Lux Water

Lux Water is a simple yet robust solution to render water surfaces, which is focused on giving you reliable results as far as refraction, reflection and lighting are concerned.

Lux Water Volumes allow you to get pixel accurate and seamless transitions between above and underwater rendering.

Lux Water Projectors allow you to project local foam or normals.

Using sliding Water Volumes you may create huge water surfaces or even infinite oceans.

Compatibility

Lux water has been successfully tested on DX11 and OpenGLCore. Since version 1.05 Metal (Desktop) is fully supported.

Although the water shader itself uses forward rendering Lux Water works best in deferred. Lux Water only supports the built in render pipeline.

SInce version 1.2. Lux Water comes with basic support for VR (Unity 2019.4 and DX11 only).

Table of Content

Getting Started

Simply create a new material and assign the Lux Water shader *(Lux Water/WaterSurface)*.
Create a plane and assign your material.
Tweak the shader properties to your liking.

Please have a look at the provided demos.

Please note: The package ships with a custom “WavingGrass” shader as used by the grass on the terrain. This is a deferred shader to play nicely with underwater post fx.

Forward Rendering

When using forward rendering you have to make sure that your camera renders a depth texture. Do so by adding the *LuxWater_CameraDepthMode* script to your camera. In case you do not use Metal and deferred rendering leave “Grab Depth Texture” unchecked.

Deferred Rendering

If you use deferred rendering and would like to use “Deep water rendering” you have to assign the *LuxWater_DeferredShading* shader:

- Go to *Edit → Project Settings → Graphics*.
- Under the *Built-in shader settings* change *Deferred* to *custom*, then assign the *LuxWater_DeferredShading* shader.

This of course might interfere with other deferred lighting shaders you may already use provided by e.g. ATG, AFS, Lux Plus or Uber. The tweaks however are rather minor, well documented and should be easy to add to the custom deferred lighting shader you are currently using. It is just three lines of code you will have to change. Look for:

```
(add) #pragma multi_compile __ LUXWATER_DEEPWATERLIGHTING
(add) #include "LuxWater_DeferredLibrary.cginc"
(replace) // UnityDeferredCalculateLightParams (i, wpos, uv, light.dir, atten, fadeDist);
LuxWater_DeferredCalculateLightParams (i, wpos, uv, light.dir, atten, fadeDist, dirLightAtten);
```

in the included “LuxWater_DeferredShading” shader.

Orthographic Projection

Since version 1.062 Lux Water offers support for cameras using orthographic projection. Find out more >

Metal and Deferred Rendering

If you use Metal and deferred rendering things unfortunately get a bit more complicated. Find out more >

Fog

Since version 1.02 Lux Water uses custom fog in order to be able to mask it out on underwater surfaces. By default it uses “Exponential Squared” fog. To enable different or custom fog functions you have to edit the “LuxWater_Setup.cginc” file (located in: LuxWater → Shaders → Includes) and comment/uncomment the corresponding defines. Then reimport the “LuxWater CG” and “LuxWater CG Tessellation” shaders.

Enabling exponential squared fog looks like this:

```
// #define FOG_LINEAR
// #define FOG_EXP
#define FOG_EXP2
```

Enabling linear fog instead would looks like this:

```
#define FOG_LINEAR
// #define FOG_EXP
// #define FOG_EXP2
```

Please note: In case you use deferred rendering and apply fog as an image effect you may benefit from water volumes. Find out more >

Azure, Enviro and Aura 2

Lux Water also supports fog rendered by Azure, Enviro or Aura 2. To enable this feature you have to edit the “LuxWater_Setup.cginc” file as well. Please make sure that you uncomment the corresponding `#define` and `#include` and comment all other `#defines`.

Please note: Azure, Enviro or Aura 2 should be installed at their default location. If you have moved them you will have to adjust the path of the `#include`.

Support for “Advanced Deferred Fog” is handled by Azure or Enviro. Accordingly tweaked shaders should ship with these packages or come very soon.

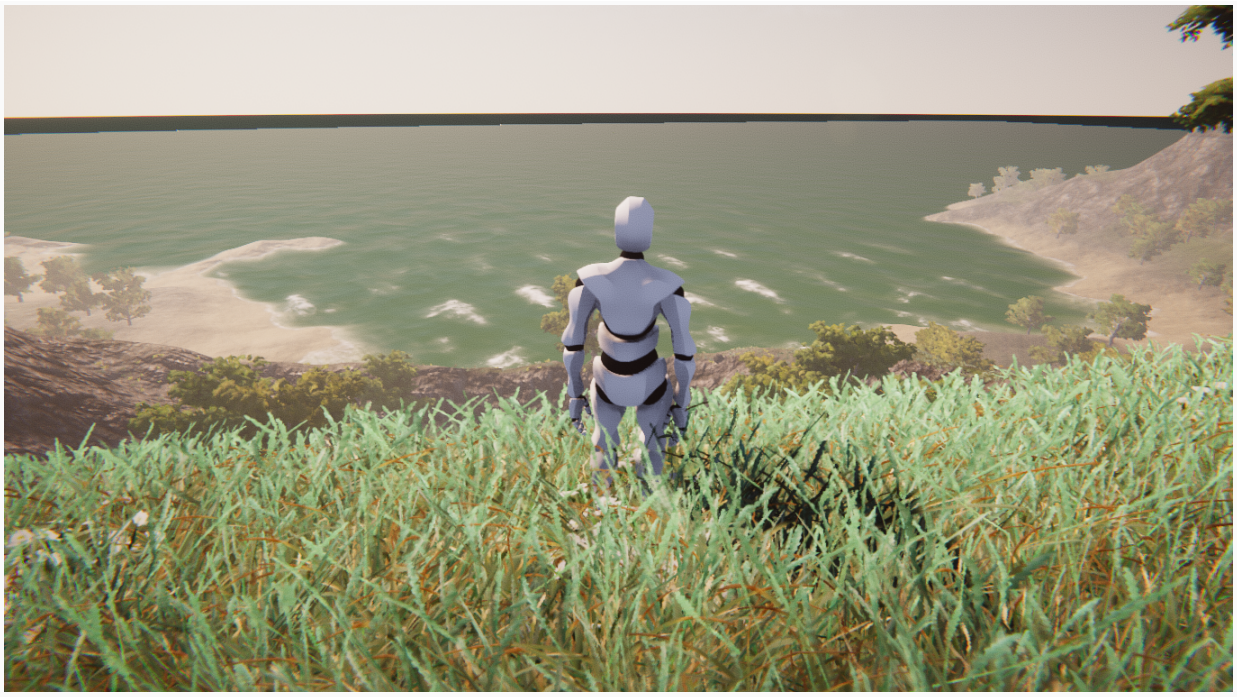
Regarding **Aura 2** the tweaked Aura fog shader is still not included in the official Aura 2 release nor in the default Lux Water package. So you have to contact the author of Auro 2 and me to get it. Contact me: larsbertram69(at)gmail(dot)com.

