trade off between performance and practicality. A CPU or GPU option can be selected on the wave spectrum scripts Fourier size setting.

Reading back the data is only supported on DX11 systems. Reading back the data can be disabled on the wave spectrum script and is disabled by default.

CPU pros

- The data is easily accessible for the height queries.

CPU cons

- Ceto does the FFT on a separate thread (Its actually split up between 3 separate threads) but even then its still much slower than the GPU so there is a limit on the Fourier size you can use. Up to 128 is provided but only up to 64 will be practical on most systems.

GPU pros

- Its very fast and Fourier sizes up to 512 are supported.

GPU cons

- Reading back the data is expensive and only practical on DX11 systems. On some older DX11 systems the read back can take up to 20ms.

As a general guide...

If your making a game with boats where buoyancy is important use the CPU.

If you making a game where the ocean is just a backdrop then use the GPU.

Ceto will only run the FFT for the height data on the CPU. There is also a FFT for the normals and foam. These will always run on the GPU.

Depth buffer or ocean depth pass?

If an object is under the ocean then it must have the underwater effect applied to it. To do this Ceto needs some depth information about the object. This can be gained using two different methods. On the Underwater component on the ocean you will see a depth mode setting. The options are USE_DEPTH_BUFFER or USE_OCEAN_DEPTH_PASS.

Depth buffer mode pros

- This will get the depth information directly from the depth buffer so is extremely fast.
- It does not require objects in you scene to be treated any differently.

Depth buffer mode cons

- The depth buffer can only be read after the opaque render queue has finished so this option can not be used for the opaque ocean prefab.
- The depth buffer has limited precision so the underwater effect may have artefacts when viewed from a large distance. This should only be a issue if your game is something like a flight simulation where the cameras far plane is set to a high value. You maybe able to prevent these artefacts by increasing the near plane value.

Ocean depth pass mode pros

- This option is independent from the render queues so can be used where the depth buffer is not normally accessible.
- In theory any arbitrary information can be written into the buffer which can be more flexible.

Ocean depth pass cons

- The ocean pass is done using a replacement shader. Replacement shaders are expensive as it requires the geometry to be drawn a second time. You can use the depth mask to select only the layers you need rendered. This will help reduce your draw calls. This may require you to organise your layers so you can easily unselect anything that will never go under the water.
- Unity has handled the Speed Tree shaders differently than normal for some reason. They do not have a render type so can not added to the replacement shader. This means you will get a strange pattern on the ocean behind the trees. You will need to deselect the layer speed trees are on from the depth mask.
- Any object that has a custom vertex shader will have to use a custom render type and then have a pass to render its ocean depth to the replacement shader. This includes any vertex displaced or tessellated objects.
- You can not have the TransparentFX layer selected in the ocean depths mask. They will not render correctly.

If you try and use the opaque prefab with the depth mode set to USE_DEPTH_BUFFER you will get a warning that this method is not supported for the opaque render queue and the ocean will not look correct.

3. Component Reference

Ceto uses five components on the ocean prefab to run all of the supported features. Each component is responsible for managing a certain aspect of the ocean. With the exception of the Ocean component all components are optional and disabling or removing them will remove their features from the ocean.

The Ocean component.

The ocean component is the only required component. This component manages the other components, handles settings that have a global scope and contains a number of subsystems used for certain features. Just having an ocean component will not result in anything being rendered. It is purely a management script with the other components providing the actual features.

Disable Warnings: The ocean will report warnings if anything unusual happens during run time that you should be aware of but may not cause a critical error. Tick this box to prevent the warnings being shown.

Disable Info: The ocean will sometime report some information. These are like warnings but are not things that can cause a problem. Tick this box to prevent the information being shown.

Double Precision Projection: If ticked the projector math will be carried out in double precision. This is recommended as single precision can result in errors when the world size is large. Single precision may be faster on some systems.

Tight Projection Fit: Ceto uses a projector for the grid and the overlays. For technical reasons the actual projector position can not always match the cameras position. If tight projection fit is ticked the projector position will try and stay close to the camera. This gives better projection resolution. For some set-up like VR however where the camera has a large degree of freedom this tight fit can cause issues like the ocean mesh wiggling or breaking. If this occurs untick this option to place the projector further from the camera.

Sun: Drag a game object onto this that contains a directional light used for the sun in your scene. The suns direction and colour will be used for certain ocean features. This is optional and if left null the direction will default to up and the colour to white.

Level: This is the sea levels world Y position.

Height Overlay Size: This is the size relative to the cameras width and height of the buffer used for the height overlays. If no height overlays are added this buffer will never be created. Each camera that renders the ocean will have its own buffer.

Foam Overlay Size: This is the size relative to the cameras width and height of the buffer used for the foam overlays. If no foam overlays are added this buffer will never be created. Each camera that renders the ocean will have its own buffer.

Normal Overlay Size: This is the size relative to the cameras width and height of the buffer used for the normal overlays. If no normal overlays are added this buffer will never be created. Each camera that renders the ocean will have its own buffer.

Clip Overlay Size: This is the size relative to the cameras width and height of the buffer used for the clip overlays. If no clip overlays are added this buffer will never be created. Each camera that renders the ocean will have its own buffer.

Height Blend Mode: This is the blend mode for the height overlays. If two height overlays are at the same location then you can choose to add their values or take the max value. This only applies to the overlays not the waves generated by the spectrum.

Foam Blend Mode: This is the blend mode for the foam overlays. If two foam overlays are at the same location then you can choose to add their values or take the max value. This only applies to the overlays not the foam generated by the spectrum.

Default Sky Colour: If no reflection component is added this will be the colour used for the sky's reflection.

Default Ocean Colour: If no underwater component is added this will colour used for the ocean's refraction.

Wind Dir: The wind direction as an angle between 0 and 360. An angle of 0 would result in a wind direction vector of x=1, z=0.

Specular Roughness: This is the roughness value used for the BRDF lighting.

Specular Intensity: This is the intensity of the specular lighting.

Fresnel Power: Used to control the falloff of the Fresnel.

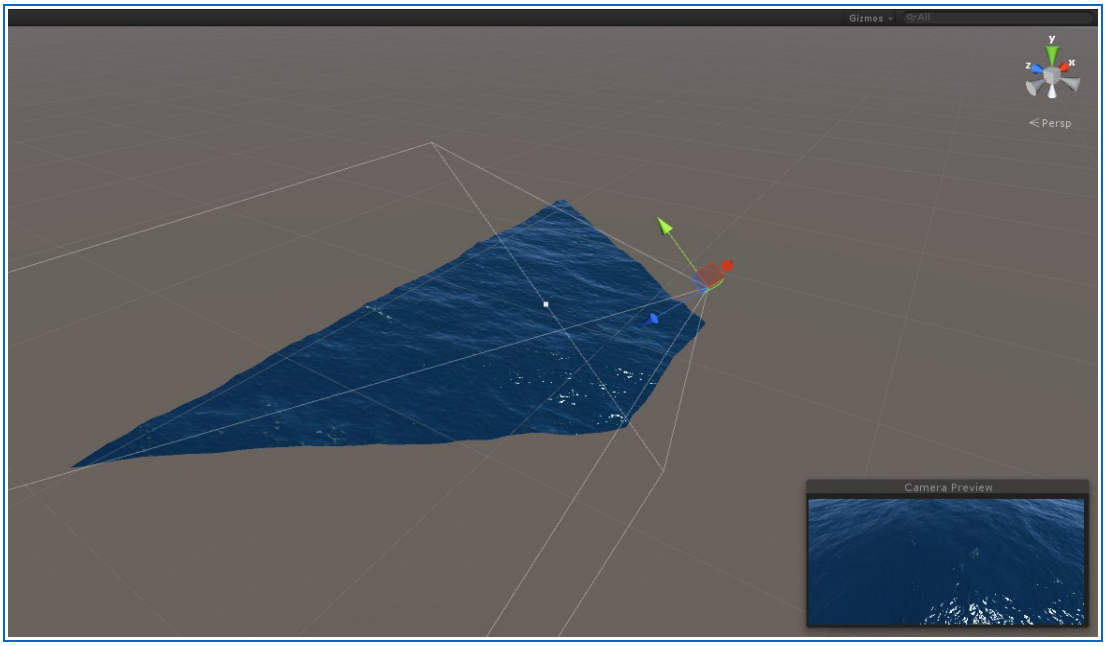
Min Fresnel: The minimum value used for the Fresnel.

Foam Tint: Use this to adjust the colour of the foam.

Foam Intensity: Adjust the intensity of the foam colour The foam tint will be multiplied by this number. Useful in HDR set-ups where the colour intensity may need adjusting.

Foam Texture 0 and 1: Provide a texture and the uv settings for the foam texturing. The two textures will be combined and averaged. If the texture is left as null a solid white texture will be used by default. Foam can be generated by the wave spectrum component and the foam overlays. This texture will be used for both.

The Projected Grid component.



The projected grid component is responsible for creating and managing the mesh used to render the ocean. If no projected grid component is add the ocean will not be rendered. This is useful for server set-ups that will only be generating the wave data and do not require anything to be rendered.

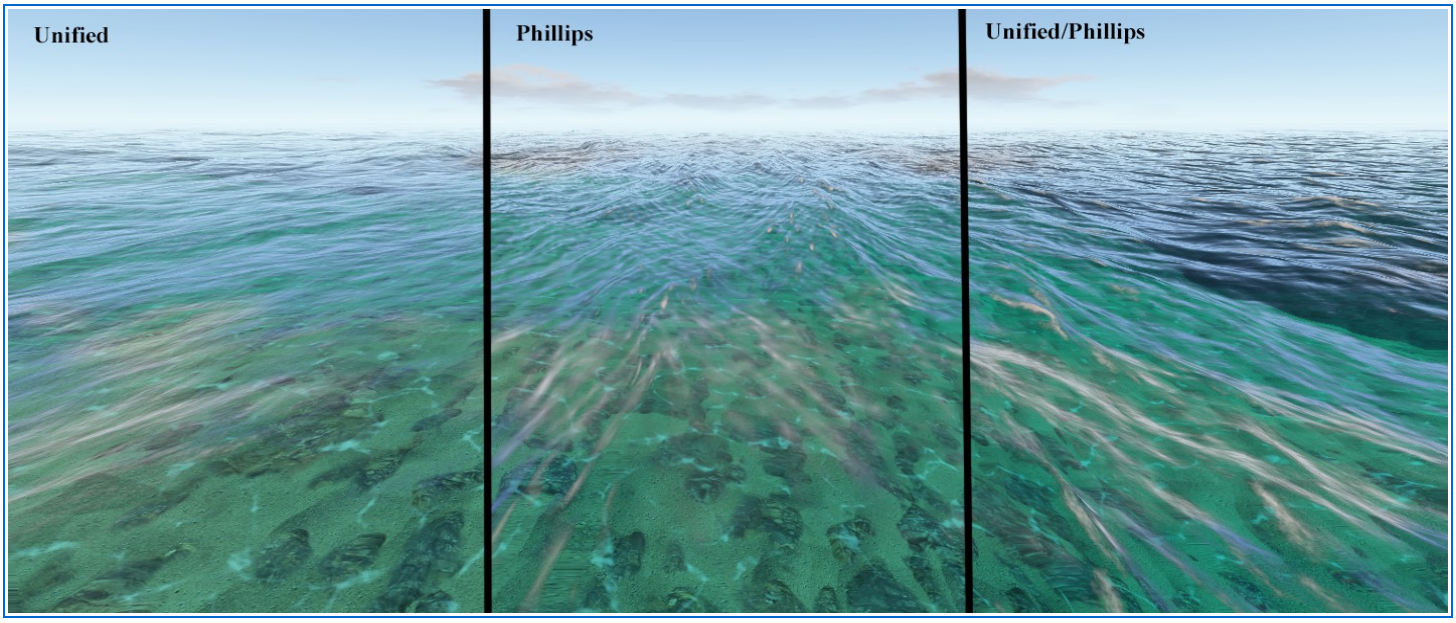
The image above is the projected grid mesh from the view of another camera. You can see the frustum outline of the main camera and how the mesh is only projected to where the camera is looking. This maximizes the mesh resolution by not wasting vertices where they can not be seen. The nature of the projection also places more vertices closer to the camera and less farther away.

Resolution: Controls the resolution of the mesh created. The mesh for the projected grid is created in screen space and the resolution is relative to the screen resolution. What this means is smaller screens require less vertices in the mesh to achieve the same result. A higher resolution may require the mesh to be split into many smaller meshes. Therefore a higher resolution will not only result in more vertices but potentially more draw calls.

Receive Shadows: Tick to enable shadows on the ocean mesh. If a transparent material is used then there will be no shadows as shadows are unsupported on transparent objects.

Ocean topside and underside material: These are the materials used to render the ocean. There can be two sides of the ocean, the topside and the underside. Each has different shader requirements so there needs to be two separate meshes with separate materials. If the material is not set the mesh will not be created. If there is no underwater component or the underwater component's mode is set to 'ABOVE_ONLY' then there is no need for the underside mesh and it will not be rendered.

The Wave Spectrum component.



The wave spectrum is responsible for generating the wave data. The wave data can consist of any combination of 3 sets of data, the displacement, the slope and the foam. The displacement is the amount of displacement the mesh vertices are moved on the y axis and/or the xz axis. The slope is the derivatives of the displacement and is used to create the normals. The foam is created using the Jacobian of the displacements. This method is quite different from other heights based coverage methods commonly used to generate foam.

The wave spectrum is not the only source of data used to create the waves with the other being the overlays. The textures generated by the wave spectrum are tiled over the mesh making the wave spectrum the only source of wave data that can cover the ocean plane on a global scale. The overlays are only designed for local effects.

The wave spectrum script contains an object that holds the spectrum data that has been created based off the current settings. There are four settings which affect the spectrum data. The Fourier size, the wind speed, the wave age and the wind direction (on the ocean component). Changing any of these will result in the spectrum data being recreated. This is a time consuming operation and will be carried out on a separate thread to prevent blocking the main thread. This operation can take from 20ms to 1 second depending on the Fourier size used. At the recommended Fourier size of 64 recreating the spectrum will take about 80ms. While the spectrum is being created it will ignore any changes to the Fourier size, wind speed, wave age and wind direction.

Fourier Size: The Fourier size has the greatest impact on the performance and the quality of the wave data created. This setting will determine the size and if the transform is carried out on the GPU or CPU. Only a limited number of sizes are available for the CPU option due to performance limitations. Which option to use depends on the type of application being developed.

For games that require interaction with the ocean surface for buoyancy calculations it is highly recommended to use a CPU option. This way the displacement data will be available for height queries and will not require a read back from the GPU which can have a high impact on the frame rate.

It is only recommended to use a GPU option if you don't need buoyancy so can disable the read back or your applications is some sort of visual demo where performance is not critical. If you keep the wave height quite small it is often not even necessary to have the displacement data and just use the sea level for the buoyancy calculations.

It is only the displacement data that will run on the CPU or GPU. If running on the CPU then multi-threading will be used to increase performance and prevent the main thread being blocked. The slope and foam data is always created on the GPU regardless of which option is selected.

Spectrum Type: There a few spectrum type that you can choose from and each will give a unique look to the waves. The options are Unified, Phillips, Unified/Phillips. Unified is the recommended option as the spectrum created will best represent waves of different scales and the motion is more realistic. Phillips is simpler and faster to create and in some ways has a more 'watery' look. The movement however will only slosh back and forth and it does not scale well for larger waves. The Unified/Phillips option is just a blend of the type with the lager grid sizes using Unified and the smaller Phillips.

Number Of Grids: The spectrum will create a texture containing the waves which is then tiled over the ocean. If only one texture was created the the tiling pattern would be quite visible. Ceto allows you to use 1-4 grids each of which have there own wave textures and are tiled at different scales. This allows you to tile the textures while minimizing any repeating patterns. Each grid generates unique wave data and there for requires its own FFT. There fore reducing the number of grids is a good way to increase performance.

Disable read back: This will disable reading the displacement data back from the GPU. Reading back the data is a DX11 feature only. Trying to read back data on a platform where it is not supported will result in a warning being printed and then falling back to running the transform on the CPU.

Disable Displacements, Slopes or Foam: This will disable the data from being created. If you only require the displacement data for buoyancy calculations and no rendering is required (on a server for example) then you will not need the slopes or foam so they can be disabled.

Texture Foam: If enabled this will apply the foam texture (set on the ocean component) to the foam generated by the spectrum.

Choppyness: This controls the amount of displacement on the xz axis. Choppy waves look more realistic but require more data to be created. Having the choppiness set too high can cause the tips of the waves to curl back on themselves. Setting the choppiness to 0 will disable the displacement data on the xz axis from being created so can increase performance.

Foam Amount: Increases the amount of foam.

Foam Coverage: Controls the amount of the wave peak which is covered by the foam.

Wind Speed: A higher wind speed will create larger waves. This doesn't just scale the waves up. The spectrum calculations take the wind speed into account so the look of the waves as well as the size will change with the wind speed.

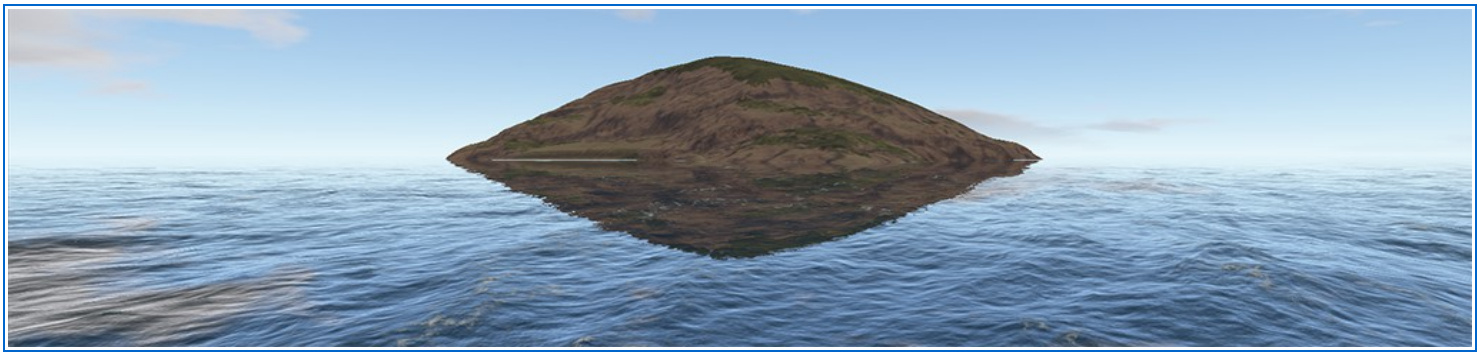
Wave Age: Controls how long the waves last for. The spectrum calculations take the wave age into account so the look of the waves will change. It's actually the inverse of this value that gets used so a lower value will result in longer waves.

Wave Speed: Scales the time value used to create the waves resulting in faster or slower moving waves.

Grid Scale: Scales the uv's used to sample the waves.

Wave Smoothing: Adjusts the way the mipmaps are sampled. Helps reduce wave aliasing issues when the projected grid resolution is low but will smooth out some of the wave detail.

The Planar Reflection component



Provides reflections using the planar reflection method. If no planar reflection component is added then the ocean will use the default sky colour as the reflection colour

Reflection Mask: Use this to select which objects get rendered for the reflections. It is highly recommended to keep the number of layers used for the reflections to a minimum.

Clip Plane Offset: The planar reflections use a clip plane to cut off anything that would be below the water plane. The plane needs a small offset so the clipped area is not too visible. If your world scale is drastically different than Cetos demo scene (1 unit = 1 meter) then this offset may have to be adjusted.

Resolution: The resolution of the reflection textures relative to the screen size.

Fog In Reflections: Tick this to enable fog on the reflection camera.

Sky Box In Reflections: Tick this to include the sky box in the reflections.

Blur Mode: If the blur mode is not set to ‘OFF” then the reflections will be blurred. Blurring the reflections can help hide artefacts that occur due to the limitations of the planar reflection method. A slight blur can also be more visually appealing.

Blur Iterations: A higher iteration will give more blur.

Reflection Tint: Tints the colour of the reflections.

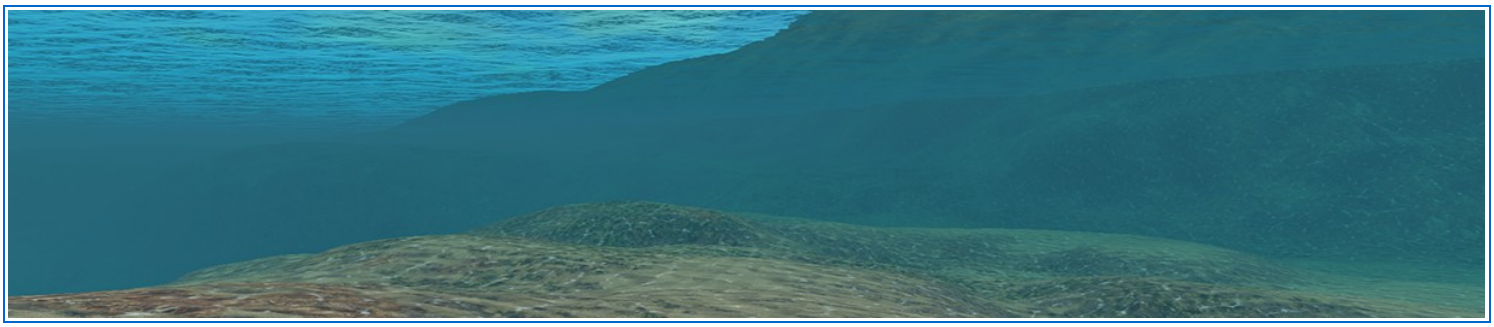
Reflection Intensity: Adjusts the intensity of the reflections. Useful for HDR.

Reflection Distortion: The amount of distortion to the reflection uv’s.

Ansio: The anisotropic filtering value used for the reflection texture.

If the waves are large causing a high amount of distortion to the ocean mesh then the planar refection method no longer works correctly. This is because the distorted mesh is no longer a flat plane but the planar reflection method always presumes it is. It is recommended to heavy blur the reflections when the waves are larger and therefore hiding the reflection artefacts. This is currently the only solution to the issue.

The Underwater component



Provides the depth and mask texture needed to apply the underwater effect. Settings related to the absorption and inscatter of light underwater are also set here. There are two distinct areas where the underwater effect is applied. When the camera is looking down on the ocean mesh and when the camera is under the ocean mesh. These two areas have some separate settings and are referred to as ‘above’ and ‘below’. If no underwater component is added then the default ocean colour will be used as the refraction colour

Adjusting the look of the underwater can be tricky as there are a lot of settings. See the tutorial section on how to adjust the effect.

There is also an underwater post effect script that applies the ‘fog’ effect when the camera is under the ocean mesh and is attached to the camera. Some of the settings on this component will affect the post effect script.

Underwater Mode: If the camera will only be looking down on the ocean and will not go under the mesh then set this to ABOVE_ONLY. If the camera will go below the mesh set it to ABOVE_AND_BELOW.

Depth Mode: To apply the underwater effect the shaders needs to know some depth information at that pixel. This information can be gathered in two ways. It can be created by rendering any mesh that’s underwater in a separate pass using a replacement shader or from directly sampling the depth buffer. To use the separate pass set this to USE_OCEAN_DEPTH_PASS and to use the depth buffer set this to USE_DEPTH_BUFFER.

Using the depth pass will increase the number of draw calls but is good for situations where you cannot sample from the depth buffer. Using the depth buffer means less draw calls but can only be done in the transparent queue and the shader needs to convert the depth value to a world position so can suffer from precision issues..

Ocean Depths Mask: If the depth mode is set to use the depth pass then this is the layer mask that controls which layers get rendered. It is recommended you organise your objects into layers that never go under the ocean surface and unselect them from the mask.

The depth pass uses a replacement shader so the objects render type must be in the shader. Some custom shaders may not have their render type added and speed trees currently do now work in the depth pass. They should not be included in the mask.

Bottom Depth: To apply the under water effect the ocean needs to be enclosed in a mesh. If the camera passes below this level then the effect will no longer be applied. This setting is the distance to the bottom of the enclosing mesh relative to the ocean level. The camera far plane should be about twice this distance to ovoid the mesh getting clipped by the far plane.

Edge Fade: Controls the strength of the fading applied where the ocean meets other objects like the terrain. A higher value gives a softer edge.

Absorption R, G and B: These are the absorption coefficients that are used to calculate the transmittance of light through the water. The transmittance causes light from the RGB channels to be lost at rate based on the coefficients. The R coefficient is the highest meaning red light will be lost at a higher rate therefore making the water appear less red. This is why water appears blue/green because it has less red light than blue and green light.

Above Absorption Modifier: This allows you to tweak the look of the absorption on the top mesh seen when looking down on the ocean. The modifier has three values, the scale, the intensity and the tint. The tint and intensity are multiplied by the final colour after the absorption is calculated. Adjusting the intensity is useful when using HDR. The tint is a quick way to change the final colour The scale is used to modify the distance the absorption calculation uses. A higher scale will mean the light travels further so the absorption will be stronger. A scale of 0 will mean no absorption will be applied.

Below Absorption Modifier: Works the same as the above modifier but applies to the areas of the screen that are under the ocean's surface.

Subsurface scatter Modifier: Works the same as the above modifier but applies to the sub surface scatter that is added to the final ocean colour on the top mesh.

Above Inscatter Modifier: This controls the settings for the inscatter which is the fog effect applied to objects underwater. The inscatter has 4 settings, the scale, intensity, mode and colour The fog effect is applied based on distance. The scale settings will adjust the distance used and the mode provides some options to how the effect is applied. The colour in the fog effects colour and the intensity scales the colour

Below Inscatter Modifier: Works the same as the above modifier but applies to the areas of the screen that are under the ocean's surface.

Above Refraction Intensity: The refraction grab pass is scaled by this number and controls the intensity of the refraction colour Useful for HDR. The is the intensity used for the top side of the ocean.

Below Refraction Intensity: The refraction grab pass is scaled by this number and controls the intensity of the refraction colour Useful for HDR. The is the intensity used when under the ocean mesh (i.e. the under side mesh and the post effect).

Refraction Distortion: The amount of distortion to the refraction uv's.

Caustic Modifier: These settings can be used to adjust the look of the caustics. The caustics are done by using a static caustic texture and then distorting it based on the wave normals. The distortion setting can be used to adjust this. The caustics will fade out the farther the camera is from the ocean surface. The depth fade setting will adjust this. The colour and intensity will tint the caustics.

Caustic Texture: The texture and the uv scale used for the caustics.

4. Sub-Systems

In addition to the 5 main components that make up the ocean Ceto also contains a number of sub-systems. These sub-systems provide addition ways to modify or interact with the ocean. These could be through components added to other game objects or by directly interaction with the ocean from scripts.

Wave Queries

To support features like buoyancy it is common to have to query what the wave height is at a world location. Ceto supports this through a feature called wave queries. Using a wave query you can request the ocean to return some information about the ocean surface at that location. Typically this is the wave height but other information can also be returned.

Ceto provides a number of functions to query the waves. Some are very simple but provide limited information while others are more complicated but can return addition information.

All of the following code presumes you are using the Ceto namespace.

To query what the wave height is at a location use the following code. The x and z values are the world space position.

```
float height = Ocean.Instance.QueryWaves(x, z);
```

If the ocean mesh has been clipped then you may want to handle the buoyancy differently. You can get the wave height and if the location is clipped by using the follow code.

```
bool isClipped;
float height = Ocean.Instance.QueryWaves(x, z, out isClipped);
```

You can also provide a wave query object to the ocean.

```
WaveQuery query = new WaveQuery(x, z);
Ocean.Instance.QueryWaves(query);

float height = query.result.height;
bool isClipped = query.result.isClipped;
```

Any result returned by the query is in the result struct in the query.

By providing a wave query object some extra features can be used.

```
WaveQuery query = new WaveQuery(x, z);

query.minError = 0.1f;
query.overrideIgnoreQuerys = true;
query.sampleOverlay = true;
query.sampleSpectrum[0] = true;
query.sampleSpectrum[1] = true;
query.sampleSpectrum[2] = false;
query.sampleSpectrum[3] = false;
query.posX = x;
query.posZ = z;
query.mode = QUERY_MODE.POSITION;
query.tag = yourNumber;

Ocean.Instance.QueryWaves(query);

float height = query.result.height;
```

Min Error: A query may need to sample the waves multiple times before it can determine the height at a location. The min error is the distance in world space which the query has to be to the requested position before it will return. For example if the min error is set to 0.1 and the requested x and z position is at 0,0 then the returned height maybe from any location in the area from -0.1,-0.1 to 0.1,0.1.

A higher min error will result in faster queries but the result will be less accurate.

Sample Overlays: Set to true if you want the query to include any modifications to the wave height from overlays. This also includes clipping. Overlays are more expensive to sample so setting this to false if you know the position does not include any overlays can result in faster queries.

Sample Spectrum: Set to true if you want the query to sample the height from the wave spectrum. The wave spectrum has four grids and you can set which grids to sample using the array positions 0 to 3. Grids 2 and 3 in the wave spectrum are the smallest waves and are not sampled by default as they don't add much to the wave height.

Pos X and Z: This is the requested position to sample.

Override Ignore Querys: Individual overlays can chose to ignore the wave query. This is useful for overlays that don't modify the wave height much so are not worth sampling. Setting override to true will sample the overlays anyway.

Mode: Queries have a mode that determines what they return as a result. The follow are the possible query modes.

POSITION: This is the default mode and will return the wave height.

CLIP_TEST: If in this mode the query will only check to see if the position has been clipped and will not sample the height. This makes the query much faster if only a clip test is needed.

Tag: This is not used by Ceto and will never be changed. Use this to give your queries a unique number (like a array index) if you need to.

You can also pass a array of queries. There is currently no performance benefit to passing a array. It can sometimes be more convenient.

```
WaveQuery[] queries = new WaveQuery[100];

for (int i = 0; i < 100; i++)
{
        queries[i] = new WaveQuery(x, z);
}

Ocean.Instance.QueryWaves(queries);

for (int i = 0; i < 100; i++)
{
        float height = queries[i].result.height;
}
```

The result struct in the wave query also has some other information that can be useful. For example the amount of displacement applied to the waves at the query location is also returned. This can be useful to create a velocity vector so objects will drift in the current.

```
            WaveQuery query = new WaveQuery(transform.position);
            Ocean.Instance.QueryWaves(query);

            float dx = query.result.displacementX;
            float dz = query.result.displacementZ;
            float speed = 0.05f;

            Vector3 velocity = new Vector3(dx, 0.0f, dz) * speed;
            transform.position = transform.position + velocity;
```

The height for the waves comes from two different sources, the wave spectrum and the overlays. You can work out how much each contributes to the height from the query.

```
        //total height is in oceans local space (ie presumes sea level is 0)
        float totalHeight = query.result.height - Ocean.Instance.level;
        float overlayHeight = query.result.overlayHeight;
        float spectrumHeight = totalHeight – overlayHeight;
```

The wave normal can also be queried. Just beware that this normal is not exactly the same as the one used for rendering and has less detail. It is calculated using finite difference from the wave height and one normal query requires 4 height queries so it is more expensive to calculate.

```
        Vector3 normal = Ocean.Instance.QueryNormal(transform.position.x, transform.position.z);
```

Wave Overlays

The wave spectrum component creates the waves that cover the ocean plane on a global scale. Sometimes you might want to modify the waves in a local area. This is what the wave overlays are for. A overly can modify a local area of the ocean in many ways. It can add waves or mask them out, add normals or mask them out, and foam or mask them out or it can clip away the mesh.

Adding the wave overlay from code is not very user friendly and it is not expected you do this. There are a number of helper scripts (like the shore mask script) to do this. See the tutorial section for more information about the helper scripts.

The overlays must have the values in the alpha channel. If your using a grey scale image make sure you tick the 'Alpha from grey scale' box on the texture.

The overlays are created by adding a WaveOverlay object to the ocean. This can be done in code as follows.

```
WaveOverlay overlay = new WaveOverlay(rotation, pos, halfSize, duration);
Ocean.Instance.OverlayManager.Add(overlay);
```

The ocean will then handle the rendering of the overlay. You must call update on the overlay every frame however. This is not done by default by the ocean so you can make optimizations and only update the overlay when you need to, for example if it moves.