When using **Enviro** please make sure that it does not render the skybox in “black ground mode” as this will break the reflections. Disable it by editing *Enviro Sky* → *Edit Profile* → *Category: Sky* → uncheck: *Black Ground Mode*.

Also when using **Enviro** you have to take care about its reflection probe whose size is too small in case you create huge water surfaces using several water tiles (size gets set automatically by enviro).

As shown in the image below the skirt geometry does not receive the same reflections as the water tiles in the foreground. Fix this by overwriting the anchor position of the skirt's water tiles in their renderer component.

Gray water surfaces at the horizon are far away water tiles not affected by Enviro’s reflection probe which is too small. *Screenshot by PyroStudio taken from the Unity forum.*



Also make sure that e.g. **enviro renders before lux water when it comes to image effects.**

The Demos

As version 1.07 introduced deep water rendering which may set some global overwrites each demo has a UnderWaterRendering script assigned to the camera which resets these overwrites in case you switch between the scenes and use deferred rendering.

This script usually is only needed in case you actually use water volumes and deep water rendering.

In case you get strange lighting on the water simply activate, then deactivate “Deep Water Lighting”.

Shader Properties

ZWrite Lets you choose whether the shader writes to the depth map or not. Usually it should write to the depth buffer, but when using Metal and deferred shading this will break water. Find out more >

ZTest Lets you choose how the shader should do depth testing. Usually it should be set to “LessEqual”. You may choose “Disabled” as well in case you use forward rendering, MSAA do get artifacts around your objects on top of the water surface, do not displace the water and look from above.

Culling Lets you specify which faces will be drawn:

- **Back** Don’t render polygons facing away from the viewer. Use this in case your camera will never go below the water surface.
- **Front** Don’t render polygons facing towards the viewer. I do not know any use case for this, it simply belongs to the culling settings.
- **Off** Disables culling – all faces are drawn. Use this in case your camera can go underwater

Enable Orthographic Support In case your camera uses an orthographic perspective you have to enable this to make water being rendered properly.

UV Space Texture Mapping If checked texture lookups will be done in uv space – otherwise in world space. Enable UV Space Texture Mapping in order to make water bumps and foam follow the shape and uvs of e.g. a river.

> **UV Direction** Lets you choose between U or V as main axis regarding the texture animation.

Uses Water Volume If checked the shader changes into mode suitable for water surfaces which also get rendered as water volume. Usually it is all handled by script so you do not have to check or uncheck it – but in case your material gets corrupted you may check and uncheck this several times until the shader is back to normal mode.

> **Water Surface Position (Y)** This is needed when using water volumes and set up by script.

Directional Lighting Fade Range The distance over which the directional lighting (sun) fades out below the water surface. This affects caustics and lets you fade them out.

Fog Lighting Fade Range The distance over which underwater fog lighting will fade out below the water surface.

Please note: Both values above will be overwritten by script in case you enable “Deep Water Lighting” in the UnderWaterRendering script. Find out more >

Basic Properties

Smoothness Smoothness of the water surface which defines the size and shape of the specular highlights.

Specular Specular color which defines the reflectivity and thus drives the relation between refraction and reflection. Actually water has a pretty dark specular color.

Edge Blend Factor Lets you fade out the shore line.

Detail Distance Distance where Details will be fully faded out. Currently this only affects Gerstner waves and foam caps and lets you fade towards simple water planes. Furthermore it helps to reduce flickering and hides tiling.

Detail Fade Range Range over which details will fade.

Reflections

Reflections are the key to good looking water. So you need a proper skybox, reflection probe or planar reflection texture. Unity's default skybox won't give you nice reflections so consider using a custom skybox.

Enable Planar Reflections Check this in case you want to use planar reflections.

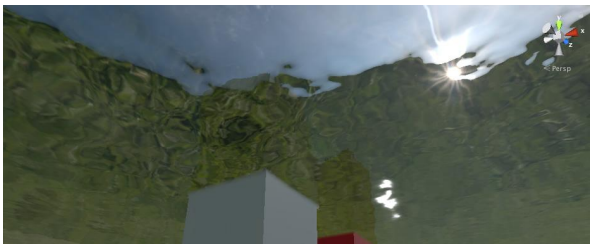
Fresnel Power Together with the *Specular* Color this specifies the relation between refraction and reflection. Using any other value than 5.0 will break physical correctness, but may help to improve the final look.

Strength Lets you adjust the strength or brightness of the reflections. Not physically correct, but nice to have.