If you don't need the overlay to render you can set its hide variable to true and if you don't need the overlay any more you can set its kill variable to true and the ocean will remove it in its next update.

```
overlay.Kill = true;
overlay.Hide = true;
```

There are a number of texture settings on the overlay which contain a number of settings relating to that texture. There is a height, normal, foam and clip texture. You can directly modify the textures on the overlay as follows.

```
overlay.HeightTex.alpha = 0.0f;
```

The follow is a explanation of each setting. Not all the overlay textures have all the options. The textures are optional. If left blank the ocean will just skip that part of the overlay.

Tex: This is the actual texture that will be used for the overlay. It must have the info in the alpha channel. If its for a height or clip texture it must also have read/write enabled if you want the queries to be able to sample it.

ScaleUV: The uv's are scale by this when rendered.

Alpha: The sampled value is multiplied by this.

Mask: The mask is used to remove the feature from the ocean. For example the mask on the height texture will mask out the wave heights. How the mask is applied depends on the mode. The mask is applied where it is white.

Mask Alpha: The sampled mask value is multiplied by this.

Mask Mode: The mask mode affects how the mask is applied.

There are currently four modes.

WAVES: This will mask out the wave data created by the spectrum.

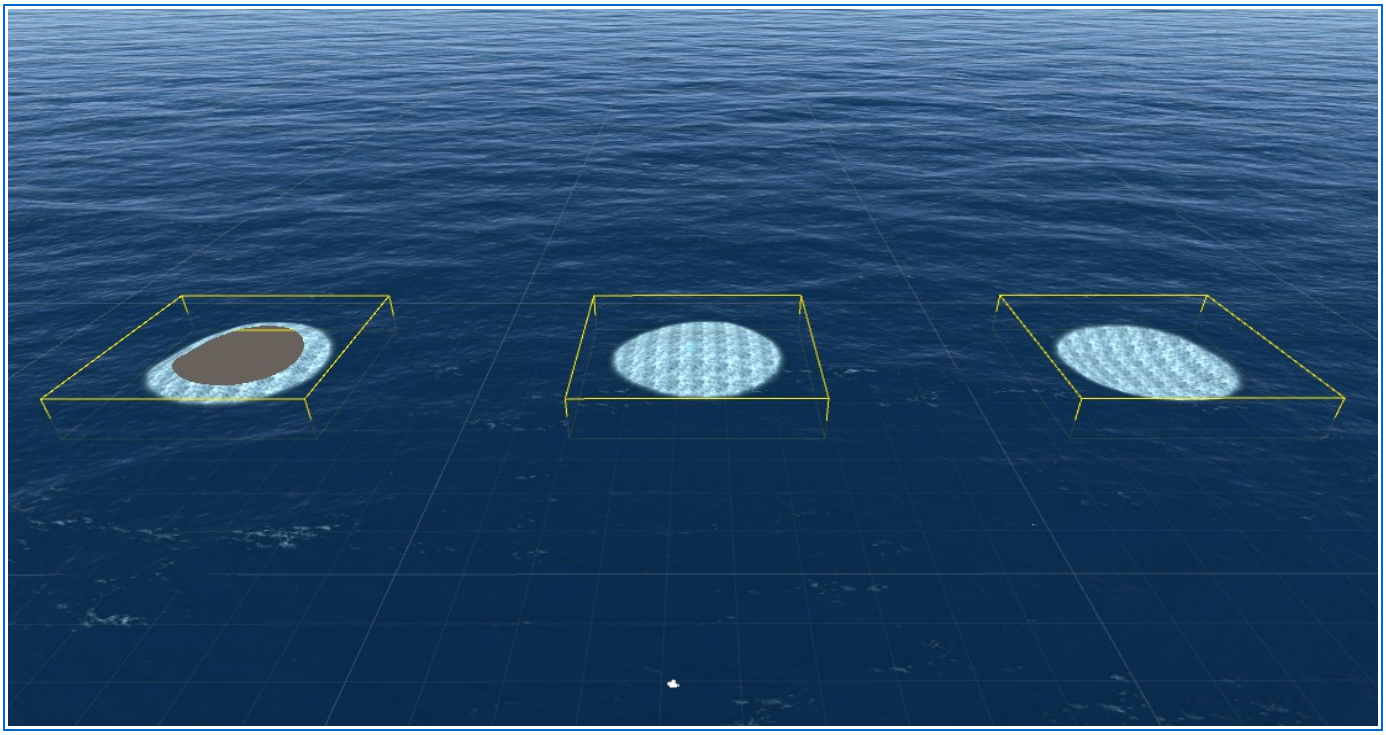
OVERLAY: This will mask out the overlay. Only the one its applied to, not any other overlays.

WAVES_AND_OVERLAY: This will mask both the overlay and waves.

WAVES_AND_OVERLAY_BLEND: This will mask both waves and overlays like above except the waves will be masked in the white parts and overlays in the black parts result in the two blending together.

Ignore Querys: If a overlay modifies the wave height then it needs to be accounted for when querying the waves. This can be expensive and for small changes it might not be necessary. Tick this to have the queries ignore this overlay and not sample it.

Below is a image of the example overlays in the demo scene. Each overlay adds foam and modifies the wave height at that location. The second overlay adds height as it has a positive alpha and the third overlay removes height as it has a negative height alpha. The first overlay adds height but also clips away the ocean at the same location.



For more information see the tutorial section on how to use wave overlays.

The Underwater post effect

This script is attached to a camera and uses a image effect to add the under water fog. The settings for the inscatter used to create the fog effect (like colour and scale) are on the underwater script on the ocean component but for everything else the settings are on the post effect script.

If there is no ocean in the scene or the ocean has no Underwater script or the underwater mode on the Underwater script is set to above only then there will be no post effect rendered.

Disable On Clip: If the ocean mesh has been clipped away (like in a hole for a cave) then you will most likely not want the post effect to run. Tick this box if you want the script to check if the camera position is over a clipped area of the mesh and not run if it is.

Control Underwater mode: The underwater mode on the ocean determines if certain resources need to be created. If the underwater mode is set to ABOVE_AND_BELOW then its expected that the camera maybe under the water and that it should enable the underside mesh and render the mask. These take time to do so you would only want that mode enabled when you need it. If this box is ticked then this script will take control of setting the mode so the resources will only be rendered when this script needs to run. The script will try and calculate if under the water could potential be seen and if it could then the script will set the underwater mode to ABOVE_AND_BELOW, if not it will set it to ABOVE_ONLY.

If you have multiple cameras in the scene that each have a post effect script then don't have this box ticked as they will both be trying to set the underwater mode and will conflict with each other.

Attenuate By Sun: If ticked the underwater fog colour will be multiplied by the suns colour so it will go dark as the sun sets.

Blur Mode: If the blur mode is not set to 'OFF" then the areas under the water will be blurred. The image will also be down sampled by a certain amount before being blurred.

Blur Iterations: A higher iteration will give more blur.

Under Water Post Effect Sdr: This is the shader used for the post effect.

6. Tutorials

Learning how to use a new asset is always time consuming. The follow should help ease this process by providing tutorials to the most commonly asked questions.

How to set Ceto up for a sever

In Ceto the rendering of the mesh is decoupled from the generation of the height data. This design makes Ceto quite easy to set up on a server. A typical situation is to have the server generate the wave data with no rendering and then sample the heights for things like buoyancy and then send the resulting object positions to the clients. The objects positions will then be synchronized amount all clients. The clients can then separately render the ocean mesh and since they don't need the wave data they can perform the wave FFT on the GPU to maximize performance. All the clients need is for the time value used for the FFT to be synchronized with the server so the render waves match the object positions from the server.

Step 1.

Create an empty game object in your scene and then add the Ceto/Components/WaveSpectrum component. Note that adding this script will also add a Ocean script as its a required component. These two scripts are all that's required to generate the wave data without actually rendering anything.

Step2.

On the WaveSpectrum script now choose a Fourier size. Please see the usage decision section for more information about which size is best. For a server where the wave data is needed to sample the heights its recommended to choose a MEDIUM_64_CPU or HIGH_128_CPU option.

The size chosen must match the size used for the clients. For example if the server size 128 and the clients is 64 then the servers sampled heights will not match the clients rendered waves.

The server can use a CPU size and the clients can have a GPU size. The wave data will match in this case. If the clients don't need to sample the wave data then its actually recommended to do this as it will be faster for the client.

Step 3.

If you do decide to use a GPU size for the server then make sure that 'Disable Read Backs' is not ticked. The server will need to support DX11 for this to work.

If you do decide to use a GPU size for the client then make sure that 'Disable Read Backs' is ticked. The client does not need to support DX11 in this situation. This presumes that the wave data does not need to be sampled by the client.

Step 4.

On the wave spectrum tick 'Disable Foam' and 'Disable Slopes' boxes. These are only needed for rendering the ocean so are not needed for a server.

Step 5.

The waves are generated based of a time value. As long as two computers use the same time value then the waves will look the same. By default Cetos uses Unity's time value. For a server set up you will want better control over what time value is used. Ceto provides a interface for the time used for the ocean.

```
public interface IoceanTime {
        /// <summary>
        /// The current time in seconds.
        /// </summary>
        float Now { get; }
}
```

If you wish to change what time value the ocean uses then simply create a new class that implements the interface and add it to the ocean in a start function somewhere. The follow is a example of a game object implementing the IOceanTime interface.

```
using Ceto;

public class OceanTimeExample : MonoBehaviour, IOceanTime //Implement the interface
{

    /// <summary>
    /// Add the now property which is the time Ceto will use.
    /// This should be the time in seconds since some arbitrary
    /// event like app or server start up.
    /// It does not really matter what value is used. As long as all
    /// clients use the same value the ocean will look the same.
    /// </summary>
    public float Now { get; private set; }


    void Start ()
    {

        //Add interface to the ocean.
        Ocean.Instance.OceanTime = this;

    }

     void Update ()
    {

        //Update the time value
        Now = YourTimeValueInSeconds;

    }
}
```

How to use wave overlays

Cetos wave overlays provide a method to modify the waves in a local area. Ceto provides a number of helper scripts that can be used to add a wave overlay to the ocean and control its setting. The AddWaveOverlay script is one of these scripts and is just a generic overlay that exposes all the main feature.

This tutorial is more of a demonstration of what a wave overlay can do rather than how to achieve a specific effect.

Step 1.

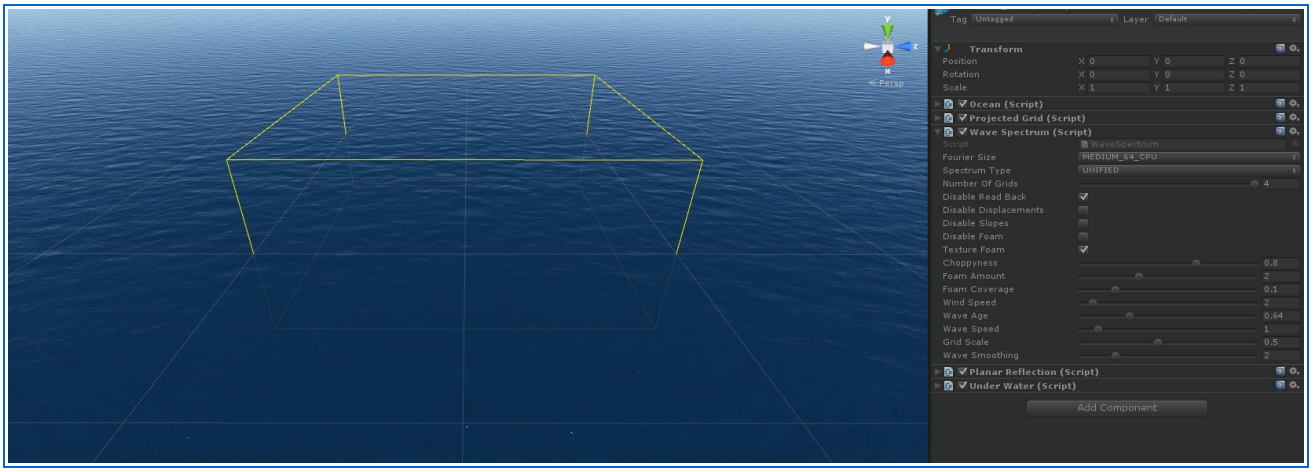
Open Cetos demo scene or create a new one and add Cetos ocean prefab. On the ocean set the wave spectrum wind speed to 2 so big waves don't obscurer the overlay effect

Step 2.

Create a new game object and add the Ceto/Overlays/AddWaveOverlay component. Move the game object to the position you want using the transform. The actual overlay will always be on the ocean surface so the transforms Y setting is not used.

Step 3.

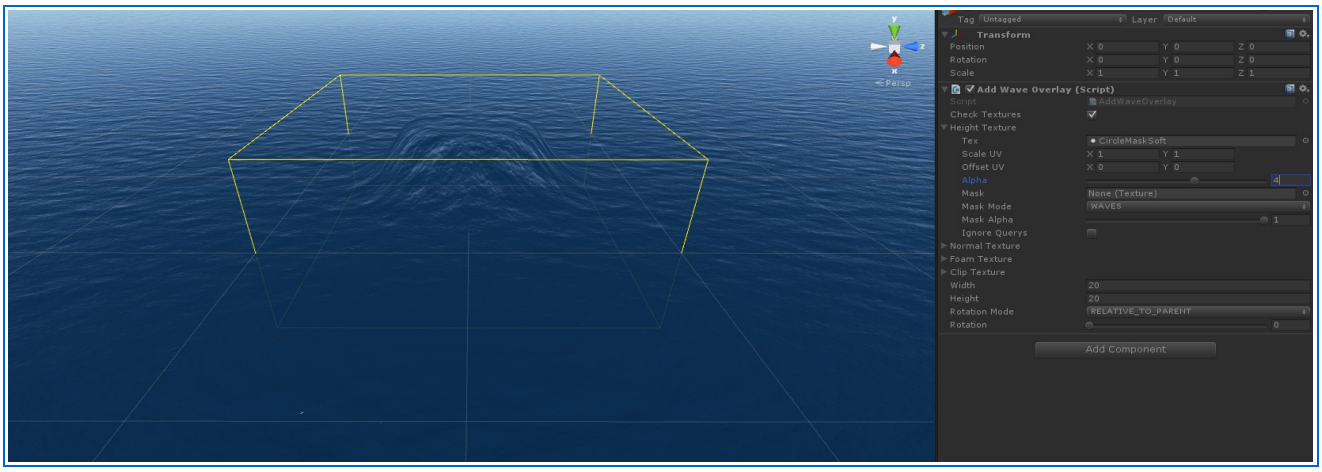
On the AddWaveOverlay script there is a width and height. Adjust that to the size you want. For this example I will use 20 for both values. In the scene window there should be a yellow box indicating the overlays size and rotation. The overlay needs textures set on certain slots for any effect to happen. At the moment it will do nothing.



Step 4.

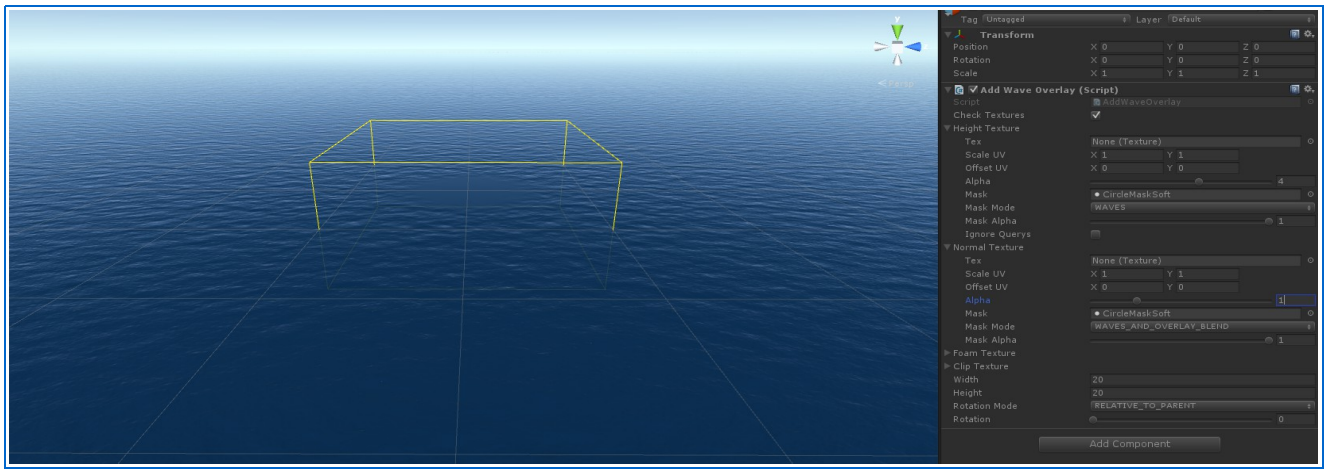
In the Ceto/Textures folder there are a few basic textures. Take the CircleMaskSoft texture and put it on the height tex slot on the wave overlay. Then set the alpha to 4. Make sure the mask mode is set to WAVES. This should be the default value but there is a bug in the current version where its not. Run the scene and you should see a bump on the ocean surface. Adjust the alpha value to modify the overlays height.

Note that the texture has 'Alpha from greyscale' ticked in the textures advanced setting. The overlays sample from the alpha channel so it is most convenient to create a greyscale image in photoshop and then tick this box. Note also that the texture has 'read/write' ticked in the textures advanced settings. If an overlay is going to modify the wave heights then it needs this box ticked if you want the wave queries to take it into account when querying the waves.



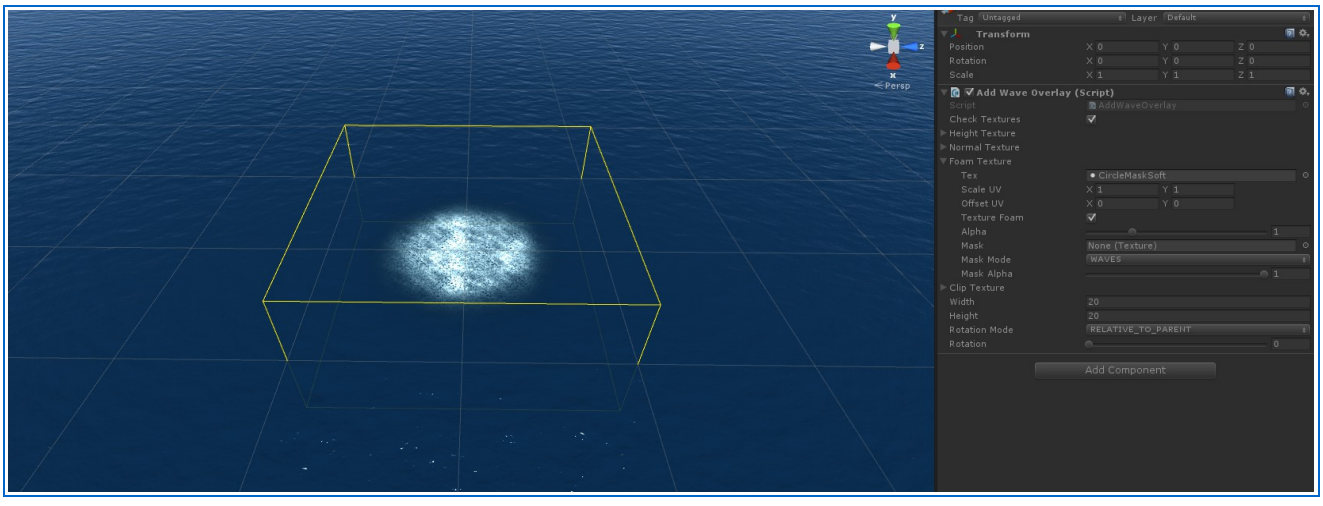
Step 5.

Set the height texture to none and drag the same texture onto the height mask and the normal mask slot. The overlay will now mask out the waves height and normals generated by the wave spectrum. The normal mask mode should be on WAVES_AND_OVERLAY_BLEND. Run the scene and there should be a smooth spot on the ocean where the mask is. The effect maybe hard to see. If so you can increase the wind speed on the wave spectrum and the masked area should be easier to see.



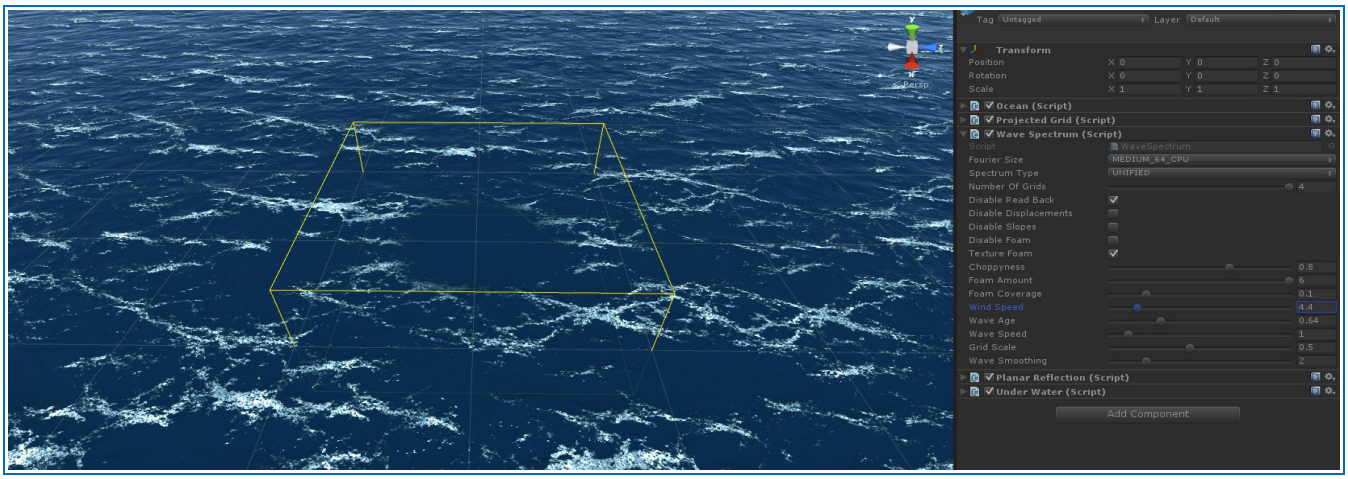
Step 6.

Now set the mask textures to none and drag the same texture onto the foam tex slot. Run the scene and you should see a white patch of foam where the overlay is.



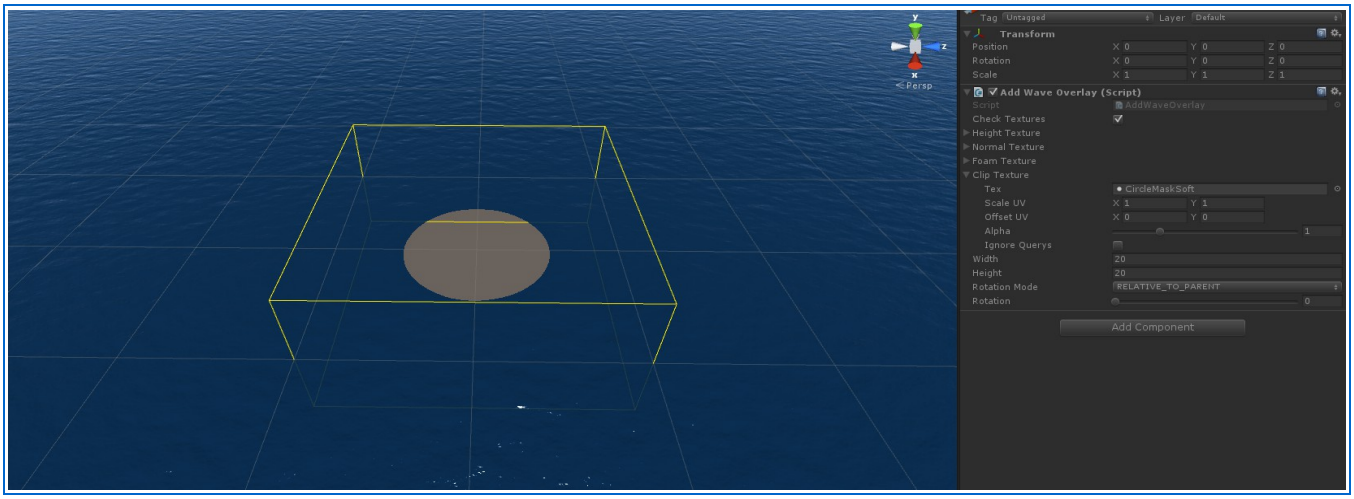
Step 7.

Set the foam texture to none and drag the same texture onto the mask slot. Run the scene and then click on the WaveSpectrum component. Increase the foam amount setting. You should see the foam generated by the spectrum increase except in the area that has been masked. You might also have to increase the wind speed a little to make the effect more visible.



Step 8.

Set the foam mask to none and now drag the same texture onto the clip tex slot. Run the scene and you should see a hole cut in the surface of the ocean. This is useful for removing the ocean mesh inland so the terrain can go below the ocean level without the ocean showing. If the UnderWaterPostEffect script on the camera has 'disable On Clip' enabled then the effect will be disabled when the camera is over a clipped area.



How to export shoreline masks for a terrain

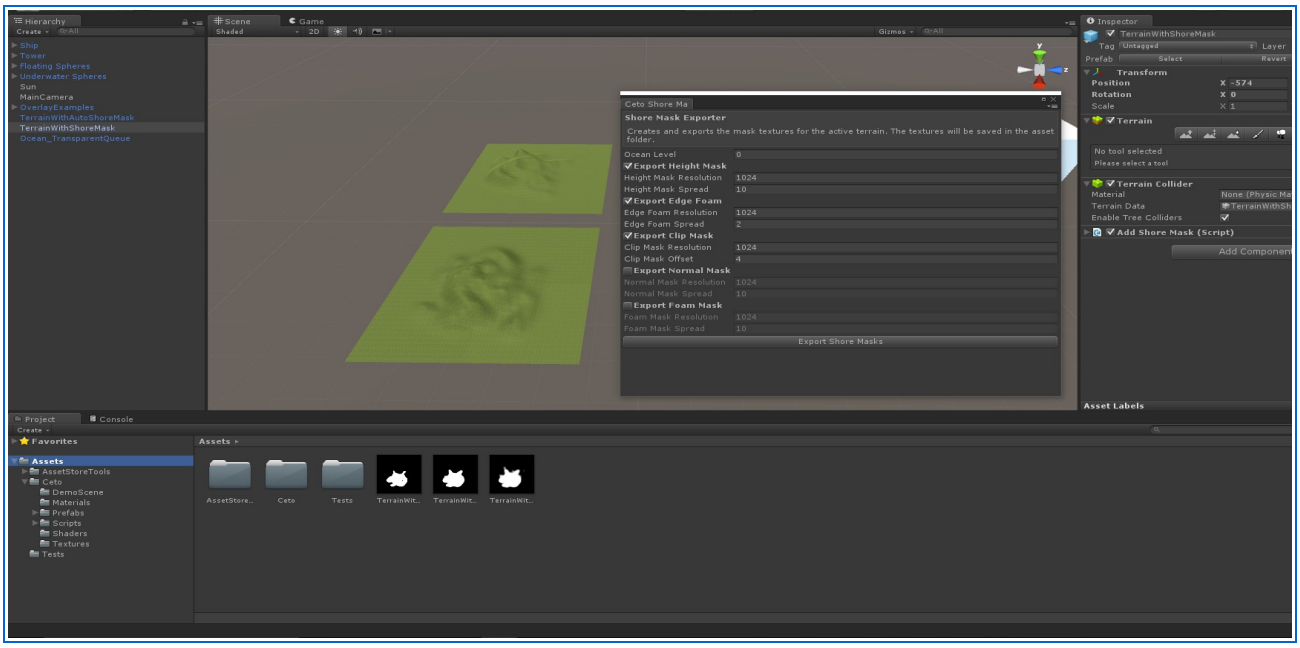
Ceto uses the wave overlays to add a shore line effect around your terrain but to do this it needs the mask textures. You can make them in other programs like World Machine or Photoshop but it is most convent to do so within Unity. Ceto provides a editor utility to export the textures for a terrain. Once you have the textures you can then use the AddShoreMask component.

Step 1.

First make sure you have selected the terrain in the scene you want the masks to be exported from.

Step 2.

Then go to Window->Ceto->Shore Mask Exporter. The follow window should appear.



Step 3.

Set the ocean level setting to match what you use as the level on the ocean.

Step 4.

Select which masks you want to export.

- The height mask will mask out the waves near the shore.
- The Edge foam will add a rim of foam around the terrain.
- The Clip mask will cut out the ocean where it is under the terrain.
- The normal mask will mask out the spectrum normals near the shore.
- The foam mask will mask out the spectrum foam near the shore.

Each mask is optional and the normal and foam are not selected by default. I find that the height mask can also be used for the normal and foam masks slots on the AddShoreMask component.

Each mask has a spread setting and increasing it will help spread the mask out from the shore. The value is based of the distance from the terrain to the ocean so areas where the terrain is steep may have a sharp mask boundary.

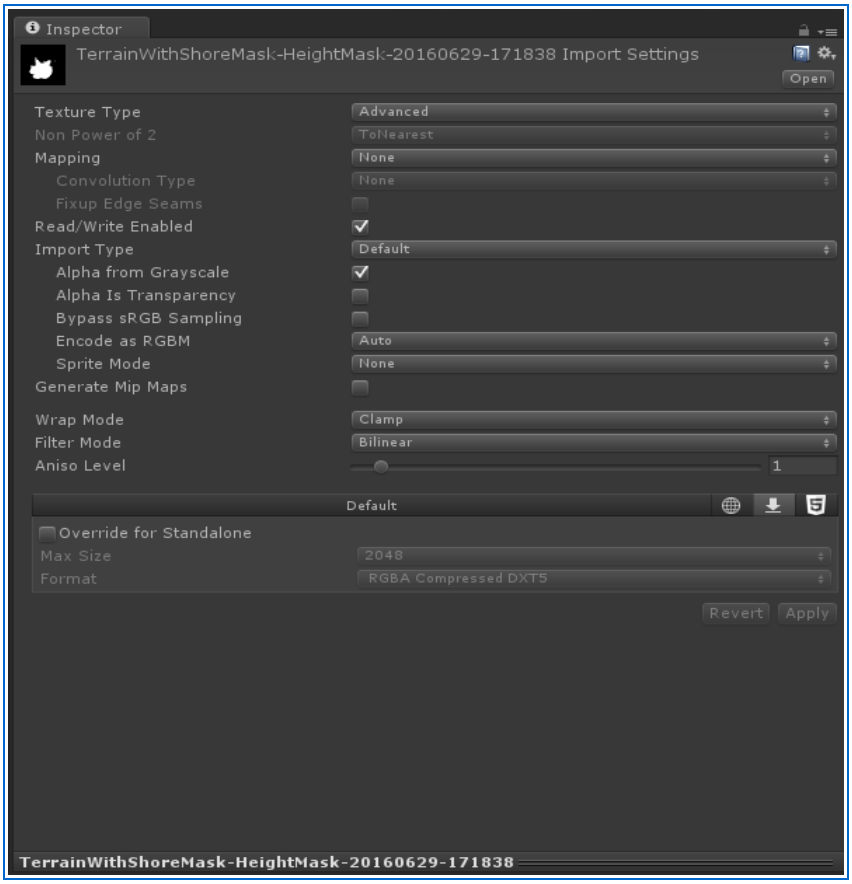
The clip mask instead has a offset which is the distance from the ocean the clip will be applied.

Step 5.

Hit the Export Shore Mask button. The textures should be saved to the asset folder.

Step 6.

You need to click on each texture and make sure the advanced texture settings match the ones in this image.



Make sure the 'Alpha from Grayscale ' is enabled.

For the height and clip masks make sure 'Read/Write' is enabled.

Make sure the wrap mode is set to clamp.

You don’t need Mip Maps so you can leave them disabled

How to create a shoreline effect

Ceto provides two helper scripts to apply a shoreline effect to a terrain. One is called AddShoreMask and the other AddAutoShoreMask. The first requires you to provide the mask textures. This means it is more work to use but gives you better control as you can open the textures in Photoshop and edit them if needed. The second script applies the shore line effect automatically and is calculated from the terrain on start up. This is much easier to use but you have less control over the effect and it will be slow to calculate for large terrain.