Smoothness Lets you blur or sharpen cubemap based reflections independently from the direct specular highlights. It does not affect planar realtime reflections though.

Bump Scale Specifies the influence of the normal maps as far as the distortion on reflections is concerned: Choose values around 0.3 to get nice and smooth reflections.

Underwater IOR Lets you tweak the index of refraction when rendering the underwater surface. The standard value of 1.3333 (which would fit water at room temperature) may not always be the best choice as it leads to very strong internal reflections.



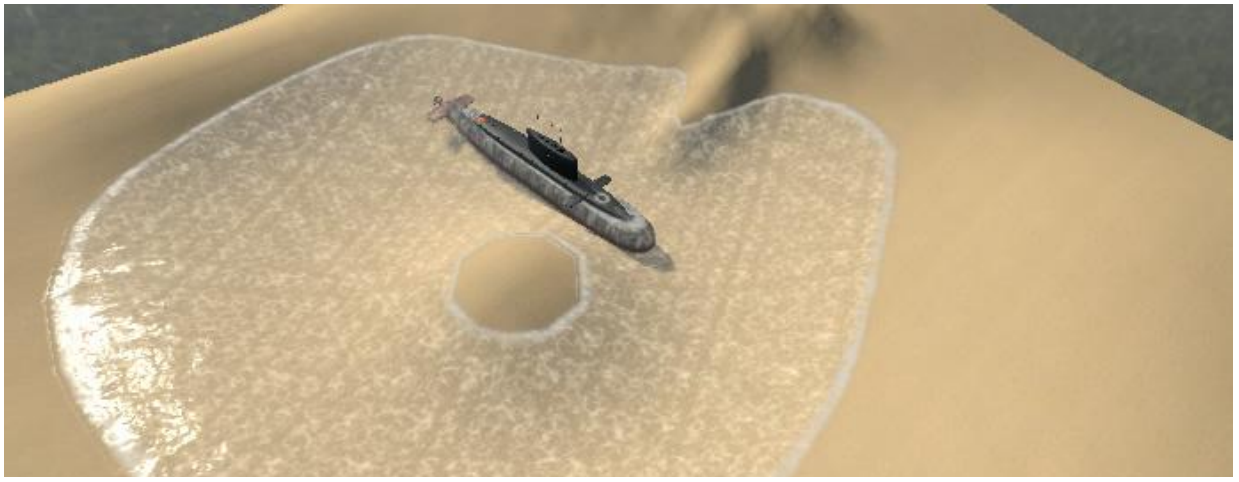
IOR 1.333 Internal reflections more or less cover the entire surface.

IOR 1.15 Lowering the IOR will give you less internal reflections.

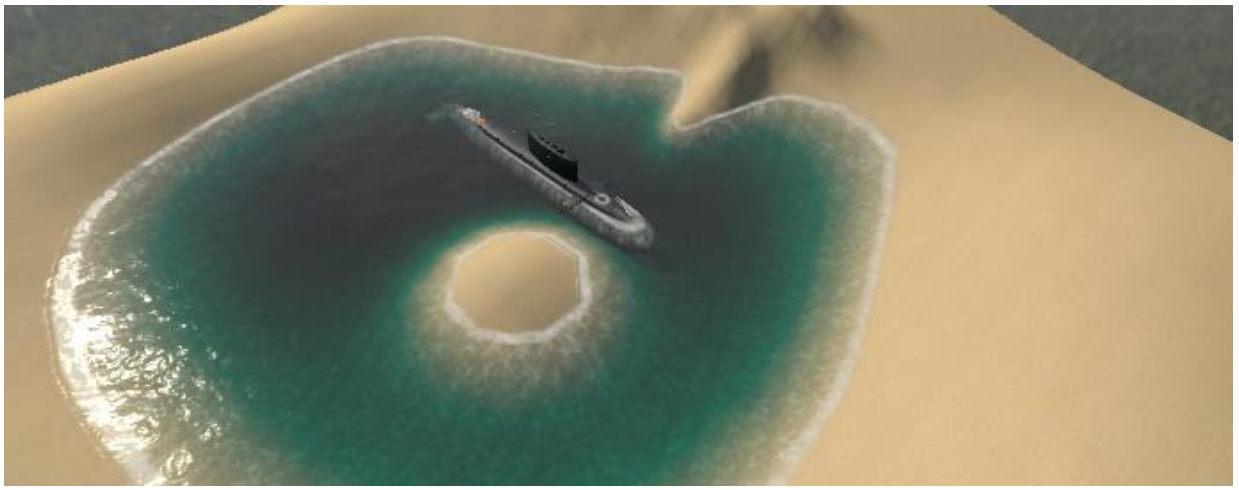
Underwater Reflection Tint As – when looking from below – the reflection will show up the bottom of the reflection cubemap, which most likely will not look like a reflection from below the water surface, you may specify a tint color here.