This tutorial is broken up into two parts. One for the AddShoreMask and the other for the AddAutoShoreMask.

Using the AddShoreMask component.

Step 1.

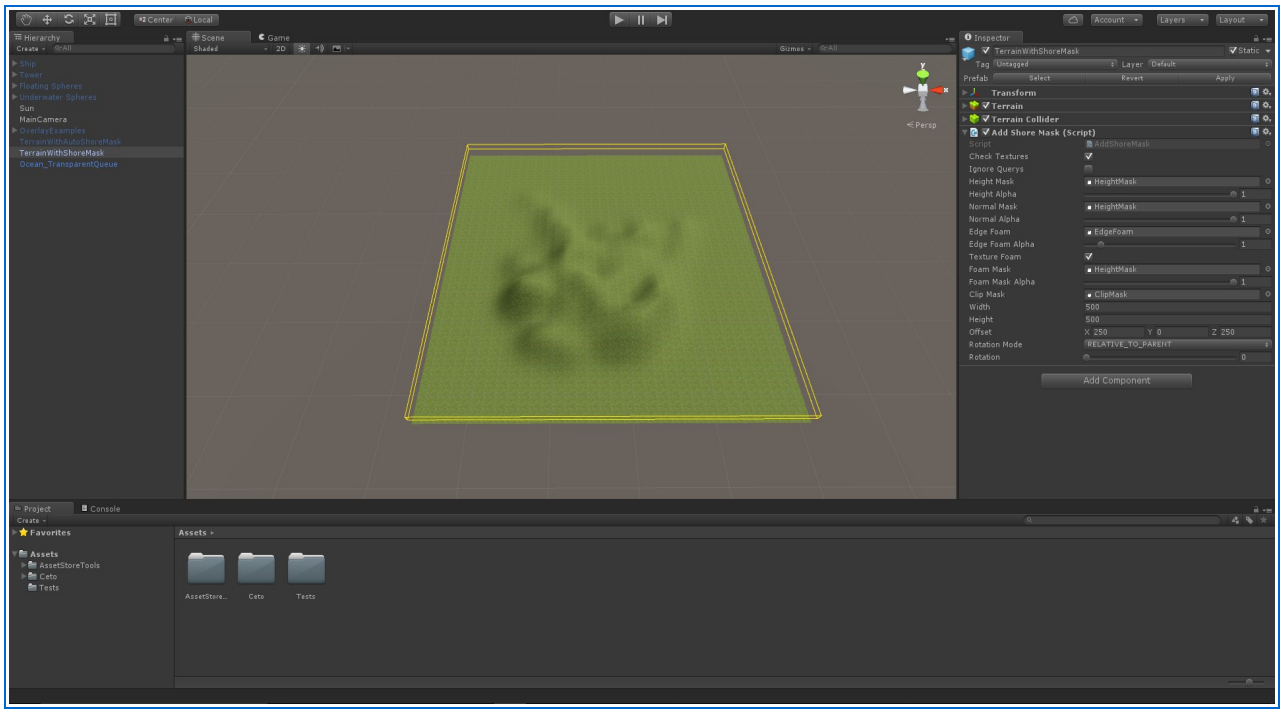
First you will need a set of mask textures. Follow the tutorial on how to export the mask textures for you terrain. There are five texture slots (height mask, foam mask, edge foam, normal mask and clip mask) but each is optional and I often reuse the height mask for the foam and normal mask slots.

Step 2.

Click on your terrain and add the Ceto/Overlays/AddShoreMask component. This script uses Cetos overlay system to modify the ocean surface.

Step 3.

Set the width and length to match your terrains size and then set the x offset to half your terrains width and the z offset to half your terrains length. In the editor window there should be a yellow box drawn around the terrain. It should match the terrains size. The height of the box is not important and will be a constant value.



Step 4.

Drag each of the mask textures onto the slots. The height mask will mask out the wave heights from the wave spectrum where its white. This allows the waves to fade in near the shore. The same applies to the normal and foam masks. The edge foam texture will add foam to the ocean where it is white. This should give a rim of foam around the terrain. The clip mask will cut out the ocean under the terrain. You may need to modify the clip mask in Photoshop so areas inland that are under the sea level clip the ocean away.

Step 5.

Each mask has a alpha slider. Adjust this to control the strength of the effect. For HDR set-ups the edge foam alpha may need to be increased to make it more visible.

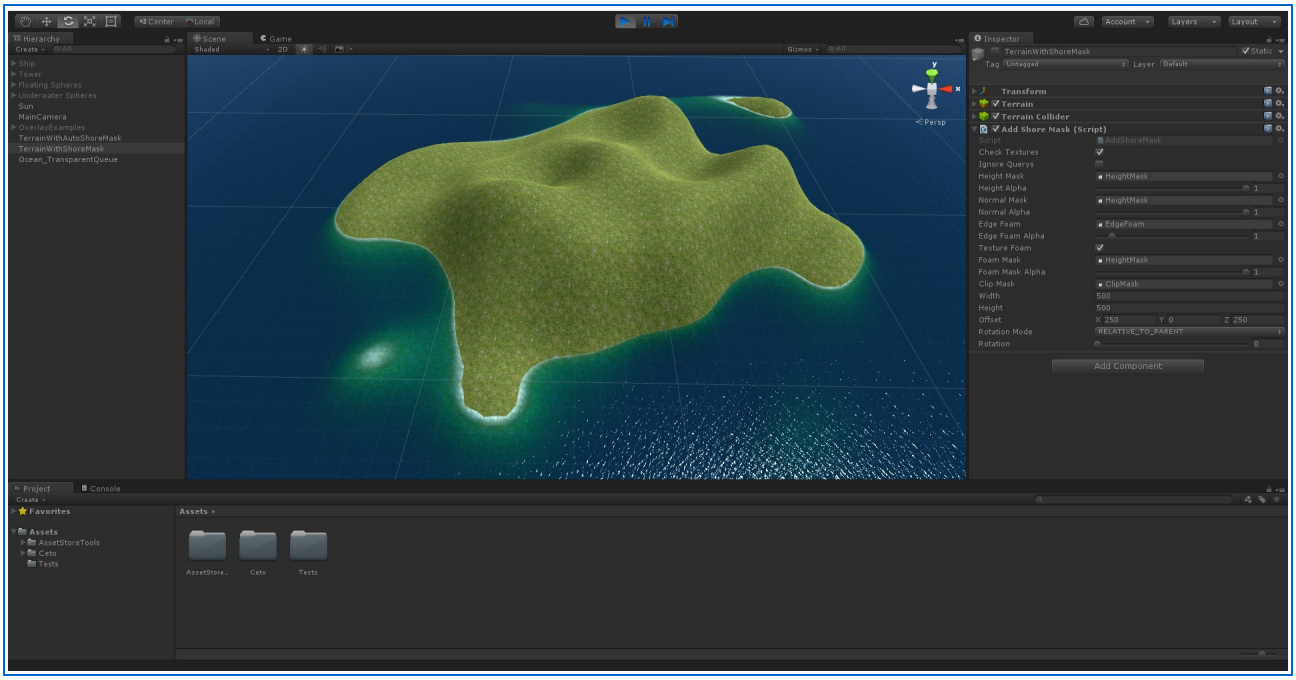
Using the AddAutoShoreMask component.

Step 1.

Click on you terrain and add the Ceto/Overlays/ AddAutoShoreMask component. This script uses Cetos overlay system to modify the ocean surface.

Step 2.

This effect is only created on start up so run the scene and you should see the shoreline effect on your terrain.



Step 3.

Untick any masks that you don’t want applied.

Step 4.

Adjust the spread value to increase the effect around the shoreline. The mask value is calculated from the terrains height so in areas where the terrain is steep the spread wont be able to increase the effect by much.

Step 5.

Each mask has a alpha slider. Adjust this to control the strength of the effect. For HDR set-ups the edge foam alpha may need to be increased to make it more visible.

How to adjust the underwater effect

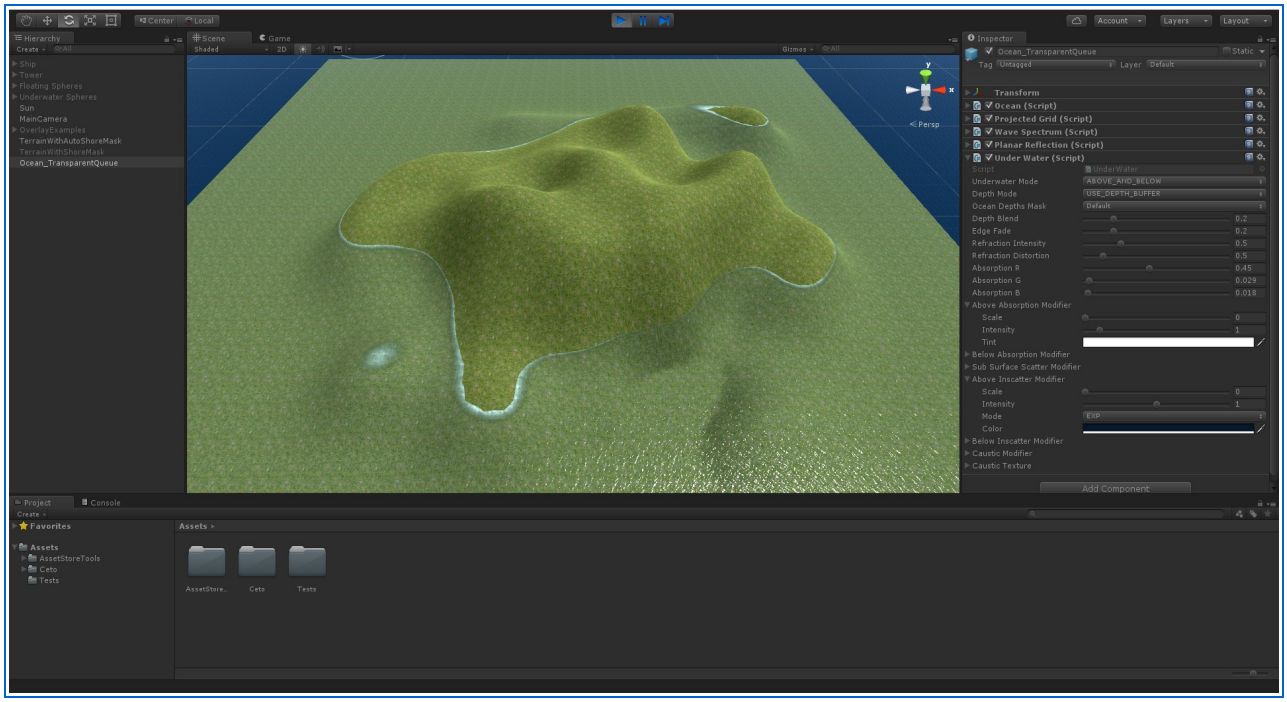
Getting the underwater effect can be tricky because of the number of settings involved. This tutorial should help clear up how to use them and what needs to be adjusted. It should also be mentioned that the underwater effect referrers to two different locations. One is when the camera is above the ocean looking down on the water surface. This is referred to as the 'above underwater effect'. The other is when the camera is below the waters surface. This is referred to as the 'below underwater effect'. This section just deals with the above effect but the same theory applies when adjusting the below effect. All settings referred to can be found on the Underwater component on the ocean.

Step 1.

First run the scene and position the camera above the terrain looking down to get the best view.

Step 2.

It helps to start of with the underwater effect set so its completely clear. Open up the 'Above Absorption Modifier' and the 'Above Inscatter Modifier'. Set the scale for both to 0. The ocean should now be clear like in the image below.



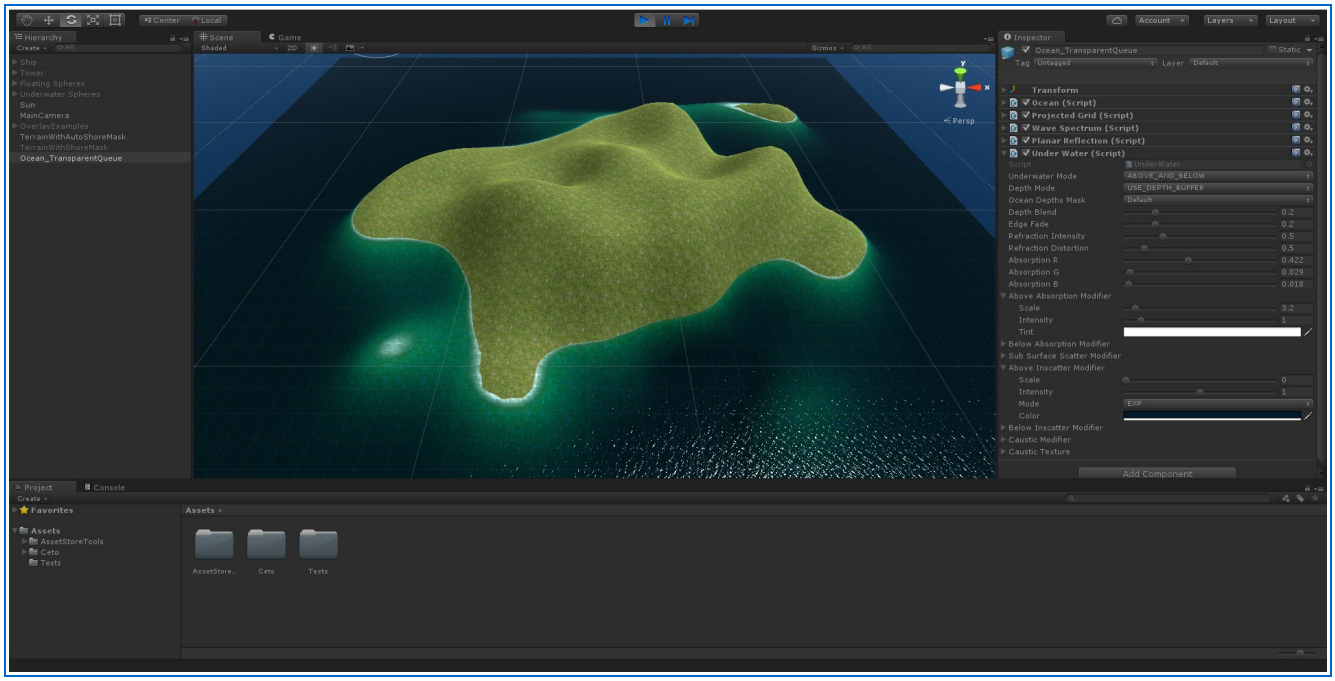
Step 3.

It helps to first adjust the absorption settings. This controls how light changes colour as it passes through the water. Slowly increase the absorption scale until the colour of the water around the shallow areas of the terrain look how you want. The deep areas of the water will be black. This is expected and don't worry about that at the moment.

The absorption R, G and B sliders control how each wave length of light changes as it moves through the water. The default settings are the real world absorption values of water. Notice the R channel is the highest. This is what makes water look blue/green as the red light gets absorption the fastest leaving just the blue and green light. You can tweak these sliders to change the colour of the water and you can even look up real world values for water like tannin or algae rich water.

You can use the tint colour to slightly tweak the colour of the absorption result as it can be hard to get the colour right from just the absorption RGB sliders but I prefer to leave the tint as white. For HDR set-ups it might be needed to adjust the intensity.

The water should look something like the image below.