Underwater Fog and Light Absorption

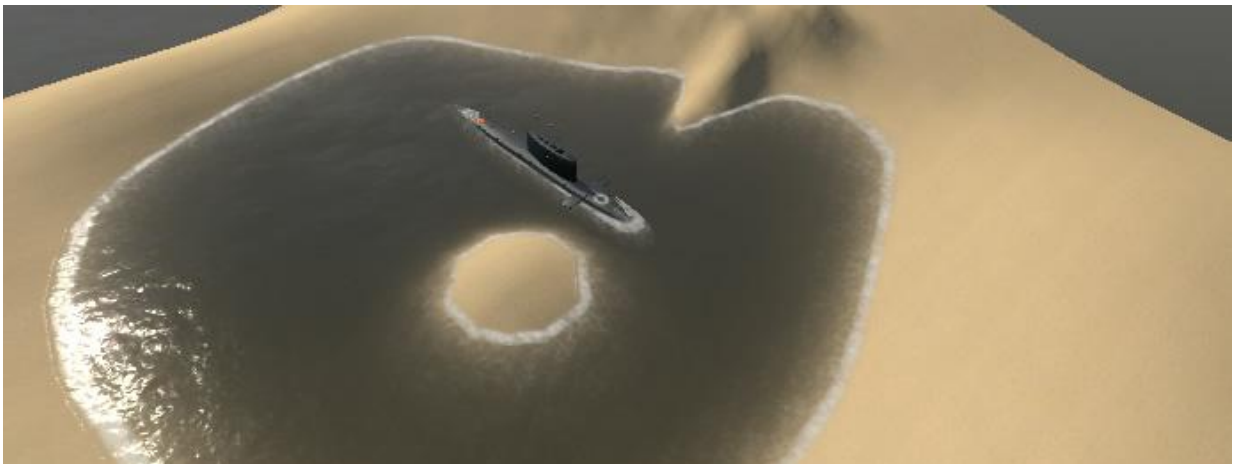
These two features go hand in hand and will turn water from something like a simple plane of glass into a water like volume.



The image above shows water without any underwater fog or absorption applied but only foam and caustics.



Adding absorption will give you some kind of caribbean sea, as while light travels through the water, it will get absorbed: *"While relatively small quantities of water appear to be colorless, pure water has a slight blue color that becomes a deeper blue as the thickness of the observed sample increases. The blue hue of water is an intrinsic property and is caused by selective absorption and scattering of white light."* Wikipedia [>]



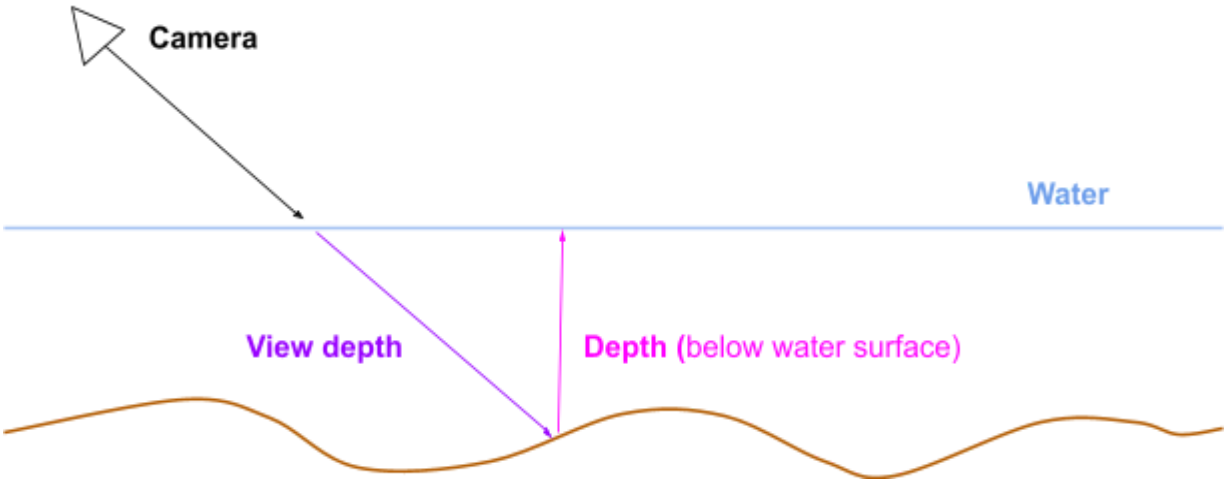
Adding underwater fog lets you tint the water: *"Dissolved and particulate material in water can cause discoloration. Slight discoloration is measured in Hazen units (HU).[8] Impurities can be deeply colored as well, for instance dissolved organic compounds called tannins can result in dark brown colors, or algae floating in the water (particles) can impart a green color."* Wikipedia [>]

Underwater fog may cancel absorption as light now will not have to travel all the way down to the ground but gets reflected way earlier.

No Cancellation The silhouette of the submarine is "quite" visible against the ground. It's fine but may not be what you want.

Full cancellation According to the fog's density light absorption will be reduced so the submarine fades smoothly with the water.

Both absorption and underwater fog are calculated taking **view depth** as well as the **depth** below the water surface into account.



While **view depth** is heavily view dependent **depth** will give you quite stable but unrealistic results. So the shader mixes both to get the best out of it.

Underwater Fog Inputs

Fog Color Color of fog added.

- **Depth** Specifies the amount of underwater fog absorption according to the depth below the water surface:

  Fade Start (X) Depth at which fog starts to fade in
  Fade Range (Y) Range over which fog will raise full density.
  Density (Z) Density multiplier

- **View Depth** Specifies the amount of underwater fog absorption according to the depth along the view ray.

Absorption Cancellation Lets you adjust the amount of *Color Absorption* relative to the amount of underwater fog.

Light Absorption Inputs

Strength Overall strength multiplier.

- **Depth** Specifies the amount of light absorption according to the depth below the water surface.
- **Max Depth** Specifies the depth at which light shall be fully absorbed.
- **View Depth** Specifies the amount of light absorption according to the depth along the view ray.

Color Absorption Specifies the amount of color absorption. If set to 0.0 water will only be darkened but not tinted.

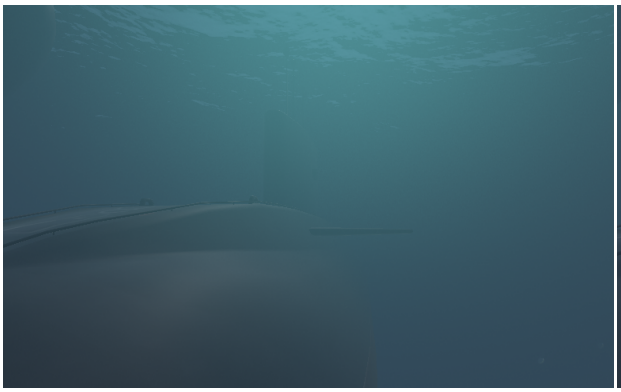
Subsurface Scattering

As Subsurface Scattering is derived from the final normal, view and light direction it heavily depends on your normals.

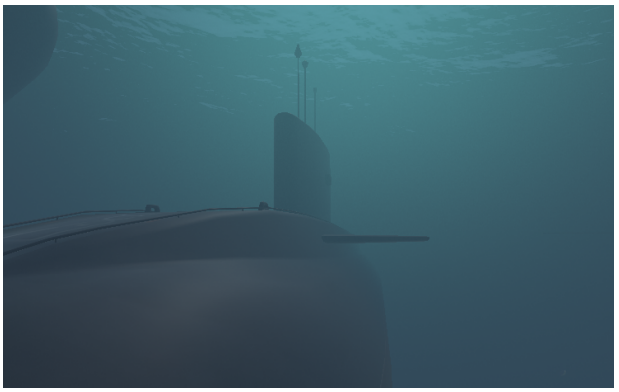
Color Tint color of the scattered light which gets multiplied by the light color.

Power Specifies the view dependency of the scattering.

Occlusion Lets you occlude scattered light based on the scene depth so objets close to the camera will receive less scattering. A value of 1.0 will make occlusion more or less vanish while a value of e.g. 0.03 will occlude scattered light on objects which are in a distance between 0 – 33m from the camera.



Occlusion =1.0.
The submarine vanishes into the scattered light.



Occlusion = 0.03.
The submarine occludes the scattered light.

Intensity (Water) Specifies the final factor for scattered light on water.

Intensity (Foam) Specifies the final factor for scattered light on foam.

Underwater Scattering

Underwater Scattering lets you control the forward scattering of the sunlight when the camera is underwater and looking towards the sun. Sun in this case means the (first) directional light. Other lights are not supported.

Color, Power and Occlusion are derived from the overall subsurface scattering settings.

Intensity Specifies the final brightness of the underwater scattering. You most likely will have to raise the underwater intensity to make it match the scattering as viewed from above the surface.

Underwater Scattering – Intensity = 0.5.
Underwater more or less is lit homogeneously.

Underwater Scattering – Intensity = 2.0.
Light gets a direction.

Normal Maps

Normals have the strongest influence on the final look of the water and drive – together with the actual geometry – all phenomena like reflection, refraction and subsurface scattering.

So you may consider using a low res (in order to save texture memory) but uncompressed normal map and make sure that “Filter Mode” is set to “Trilinear”. Using a RGBA32 bit bump map will not introduce any compression artifacts which otherwise will quantize direct and – even more important – indirect ambient specular reflections.

Lux Water mixes four normal samples to create the final bumpiness. The first sample kicks in at far distances to reduce tiling artifacts, while the detail samples are present all the time.

Far Normal

Tiling (X) Tiling of the far normal relative to the tiling of the first detail normal (as specified in Detail Normals -> Tiling -> X) The smaller the value (0.1 or even 0.01) the larger the far normals will get.

Distance (Y) Drives the fade between far normal and first detail normal. The far normal will be fully revealed at the given distance (meters).

Scale (Z, 0-1) Lets you downscale the far normal. Keep this value in the 0-1 range which means that you can't scale the far normals up (due to the optimized normal blending used), but usually you just want to scale them down anyway.

Speed (W) Multiplier on the Speed and Rotation derived from the first detail normal sample.

Detail Normals

Scale Lets you scale the bumpiness. X scales the first, Y the second and Z the third sample.
Tip: *Using negative values one one or two of the samples will increase diversity.*

Tiling Specifies the tiling for each sample.
Global Factor Specifies the base UV scale.

Speed Speed of the scroll animation for each sample.
Global Factor Overall speed multiplier.

Rotation Rotation of the scroll animation in degrees.
Global Factor Overall rotation in degrees.
Tip: *Use the global factor to easily adjust the orientation of the scroll animation according to the given wind direction for example.*

> **Please note:** Speed and Rotation will be combined into a Vector2 property by the material editor and get written to the hidden "_FinalBumpSpeed01" and "_FinalBumpSpeed23" Vector4 properties which actually are used by the shader. Please keep this in mind in case you want to change speed and rotation by script.

Refraction Specifies the strength of the refraction effect.

Foam

Like most water shaders Lux Water automatically calculates foam based on the distance between the water surface and background along the view ray — which makes foam highly view angle dependent and may lead to strange artifacts.

So i did not put that much effort into foam and it finally gets away with a single additional texture lookup and some math.

Regular Foam

Enable Foam If checked foam will be rendered.

Normal (RGB) Mask (A) The foam texture – where RGB stores the normal map and the foam mask. Please make sure that you have unchecked "sRGB Texture" in the import settings.

Tiling Tiling factor for the foam texture in world space.

Color (RGB) Opacity (A) RGB will tint the foam mask. Alpha drives the opacity.

Smoothness The smoothness of the foam.

Scale Foam is masked by various inputs such as the water's normals and the edge blending factor so it might get more or less invisible. Use *Scale* to bring it back to screen.

Speed Speed and direction of the foam animation are derived from the animation of the first detail normal sample. The foam's speed parameter acts as a multiplier on this. Using negative values will make foam slower than the first detail normal.

Parallax Parallax offset for sampling the foam texture derived from the water's normals.

Normal Scale Let's you scale the foam's normal.

Edge Blend Factor Determines the size of the dynamically calculated foam border.

Mask by Normal Lets you mask the foam by the detail normals.