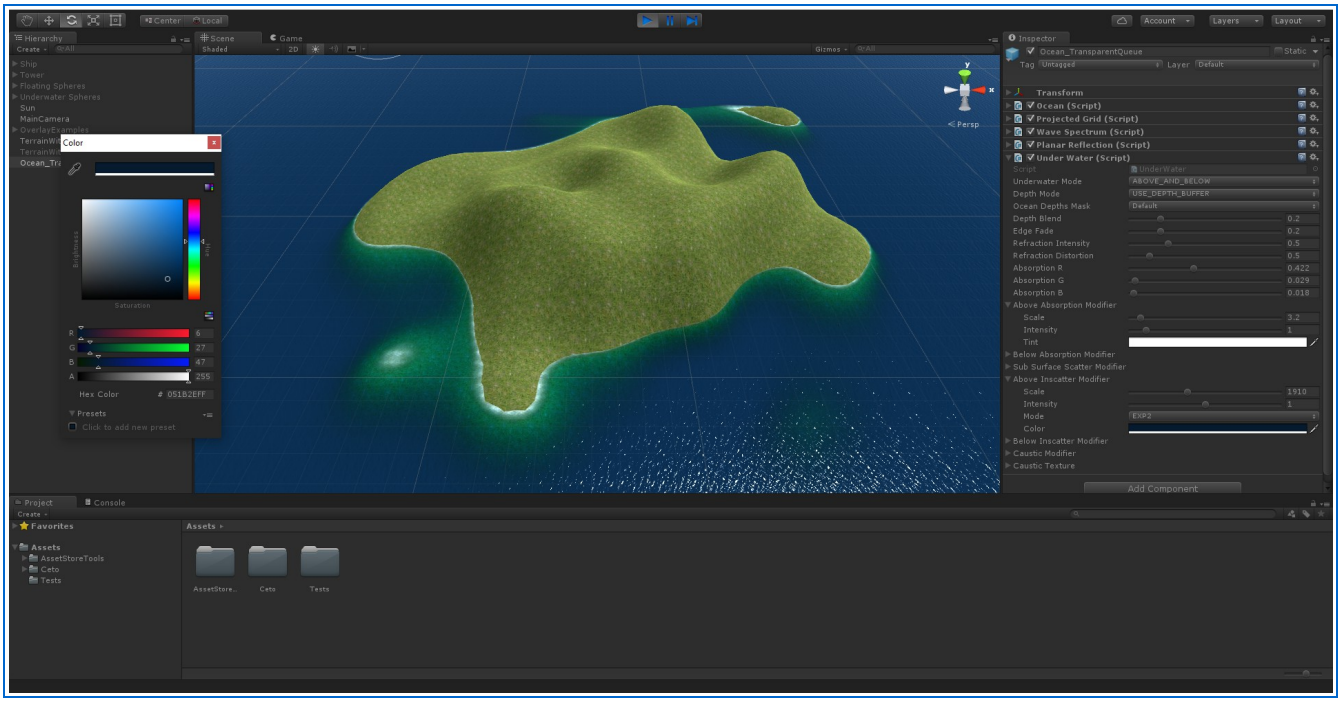
Step 4.

Next the inscatter needs to be added. In the real world the water does not go black the deeper it is because small particles reflect light back to the viewer making the water look 'foggier' until you just see a solid blue colour that is the result of inscaterred light. Inscatter mean light that has been scattered from the water into the view path of the camera.

Real inscatter is too complicated to calculate in real time so Ceto just uses a basic fog effect to approximate it.

Slowly adjust the inscatter scale until the fog covers the black areas of the water. You can change the mode for different fall off types. For example EXP2 will be quite clear near the surface but fall off quickly in the depths while EXP and LINEAR will be less clear near the surface and the fall off is more gradual. If changing modes then the scale value used needs to be much higher for EXP2 and much less for EXP and LINEAR.

Adjust the colour value to change the colour of the fog. The water should look something like the image below.



How to add per-camera settings for the ocean

Ceto will automatically render the ocean for each camera in the scene that views the water culling layer. For secondary cameras you may not want the ocean to be as detailed to help save some frames. There is a script that can be added to secondary cameras to adjust some of the ocean settings.

Its recommended to just use this script for secondary cameras and leave the main camera to use the default setting of the ocean.

Step 1.

Click on the camera and add the Ceto/Camera/OceanCameraSettings component.

Step 2.

Select the options for that camera. Reducing what is drawn in the reflection and ocean depth mask will have the biggest effect on the frame rate. Reducing the overlay buffer sizes can help reduce memory use.

The ocean depth mask is only used if the underwater mode is set to USE_OCEAN_DEPTH_PASS.

Currently all cameras will reuse the same ocean mesh so you cant adjust the mesh resolution per camera.

How to find the height range of the waves

Ceto needs to know the height range of the waves for certain calculations. You can retrieve this information if you also need it. The follow code will find the max and min range of the waves.

```
bool includeOverlays = false;
float maxWaveHeight = Ocean.Instance.FindMaxDisplacement(includeOverlays);

//The min max range in world space.
float minRange = Ocean.Instance.level - maxWaveHeight;
float maxRange = Ocean.Instance.level + maxWaveHeight;
```

The include overlay option allows you to specify if you want the range to include any height added to the ocean by the overlays. Since the overlays only modify the ocean in local areas you most likely don't want this included in the range.

The max range is only the range that's possible in theory. In practice the waves will only reach about 70% of this value most of the time.

The wave range can only be calculated if the wave spectrum performs the FFT on the CPU. If the FFT is done on the GPU then Ceto can not access the wave heights to find the range so a constant value is used which will be the max possible displacement at the highest wind speed. This range will be much larger than the waves in most cases.

How to adjust the ocean conditions from a script

If you want to script changing conditions to the ocean this how to access ocean settings and what needs to change. There are many settings that you might want to change as the weather changes but most of them will be on the ocean or wave spectrum component.

You will need at least Ceto version 1.1.2 for the follow code to work and presumes your using the Ceto namespace.

```
Ocean ocean = Ocean.Instance;
WaveSpectrum spectrum = ocean.Spectrum;

ocean.windDir = yourWindDir;
ocean.foamIntensity = yourFoamIntensity;
spectrum.windSpeed = yourWindSpeed;
spectrum.choppyness = yourChoppyness;
spectrum.foamAmount = yourFoamAmount;
spectrum.foamCoverage = yourFoamCoverage;
```

The spectrum wind speed controls the size of the waves generated. Larger waves will create more foam but you might want to also adjust the foam amount and coverage based on the weather.

As the waves become larger a high choppyness value may cause issues. You might want to script it to decrease a bit as the wind speed increases.

At night the foam can appear to be very bright. You may want to script the foam intensity to decrease during night time.

You can access the other components to change their settings in the same way.

```
ProjectedGrid grid = ocean.Grid;
UnderWater underwater = ocean.UnderWater;
PlanarReflection reflection = ocean.Reflection;
```

It would also be good practice to check if the ocean instance or components are null before using them. The ocean instance will be null if the ocean game object is disabled and the components will be null if they are not enabled or missing from the ocean game object.

When certain settings change they may cause the spectrum conditions to be recreated. This can take a long time but is performed on another thread so you might not notice it. See the tutorial on spectrum caching for more information on what causes the conditions to be created and how to reduce its impact.

How to cache spectrum conditions

The wave spectrum uses FFT to produce the waves from a initial spectrum based of 5 settings. The Fourier size, the wind direction, the wind speed, the wave age and the spectrum type. If any of these change then the spectrum needs to be recreated. This can take a long time. Around 80ms at a Fourier size of 64 and 1000ms at a Fourier size of 512. This is done on another thread so it wont impact on your frame rate but there may still be a delay between when the setting changes and when the waves actually change.

To help decrease this delay Ceto uses caching to store any previously created spectrum's. When the settings change it will first look to see if there is already one created. If there is then it will just use that one. If not then it will create one and add it to the cache. The cache has limited capacity. If its full then the oldest condition in the cache will be removed before adding a new one.

It is possible to tell Ceto to create and cache some conditions on start up. This way the conditions are already in the cache when the settings change.

You will need at least Ceto version 1.1.2 for the follow code to work and presumes your using the Ceto namespace.

First you will need to grab the spectrum from your script. Make sure you do this in a start function as any conditions you tell Ceto to cache it will create them immediately. This may take a while if your caching a lot.

```
WaveSpectrum spectrum = Ocean.Instance.Spectrum;
```

Next you might want to adjust the cache size. Ceto will only store a certain amount of conditions. If the cache capacity is exceeded while manually caching conditions then you will get warning that the cache is full and the condition will not be added. The default value is 10.

```
spectrum.MaxConditionCacheSize = 50;
```

Next tell Ceto to create and cache the condition. The settings are the Fourier size, the wind speed, the wind direction and the wave age in that order.

```
spectrum.CreateAndCacheCondition(64, 3.0f, 0.0f, 0.64f);
```

To improve the likely hood that a condition will be in the cache when a setting is changed during runtime you might want to limit what settings you change. I would leave the Fourier size and the wave age as constants during run time and just change the wind speed and direction if possible.

Also you might want to increment the wind speed and direction in fixed amounts. For example in units of 1 or 2. If you increment using the delta time or some other arbitrary value then the chance that a previous condition is in the cache would be very low.

Remember that during run time Ceto will be adding any new conditions to the cache so if your changing the settings a lot by arbitrary amounts then its possible for the cache to fill and the conditions you cached will be pushed out to make way for the new ones.

When any of these settings change during run time and it causes Ceto to start creating a new condition it will ignore any new changes to the settings until the condition is finished. For example if you change the wind speed and it triggers a new condition to be created it maybe be 10 frames latter that the condition is done. You might want to check if the conditions is done before changing any settings again.

```
if(!spectrum.IsCreatingNewCondition) {
        spectrum.windSpeed = yourWindSpeed;
}
```

How to use asynchronous wave query tasks

Ceto can query information about the ocean surface very quickly. Its likely you can query the ocean hundreds of times a frame with out too much of a problem. In some cases however you may want to optimize this further by using Cetos asynchronous query tasks. There are two types of tasks. A threaded task that will run on a separate thread or a coroutine task that will run asynchronous on the main thread.

The threaded task will be faster but has some limitations. It wont sample the overlays (ie used for the shoreline masks) as the overlays are stored in Unitys texture objects and dont support access other than from the main thread. If your game is mostly on the open ocean or you dont use the overlays then it may still be useful to use this task.

The coroutine task runs on the main thread so it does support sampling the overlays.

The tasks take a array of wave query objects and a callback function. When the task is finished the call back will be called with the query results.

You need to make sure you have a reasonable number of queries per task. Too many and the task will take a long time before the callback is called and to little and the overhead of managing the thread wont be worth the benefits. The exact number depends on your application but I would say around 100 per task is reasonable.

You will need at least Ceto version 1.1.2 for the follow code to work and presumes your using the Ceto namespace.

First you need to make a array of querys and set their positions.

```
int size = 100;

WaveQuery[] querys = new WaveQuery[size];

for (int i = 0; i < size; i++)
{
	querys[i] = new WaveQuery(yourPosition);

	querys[i].tag = i;
}
```

The query object has a tag setting. It might be helpful to store the querys index for latter.

You will also need to have a call back function that takes a enumeration of querys.

```
void CallBack(IEnumerable<WaveQuery> querys)
{
}
```

Next you need to create your task. This would be for a threaded task.

```
WaveQueryTask task = new ThreadedWaveQueryTask(querys, CallBack);
```

And the following for a coroutine task

```
int querysPerIteration = 16;
WaveQueryTask task = new CoroutineWaveQueryTask(querys, CallBack, querysPerIteration);
```

For the coroutine task you can also tell it how many querys to run per iteration. For example if the query array has a length of 64 then each iteration of the coroutine will run 16 tasks and there fore finish after 4 frames.

Next you must add the task to the ocean. The ocean scheduler will then run the task. Since you will mostly likely be re-running the task once its done in a update function its important that you dont add the task to the ocean again before the task has finished from the last time it was added. The ocean will not run it if its not done.

```
        void Update()
        {
                if (!task.IsScheduled || task.Done)
                {
                        foreach (WaveQuery query in task.Querys)
                        {
                                query.posX = yourNewPosition.x;
                                query.posZ = yourNewPosition.z;
                        }

                        Ocean.Instance.QueryWavesWithAsyncTask(task);
                }
        }
```

The task has a IsScheduled and a Done property. If the task is not scheduled or is done then its safe to send it to the ocean to rerun. You will also want to update the query positions before re-running it.

Both the threaded and coroutine task inherit from a base class so both are added to the ocean the same way and you could create your own query task if you want as long as it inherits from the same base class.

Once the task is done the call back will be called with the querys. The follow would use the query tag so you know which position the query is for but you can use other methods is your prefer like using the query as a key in a dictionary.

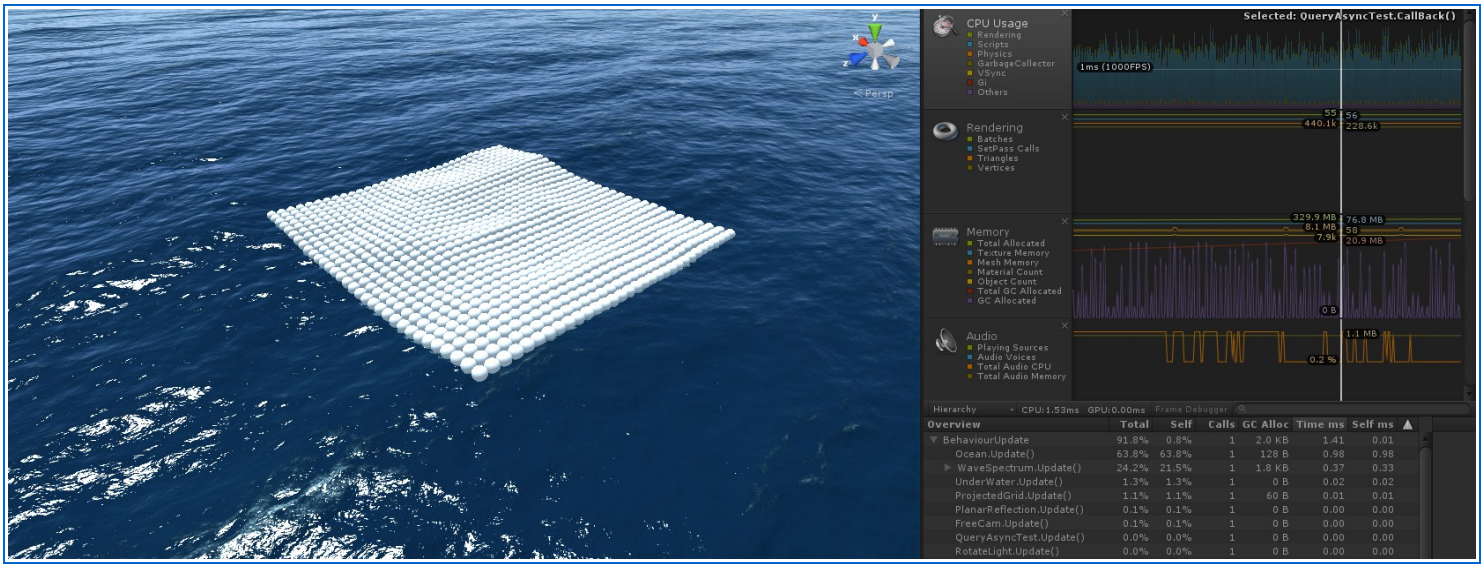
```
        void CallBack(IEnumerable<WaveQuery> querys)
        {
                foreach (WaveQuery query in querys)
                {
                    float height = query.result.height;
                    int index = query.tag;

                    yourPositions[index].y = height;
                }
        }
```

It maybe hard to see from that example how exactly to use the task so I have included a full script that creates a array of spheres and then uses a threaded task to set there y position to match the ocean surface.

```
using UnityEngine;
using System.Collections.Generic;

using Ceto;

public class QueryAsyncTest : MonoBehaviour
{

    public int sphereGridSize = 8;

    GameObject[] spheres;

    WaveQueryTask task;

    void Start()
    {
        spheres = new GameObject[sphereGridSize * sphereGridSize];

        WaveQuery[] querys = new WaveQuery[sphereGridSize * sphereGridSize];

        for(int x = 0; x < sphereGridSize; x++)
        {
            for (int z = 0; z < sphereGridSize; z++)
            {
                GameObject sphere = GameObject.CreatePrimitive(PrimitiveType.Sphere);
                sphere.transform.parent = transform;
                sphere.GetComponent<Collider>().enabled = false;

                Vector3 pos = transform.position;

                pos.x += -sphereGridSize / 2 + x;
                pos.z += -sphereGridSize / 2 + z;

                sphere.transform.position = pos;

                int index = x + z * sphereGridSize;

                spheres[index] = sphere;

                querys[index] = new WaveQuery(sphere.transform.position);

                querys[index].tag = index;
            }
        }

        task = new ThreadedWaveQueryTask(querys, CallBack);

    }

    void Update()
    {

        if (!task.IsScheduled || task.Done)
        {
            foreach (WaveQuery query in task.Querys)
            {
                int index = query.tag;
                Vector3 pos = spheres[index].transform.position;
```

```
                query.posX = pos.x;
                query.posZ = pos.z;
            }

            Ocean.Instance.QueryWavesWithAsyncTask(task);
        }
    }

    void CallBack(IEnumerable<WaveQuery> querys)
    {
        foreach (WaveQuery query in querys)
        {
            float height = query.result.height;

            int index = query.tag;

            Vector3 pos = spheres[index].transform.position;

            spheres[index].transform.position = new Vector3(pos.x, height, pos.z);
        }
    }

}
```

This is a screen shot of the script running with the sphere grid size set to 32 so there will be 1024 spheres and there fore 1024 querys in the task. Its a extreme number but it shows that the cost of the querys dont show in the profiler and the scene runs at 500fps on my PC. The cost you do see from the ocean (0.98ms) is the cost of the call back to update the positions of all those objects.