Slope aware foam Determines the foam amount added automatically according to the given surface slope of the water. Super simple but quite effective when it comes to rivers and rapids or small waterfalls.

> **Suppress Mask by Normal** Lets you fade out the foam mask generated from the detail normals on steep slopes.

Final Foam Erosion Lets tweak the final "sharpness" of the foam. Higher values will result in smoother foam shapes. The default value was 0.375.

Gerstner Foam

Gerstner foam or foam caps are only rendered if Gerstner waves are enabled.

Foam Caps Specifies the strength of the foam caps as calculated from the final displacement of the vertices. Basic properties like tiling, color and scale are taken from the regular foam settings.

> **Mask by Elevation** If set to 1 mainly positive displacement along y will form the final foam caps. If set to 0 Gerstner displacement along the xz axies are also taken into account.

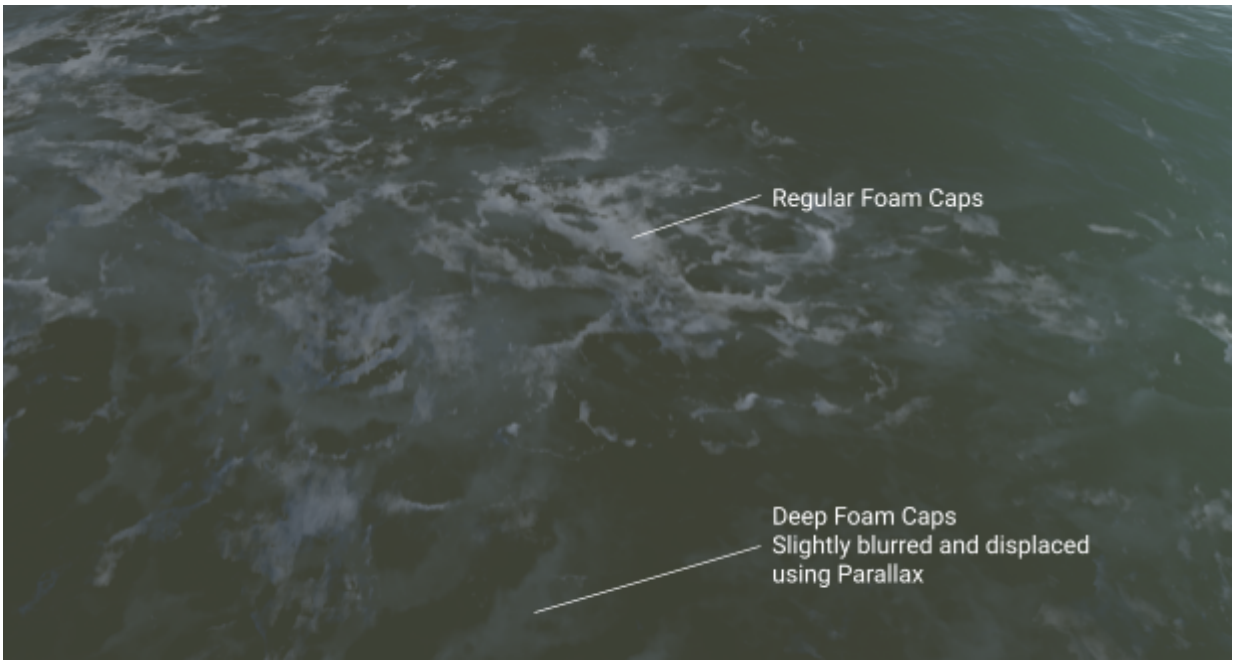
Deep Foam Caps Factor on top of *Foam Caps* in order to scale deep foam caps. So if you decrease *Foam Scale* Deep Foam caps will shrink as well.

Deep Foam Color (RGB) Alpha (A) Color and Opacity of the deep foam caps.

Deep Foam Parallax Parallax offset for sampling the deep foam (which only gets rendered when gerstner waves are enabled). Lets you actually increase the depth illusion.

Deep Foam Tiling Tiling relative to the regular foam texture tiling.

Deep Foam Blur Lets you "blur" the deep foam sample by simply using tex2Dbias.



Caustics

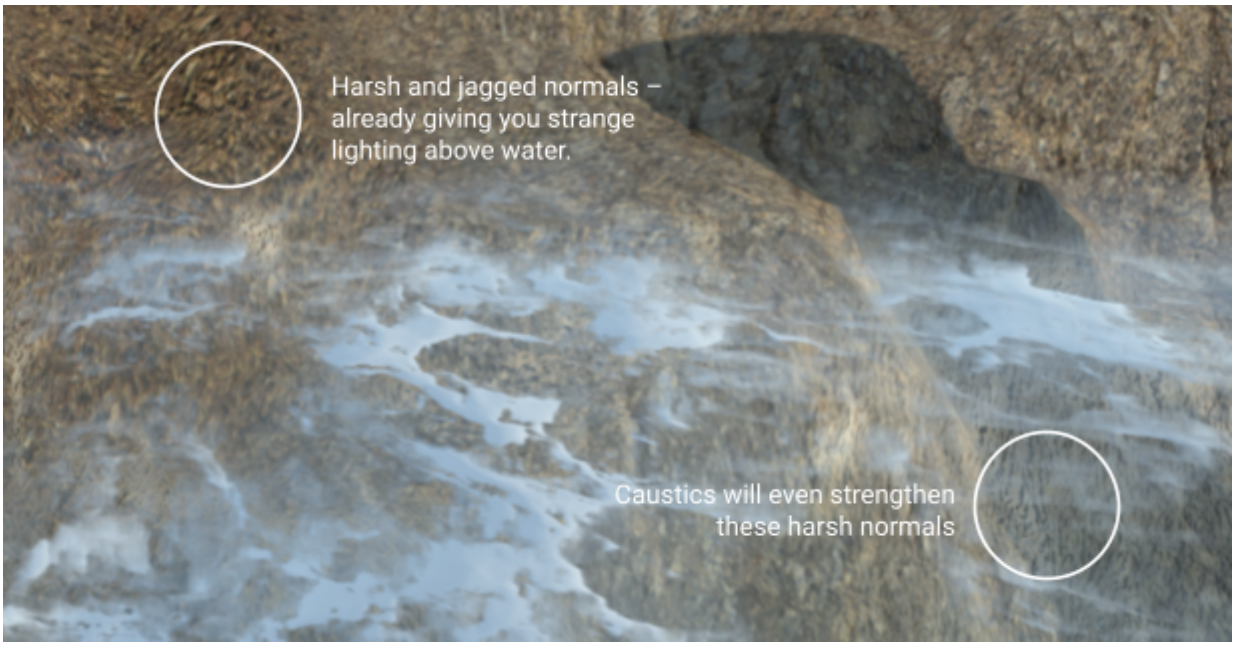
Lux Water does not use any projectors to render caustics but renders them together with the water surface directly to screen taking the depth texture and – in case you use deferred rendering – the deferred normals into account. So unlike other water solutions Lux Water does not have to draw your scene multiple times to get in caustics.