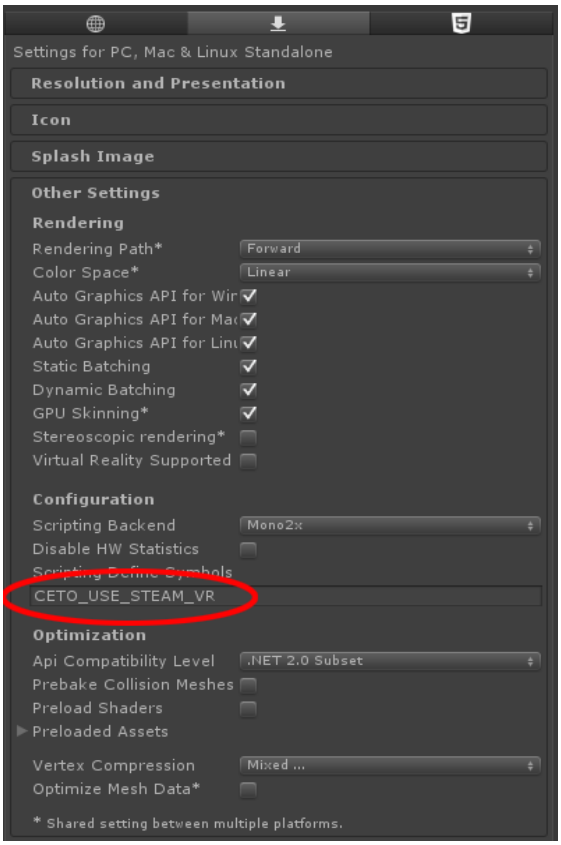


How to set Ceto up for SteamVR

To use SteamVR you will need to tell Ceto to use SteamVR by defining a preprocessor. A preprocessor is used so there will not be a error for people who dont have SteamVR in their project. You will need at least Ceto version 1.1.2 for the following to work.

Go to Edit->Project Setting->Player. Then in the inspector click on other settings.

You then need to type CETO_USE_STEAM_VR in the scripting define symbols box and hit enter. When running the scene you should see a info message from Ceto stating that it is using SteamVR.



It is possible to use SteamVR with Ceto without defining this preprocessor. If not defined Ceto will behave normally and before Unity 5.4 this worked fine. From Unity 5.4 new features have been added to SteamVR that allow for more efficient rendering by only using one camera. The issue is that certain effects done by Ceto (like planar reflections) need to be rendered per eye. By only using one camera Ceto now has to directly interact with the StreamVR API to retrieve the information it needs to render these effects correctly for each eye.

This mean that in Unity versions before 5.4 defining this preprocessor will not cause Ceto to behave any different but it is still recommended to define it in case that changes in future Ceto versions.

From Unity version 5.4 you must use the USE_OCEAN_DEPTH_PASS as the depth mode on the underwater component with SteamVR. The new changes mean that the USE_DEPTH_BUFFER option is not correct for each eye.

The large degree of freedom of the camera in VR can cause issues with the projected grid causing it to wiggle and sometime breaks especially when looking down. If you are having this issue click on the ocean component and untick the 'Tight Projection Fit' option.

See the common problems section on hoUnity 5.4 upgrade notes.w to add support for single pass rendering in Unity 5.5+.

6. Common Problems

Setting up a new asset in your project can be challenging and Ceto is no exception. Although I have done my best to make sure Ceto works with the minimal amount of set up there can still be issues. The follow is a list of the most common problems that have come up and how to solve them if possible.

If you feel Ceto is not behaving as it should then feel free to send me a email at the support address on the asset store or post a question on the Ceto thread on the Unity forum.

Remember to stay up to date with the latest Ceto version.

Where are my reflections?

If you do not have any reflections showing on the ocean or an object is missing from the reflections then these are the things you should check. All these settings can be found on the Planar Reflection component on the ocean unless otherwise stated.

- By default the reflections in Ceto are blurred. It maybe that the blur is obscuring the object in the reflection. Set the blur mode to off or lower the blur iterations and see if the object is there.
- If the sky box is missing from the reflections then make sure that the 'Reflect Sky Box' option is ticked.
- If an object or the sky is missing from the reflections then it maybe not selected in the reflection mask. On the reflection mask have a look to see if the object layer is selected. The best way to check this is to select everything and see if the object appears. If it does then just deselect any layers not needed.
- If you can see the reflections but they are very faint on close up objects then its got to do with the Fresnel. The Fresnel naturally makes the reflections on the ocean close to the camera fainter than further away. You can adjust the min Fresnel and the Fresnel power on the ocean component.

Why does this object not have the underwater effect applied to it?

If an object in your scene looks like the underwater effect in not being applied to it correctly then these are the things you should check. All these settings can be found on the Underwater component on the ocean unless over wise stated.

- If you are using the USE_OCEAN_DEPTH_PASS depth mode then the object layer must be selected in the ocean depth mask. The best way to check this is to select everything and see if the object appears. If it does then just deselect any layers not needed.
- If you are using the USE_OCEAN_DEPTH_PASS depth mode and your object has a custom vertex displacement function then its render type needs to be added to Cetos OceanDepths replacement shader. You may need to contact me about how to do this. This may not be possible for some complex tessellated shaders.

Make sure the TransparentFX layer is not selected as they will not show up correctly.

Why is this wave overlay not working?

If you have created a texture and added it to a wave overlay and it does not appear to be working then these are the things you should check.

- The overlays sample from the textures alpha channel. You can fill the textures alpha channel by making a grey scale image in Photoshop and then in Unity click on the texture and go into the advanced settings. Then select the 'alpha from grey scale' option.
- If you have manually added the mask to the alpha channel within Photoshop then make sure that the 'alpha from grey scale' is not ticked as Unity will replace the alpha channel with the grey scale value.
- Make sure the mask mode is set correctly. It should be set to WAVES except for the normal mask which should be WAVES_AND_OVERLAY_BLEND. If the mode is WAVES_AND_OVERLAY_BLEND then a texture is needed for both the texture slot and the mask slot.
- If the overlay modifies the wave heights and it looks like Cetos wave queries are not taking it into account when sampling the wave height then you need to click on the texture and go to the advanced settings. You need to make sure the 'Read/Write' option is ticked. You also need to check that 'Ignore Queries' is not ticked on the overlay script if it exposes that setting (not all overlay scripts do).

Why cant I get global fog to be applied to the ocean?

Images effects like global fog and many sky assets atmospheric scattering are applied after the opaque objects are rendered but before the transparent ones. If you want these effects on your ocean you need to use the opaque ocean prefab which will run in the opaque render queue. If you use the transparent prefab the ocean will get rendered after the image effect has run and will there fore nor be applied to the ocean.

Alternatively you could shift when the effect gets applied to after the transparent queue. This means removing the 'ImageEffectOpaque' tag above the OnRenderImage function in the post effect script. This may have side effects and will not work in all cases. The effect will also be incorrectly applied to other transparent objects.

The correct way to solve this issue is to add a final function in the ocean shader that manually adds the effect to the ocean after the oceans final colour has been resolved. This obviously requires you know how to code both the ocean and sky shader which is not practical for most people and as the ocean and sky assets are made by different publishers there is no guaranteed compatibility between them.

Why cant I get shadows to show on the ocean?

Shadows are only supported on opaque objects so will only be on the ocean if you are using the opaque prefab.

You also need to click on the projected grid component on the ocean and make sure 'Receive shadows' is ticked.

Why cant I get point and spot lights to show on the ocean?

Cetos shader will always run in forward rendering. The shaders are quite expensive to run so multiple light sources are disabled by default. You will just have the main directional light on the ocean. To enable other light sources like point and spot lights you need to go into the OceanTopSide_Transparent and OceanTopSide_Opaque shader and remove the noforwardadd tag

```
        #pragma surface OceanSurfTop OceanBRDF noforwardadd nolightmap keepalpha
```

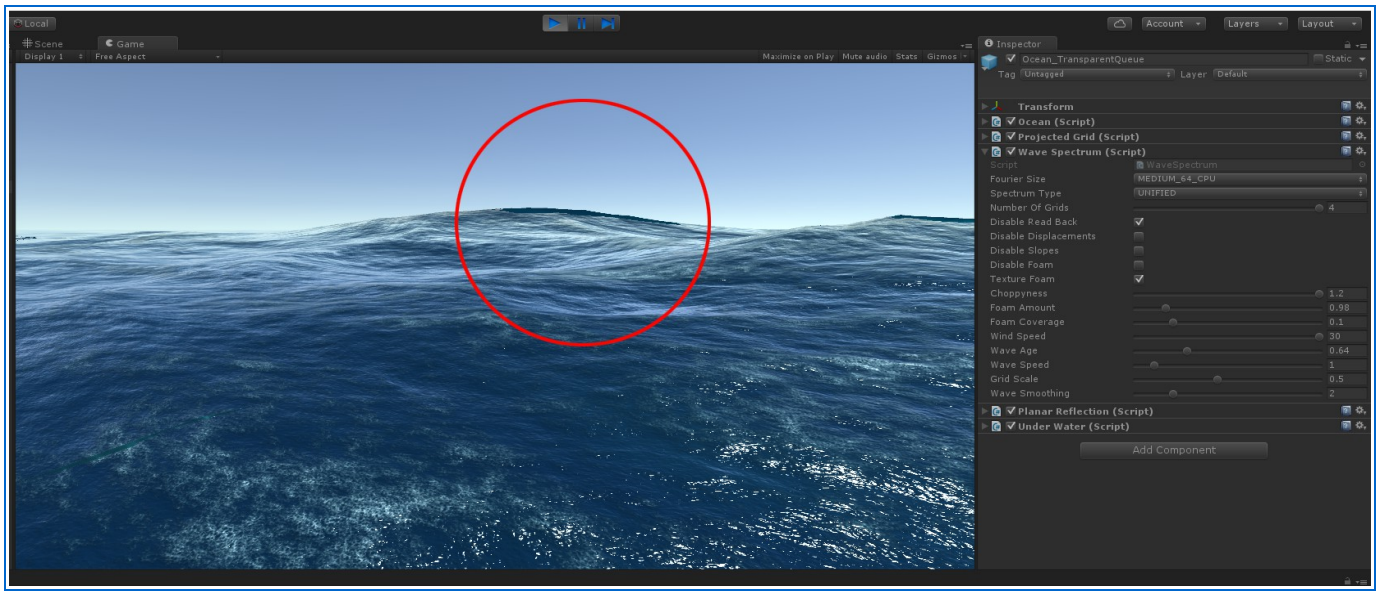
Why is there a strange artefact on the ocean behind a Speed Tree?

If the depth mode on the underwater component is set to USE_OCEAN_DEPTH_PASS then the depth information used to apply the underwater effect is generated using a replacement shader. This is not currently support for Speed Trees or transparent objects.

The layer these objects are on should not be in the ocean depths mask setting which can be found on the underwater component. You need to remove them from the mask. This also means that these objects should never be placed under the water as the depth effect will not be applied to them correctly.

Why is there a strange artefact on the tips of the waves?

On the wave spectrum component there is a choppyness setting. This increases the amount of displacement applied to the waves on the x/z axis and makes the waves have a sharper peak. The issue is that if this is increased too much then the tips of the waves can curl back on themselves like in the image below.



The artefact is the underside mesh so if the underwater mode is set to ABOVE_ONLY there wont be a underside mesh and this area will be clear instead.

This will mainly occur as the wind speed increases as the larger waves created also have a higher x/z displacement amount. You may need to lower the choppyness as the waves get larger to prevent this.

You may also see a similar issue around the edges of the screen which is also caused by the choppyness being to high when the wind speed is increased.

Why is my object not floating on the waves?

It is possible that the wave queries are not sampling the wave height so any objects with buoyancy wont be affected by the waves and will just sit at the ocean level. If the wave spectrum Fourier size has a GPU at the end then the wave data is in the GPU memory and not accessible by Ceto. You need to use a CPU option instead.

See the use decision section on which Fourier size to used there pros and cons.

Why is a object showing in the reflections when its not seen by the main camera?

There are some instances where a object will be showing up in the reflections but has been culled by the main camera and visa versa. The are two causes for this that I am aware off.

The first occurs if you have manually set the culling distances for the camera. By default the reflection camera will not use the culling distances unless 'Copy Culling Distances' is ticked on the planar reflection script.

The second issue is because the reflection camera is not in the same position as the main camera there maybe a difference between what gets culled by each. The reflection camera also used a oblique project matrix which complicates things. You may have to manually set what the reflection cameras culling distances are for each layer.

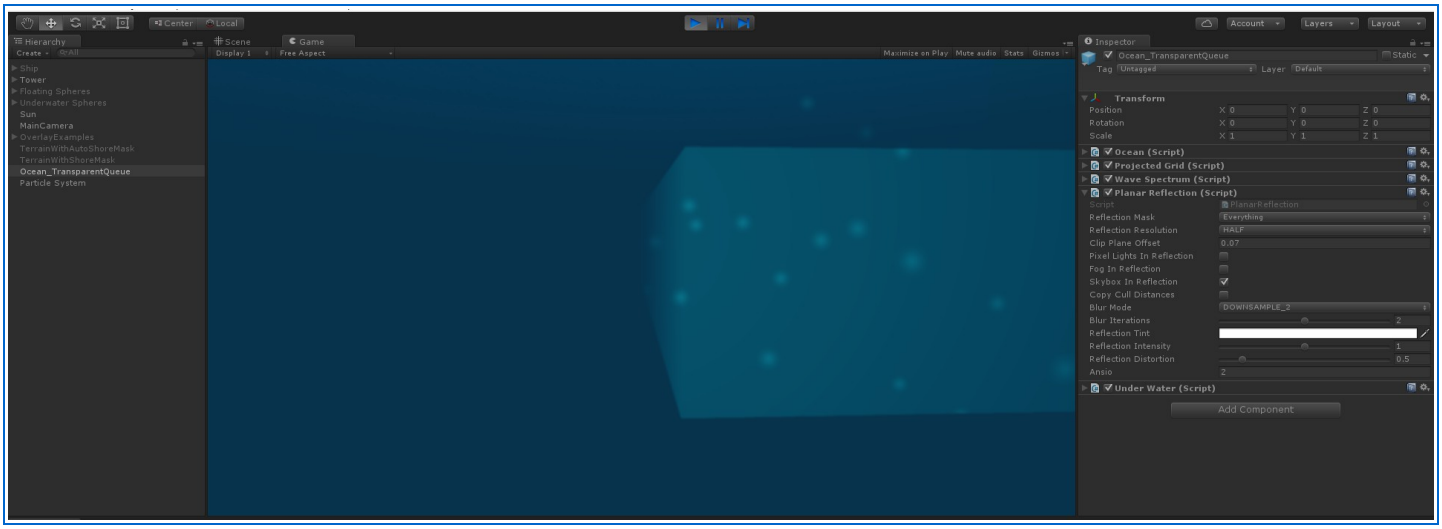
Click on your camera and add the Ceto/Camera/ReflectionCameraCullingDistances component. You can then manually adjust the distances for each layer. If you use this component make sure that the 'Copy Culling Distances' option on the planar reflection component is not enabled as it will cause the distances from the main camera to be copied onto the reflection camera overwriting those you manually set.

Why are my particles not showing correctly underwater?

Cetos underwater post effect camera will apply the effect after transparent objects have been drawn as it needs to be applied after the ocean has been drawn. This means that transparent particles will also have then effect applied to them and it will not be applied correctly.

The issue is that the transparent particles don't write to the depth buffer and Ceto needs the depth value to apply the effect. The shader knows the oceans depth value as it is stored it in a separate texture but there's no way to get the particles depth value.

The issue will appear as the particles only being visible when they pass in front of another object like in the image below.



What's happening is the underwater post effect is using the depth value of what ever is behind the particles. If there is nothing behind them the effect thinks they are very far away and should be covered by the underwater fog.

There is a way to fix this but is requires using a second camera that only renders the particles and nothing else and will only work in froward rendering.

To set this up follow these steps.

Step 1.

Click on the particle system and click on the layer tab. Click 'Add Layer'. Create a new layer called particles and set this as the particles systems layer.

Step 2.

Click on the main camera and on the culling mask deselect the particle layer. You don’t want the particles drawn by the main camera any more.

Step 3.

Add a new game object as a child to you main camera and add a camera component. You need to make sure that the second cameras position always matches the main camera when its moved in the scene. Make sure that what ever camera controller script your using will do this.

Step 4.

Click on the second camera and set its culling mask to nothing and then only select the particle layer. This camera should only draw the particles.

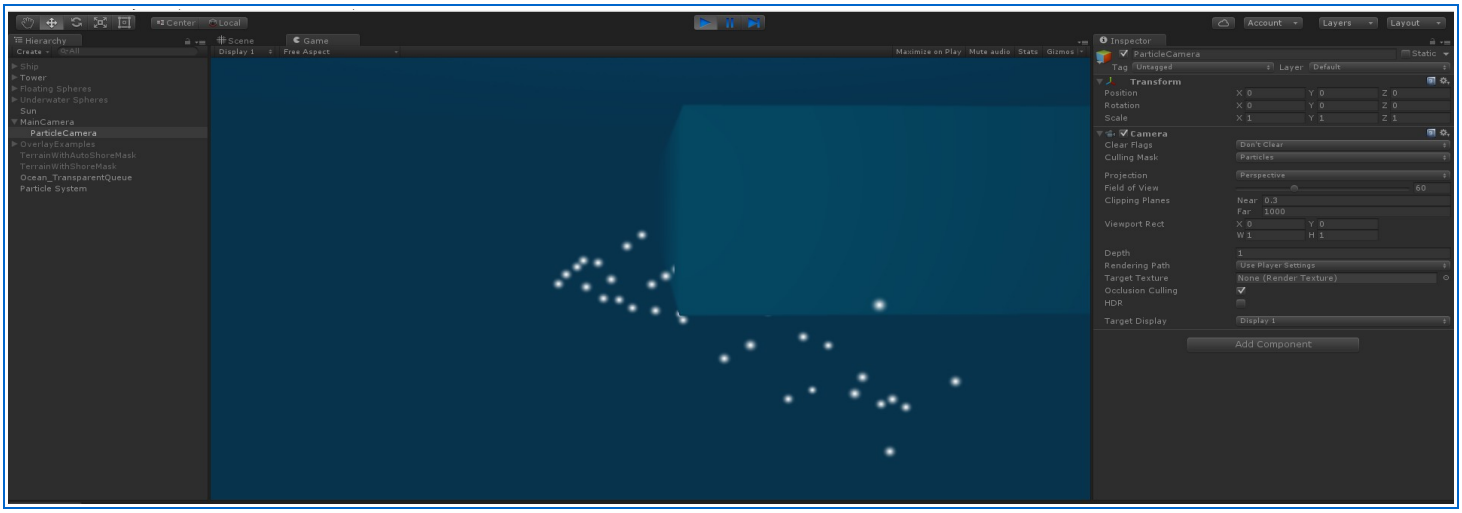
Step 5.

Set the second cameras depth to higher than the main camera. It should be drawn after your main camera does.

Step 6.

Set the second cameras clear flag to 'Don’t Clear'. You want the second camera to add the particles to what is already on the screen with out clearing what is already there.

The particles should now be rendered like they normally do.



Obviously the particles are not going to have Cetos underwater post effect applied to them but its a better result than before.

Why are the caustics not showing up in my scene?

Currently the caustics will only work if the depth mode is set to USE_DEPTH_BUFFER on the underwater component. That also means the caustics will not work with the opaque ocean prefab.

The caustics when using deferred rendering require at least Unity version 5.2.

If you have checked that and they are still not showing make sure the caustic texture has been set. It is set on the Underwater component on the ocean prefab.

I am getting a error after I upgraded to a new Ceto version. How do I fix it?

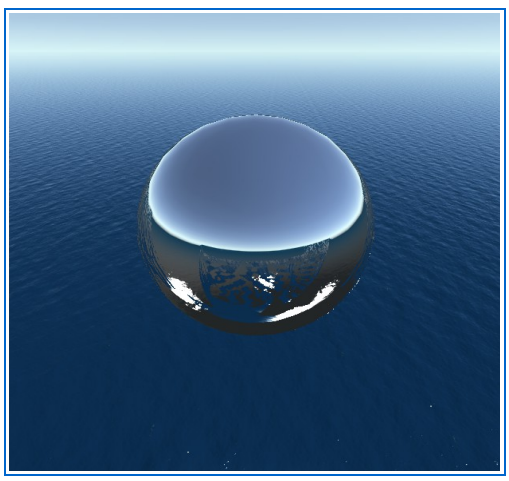
Sometimes after upgrading you may get a error. These are the most common sources of errors and how to fix them.

- If you are getting a error stating that a script is missing or a script is missing a method you need to delete the Ceto folder before importing the new version. Sometimes I need to move scripts to new folders to keep things organised. When Unity imports the new version it will just add any new files and override existing ones but wont delete anything. This means that you can end up with two copies of the same script in different folders if the new version has moved a script and this will cause a error.
- If you are getting a shader error when running the scene or some strange graphical glitch then go to the Ceto/Shaders folder, click on each shader and look on the inspector. If you see one with a error then right click on it and hit reimport. Alternatively just right click on the entire shader folder and hit reimport. Sometimes when Unity imports a asset it causes some shaders to have a error. Reimporting them normally fixes it.
- If you get a null reference error when running the scene and you have created you own ocean prefab then I have probably added a new hidden shader or material to the script and the prefab you created is missing the reference to it. You might need to create a new prefab or delete the component that has the error and add a new one. The new one should now have the reference.

If none of these solutions work for you then you can contact me at the Ceto support email or on the Ceto thread at the unity forums.

Why is the ocean not rendered correctly in reflection probes?

Ceto's projected grid does not render correctly when rendered into a cubemap. This means that it will appear broken in the reflection probes. You should not include the layer the ocean is on in your reflection probe mask. The ocean is on the water layer by default. You can change that in the Ocean script.



Why is the ocean not showing up in Unity's DepthNormal texture?

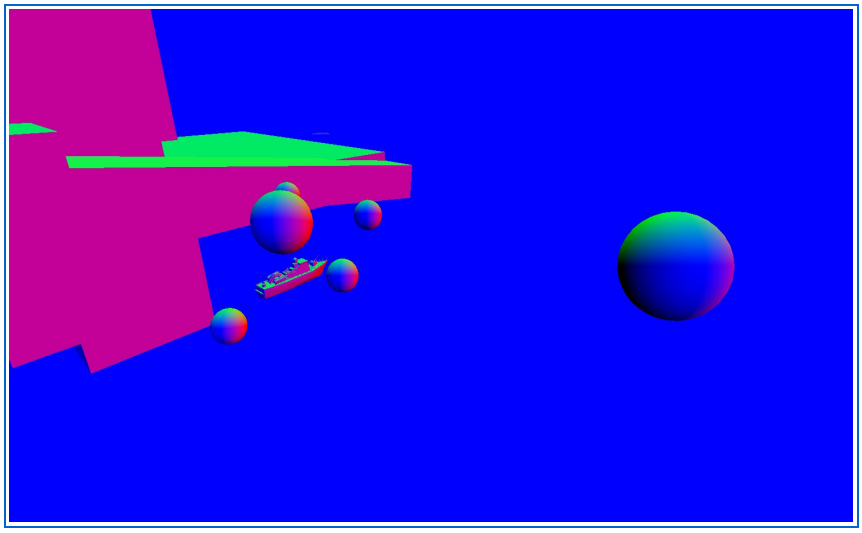
Cetos ocean shader uses a custom render type. This is because the shader has custom vertex displacement for the waves. If I had left the render type as the default then when ever a replacement shader is used the ocean would be drawn with the incorrect vertex function. With a custom render type the ocean will not be drawn by the replacement shader unless its render type is added. Most of the time you will not need to worry about this but there is one instance where this may became a issue.

Unity uses a replacement shader to render forward rendered objects depths and normals into a texture. This texture is uses by many image effects such as ambient occlusion and screen space reflection. If you want the effect applied to the ocean you need add a version of unity default Camera-DepthNormalTexture shader to the asset folder with Cetos render type added to it. These image effects are normally only applied to objects in the opaque queue so you may also need to use the opaque ocean prefab.

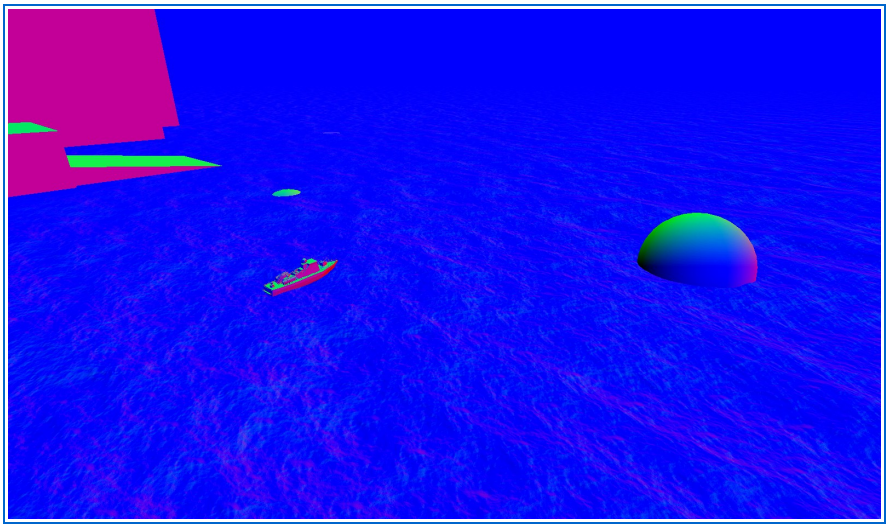
In newer versions of Unity (5.5+) you now have to tell Unity that you are using a depth-normal replacement shader in the graphics settings in the editor.

You can download a copy of the shader here It includes a script to render the DepthNormal texture as the screen color so you can check the shader is working and a small text file with instructions.

Below is a screen shot of the scene without Ceto rendering into the DepthNormal texture.



Below is a screen shot of the scene with Ceto rendering into the DepthNormal texture.



Unity 5.4 upgrade notes.

Unity have changed something about their shaders in the 5.4 that will make the ocean look darker. You can get the ocean looking like it did before by adjusting some of Cetos settings.

Click on the Underwater component

Change the Above refraction intensity from 0.5 to 1.0.

Change the Above Inscatter Modifier color from 5,25,43 to 7,51,77

Why is the ocean rendering over the tree billboards on the terrain?

The render queue of the tree billboard shader is set to draw before the ocean. You can move the ocean shader to render before the billboards. This issue will only occur if your using the transparent prefab as its only then that the ocean gets drawn after the trees.

Open the Ceto/Shader/OceanTopSide_Transparent shader and you will see this in the tag

"Queue"="Transparent-1"

Change it to this.

"Queue"="Transparent-101"

You then need to open the Ceto/Shader/OceanUnderSide_Transparent shader and change its queue to this. The under side should be drawn before the top.

"Queue"="Transparent-102"

The tree billboards have a queue of Transparent-100 so as long as the ocean is before this it will be drawn first. The billboard shader does not write to the depth buffer so the ocean will just draw right over it if its drawn after.

Why does moving the transparent ocean render queue to “ Transparent-500” break the ocean?

Do not move any transparent shader to less than Transparent-499. The transparent queue is at 3000 and Unity's Opaque queue ends at 2500. This means that a queue of Transparent-500 will be 2500 and there for this will push the transparent object into the opaque render queue. Drawing a transparent object in the opaque queue will most likely not work.

The ocean will copy the depth buffer at the end of the transparent queue which is needed for its shader and if its drawn before this happens the ocean will not look correct.

If you are moving the queue to work with TrueSky then just use Cetos Opaque prefab.

Unity 5.5 upgrade notes.

Unity have made some changes to the way the depth buffer value is handled in 5.5. This makes the Ocean go completely transparent when the Cetos Depth mode is set to USE_DEPTH_BUFFER.

In modern graphic API's the depth value is reversed. Unity handles this for you most of the time but there is a function in Ceto where it needs to be manually corrected.

Open the Ceto/Shaders/OceanUnderWater.cginc file. In it you will find this function.

```
/*
* Calculates the world position from the depth buffer value.
*/
float3 WorldPosFromDepth(float2 uv, float depth)
{
     float4 ndc = float4(uv.x * 2.0 - 1.0, uv.y * 2.0 - 1.0, depth * 2.0 - 1.0, 1);

     float4 worldPos = mul(GetIVPMatrix(), ndc);
     worldPos /= worldPos.w;

     return worldPos.xyz;
}
```

You will need to change it to this.

```
float3 WorldPosFromDepth(float2 uv, float depth)
{

     #if defined(UNITY_REVERSED_Z)
        depth = 1.0 - depth;
     #endif

     float4 ndc = float4(uv.x * 2.0 - 1.0, uv.y * 2.0 - 1.0, depth * 2.0 - 1.0, 1);

     float4 worldPos = mul(GetIVPMatrix(), ndc);
     worldPos /= worldPos.w;

    return worldPos.xyz;
}
```

In Unity 5.5.1 you may also see the ocean appear to be much brighter. In version 5.4 unity made some changes to make the ocean appear darker and it appears they may have changed the back in 5.5.1. You will need to tweak the ocean colour again but this time reverse the changes recommended in the Unity 5.4 upgrade notes.

How to add support for single pass rendering in VR to Ceto.

From Unity version 5.5 new shader features have been added to better allow custom shaders to support single pass rendering in VR. With some minor changes you can now get Ceto to run in single pass rendering. There is only one shader that needs to be changed. In the Ceto shaders folder there is a shader called OceanSurfaceShaderBody.cginc.

Open this shader and make the following changes

Input data structs need to have the instance id added.

```
struct Input
{
 float4 wPos;
 float4 screenUV;
 float4 grabUV;
 float4 texUV;
 UNITY_VERTEX_INPUT_INSTANCE_ID
};
```

Vertex shaders then need to have the setup instance and vertex out put initialize. Add these lines below the current UNITY_INITIALIZE_OUTPUT macro.

```
 UNITY_INITIALIZE_OUTPUT(Input, OUT);
 UNITY_SETUP_INSTANCE_ID(v);
 UNITY_INITIALIZE_VERTEX_OUTPUT_STEREO(OUT);
```

For the position in the vert shader you must use UnityObjectToClipPos(v.vertex) instead of mul(UNITY_MATRIX_MVP, v.vertex).

Change the calculation of the screen pos to this

```
float4 screenPos = UnityObjectToClipPos(v.vertex);
```

And finally you need to add a offset to the screen uv depending on the eye instance if single pass is enabled. In the fragment shader for the top side and under side add this under where the screen uv is set.

```
 float4 screenUV;
 screenUV.xy = IN.screenUV.xy / IN.screenUV.w;
 screenUV.zw = IN.grabUV.xy / IN.grabUV.w;

#if UNITY_SINGLE_PASS_STEREO
 // If Single-Pass Stereo mode is active, transform the
 // coordinates to get the correct output UV for the current eye.
 float4 scaleOffset = unity_StereoScaleOffset[unity_StereoEyeIndex];

 screenUV.xy = (screenUV.xy - scaleOffset.zw) / scaleOffset.xy;
 screenUV.zw = (screenUV.zw - scaleOffset.zw) / scaleOffset.xy;
#endif
```