In case you use forward rendering there is no GBuffer and thus no deferred normals. So the shader will reconstruct normals from the depth buffer only which unfortunately does not produce as smooth gradients as deferred normals do. Furthermore forward normals produce gaps/ghosts as they are based on the unrefracted depth buffer.

Deferred normals. Smooth fading of the caustics according to the given normals.

Normals reconstructed from the depth texture. The gradient gets quantized.

In case you use high frequent or even jagged normal maps especially on unity terrains caustics will reveal these and create rather harsh contrasts in shading. Solve this by creating proper normal maps or reduce the caustics scale.



Caustics will add three more texture lookups for the caustic animation + one in case you use deferred normals. They are real time lit and react to directional, point and spot lights automatically.

Caustics are top down projected in world space. So in order to avoid any stretching artifacts they are cancelled by the reconstructed normal which means that there will not be any caustics on steeply angled geometry.

Caustics previously only got attenuated by depth based Color Absorption or Fog. You now can specify a "Max directional Light Depth" in the water material or in case you use deferred and the new deep water lighting by setting it globally using "Directional Lighting Fade Range".

Same goes with Fog which now fades out its lighting according to the position of the camera below the water surface. The fade range can be specified in the material or by script (LuxWater_UnderWaterRendering.cs).

Please note: Script set values will overwrite values in the material.

Inputs

Enable Caustics If checked caustic will be rendered.

Normals from GBuffer Check this in case you use deferred rendering in order to get high quality caustics. In case you use forward rendering you have to uncheck this.

Caustics (R) Noise (GB) The caustics texture which should contain the caustics mask in R and some noise in GB. Noise is used to distort the caustics mask of the next sample.

Tiling Tiling of the caustics texture in world space.

Scale Specifies the strength of the projected caustics.

Speed Speed and direction of the caustics animation are derived from the animation of the first detail normal sample. The caustics' speed parameter acts as a multiplier on this.

Distortion Specifies the amount of distortion applied to the UVs as retrieved from the noise sample (GB). Please note: The first caustic texture lookup does not get distorted.

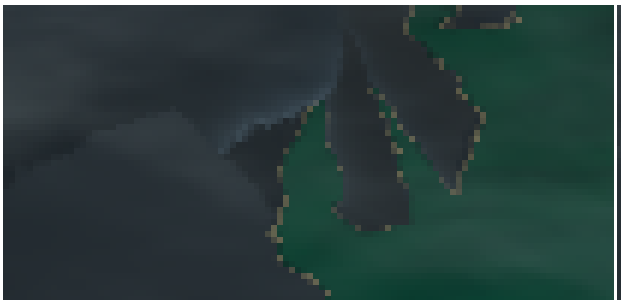
Advanced Options

Pixel Snapping If set to **“Point”** the shader will snap the distorted GrabUVs to pixels, as otherwise the depth texture lookup will return a false depth, which may lead to a 1 pixel error (caused by fog and absorption) at high depth and color discontinuities.

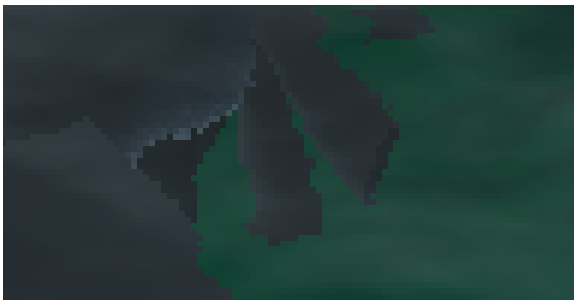
This artifact is barely visible on platforms using a reversed zBuffer like dx11, but it still exists.

Imagine you have a dark submarine and a rather bright terrain as bottom of your water volume, then the grabtexture returns a lerped value from submarine (black) and the terrain (sand), but fog and absorption may be too weak as the depth texture lookup returns a value somewhere between the submarine and the terrain.

Enabling snapping will add some more math to the shader, but most significantly, it will some kind of point sample the refractions resulting in harsher refractions.



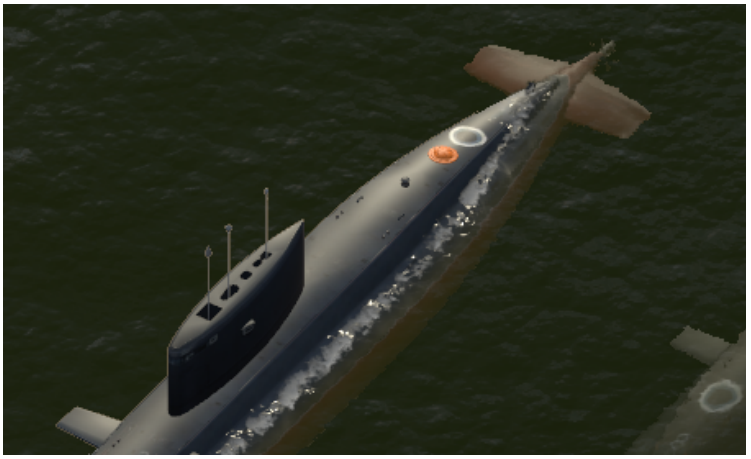
No snapping. Please notice the bright pixels at the edge of the geometry.



Snapping enabled.

If “Pixel Snapping” is set to **“MSAA_4x”** (experimental) the shader tries to find the best matching depth in case you use forward rendering and MSAA. “MSAA_4x” is quite expensive so only activate it in case it really helps.

Using “MSAA_4x” can only solve issues introduced by refraction. Aliasing artifacts around the silhouette of objects above the water surface can’t be solved.



MSAA_4x limits While there are no “bright pixels” around the propellers and anything below the water surface in the screenshot above you will get some around the towers and antennas of the submarines: Here the pixels on screen are already taken by the terrain in the background (which draws first) and the submarine (which comes next). When water is drawn (after the submarine as it is transparent) the depth buffer is already occupied by terrain and submarine

and water simply will not be drawn at the bright edges you see. These show up the anti aliased combination of terrain and submarine, but no water.

You may solve this by setting “ZTest” to “Disabled” and fully rely on “Edge Blending”. This however may cause strange artifacts with perspective projection and any vertex displacement like Gerstner Waves.

The screenshot below shows orthographic projection and water using “ZTest” set to “Disabled”: No more bright pixels but the antennas are almost gone.

Gerstner Waves

Gerstner waves are based upon Unity’s Water4 shader. They will actually displace the water geometry in the vertex shader and therefore need a nicely subdivided water geometry.

Enable Gerstner Waves If enabled the shader will Gerstner based vertex displacement to the water mesh. Please make sure that your mesh has a nice factor of subdivisions.

Amplitude Height of the waves, which gets multiplied with the *Final Displacement Y* value.

Frequency Distance between or size of the waves.
The *Global Factor* below lets you scale all four values at once.

Steepness Steepness of the waves which controls extraction and contraction along the xz axis in worldspace.

Speed Speed of the waves.
The *Global Factor* below lets you scale all four values at once.

Rotation Rotation of the waves’ direction in degrees.
The *Global Factor* below lets you rotate all four values at once.

Final Displacement Acts as a multiplier on *Amplitude* and *Steepness* and lets you finetune the waves.

Normal Scale Scale which will be applied to the geometry’s normals. As Gerstner normals are quite off, they will most likely give your some very strange reflections. So use a rather low *Scale* to reduce shading artifacts.

Please note: All values, which have a "global factor" assigned, will be calculated by the material editor and finally written to hidden properties. Please keep this in mind in case you want to change these by script.

Secondary Gerstner Waves

By default the shader only calculates 4 Gerstner Waves. You may use 8 tho. To enable the 2nd set of Gerstner Waves just make sure that the x-component of the *Factors* (Amplitude) is greater than 0.0. In order to keep things simple the secondary set will be derived from the first set. All you specify are factors for the most relevant inputs.

In case Amplitude is set to 0 the vertex shaders will efficiently skip the second wave calculation.

Factors Lets you tune the secondary set by specifying factors for:

- **Amplitude (X)**
- **Frequency (Y)**
- **Steepness (Z)**
- **Speed (W)**

Rotation (Directions) will be swizzled between waves: While the first set uses XYZW the second uses YXWZ.

Tessellation

Please note: Tessellation params are only available if you have assigned the Lux Water/WaterSurface Tessellation" shader.

Edge Length Defines the tessellation level based on triangle edge length on the screen. The smaller the edge length value the more tessellation you will get.

Extrusion In case you use Normal Projectors these may not only influence the final water normal but also extrude the water surface. The final extrusion is driven by the height map in the projector's texture and this factor.

Orthographic Projection

When using orthographic projection your camera will automatically be set to forward – which means that you will have to add the *LuxWater_CameraDepthMode* script to your camera.

In case your camera uses orthographic projection not all features are supported or work as in perspective projection and you most likely will have to tweak your materials to make them look properly.

Unsupported features

- **Water volumes** – as they do not make much sense.

Feature which need some Love

- **Refraction** – which needs smaller values

Planar Reflections

Planar reflections are based upon Unity's Water4 shader. In order to enable real time planar reflections you have to attach the "LuxWater_PlanarReflection" script to your water object. Additionally you have to check **enable planar reflections** in the material.

In case you have multiple water surfaces or water tiles, please have a look at Using multiple water tiles

Update Scene View If checked the real time reflection texture will be updated in the scene view. In case you use multiple water tiles this may cause a huge amount of draw calls and slow down the editor. So consider unchecking this unless you really need it.

> **Is Master** is only relevant in case you have multiple water surfaces. Please have a look at Using multiple water tiles
>
> **Water Materials** is only relevant in case you have multiple water surfaces. Please have a look at Using multiple water tiles

Reflection Mask Includes or omits layers to be rendered by the reflection camera. Less included layers will improve performance.

Resolution The resolution of the reflection texture relative to the screen resolution.

Clear Color The color applied to the remaining screen after all elements in view have been drawn and there is no skybox.

Reflect Skybox If enabled the reflection camera will clear using the skybox.

Disable Pixel Lights Lets you disable all pixel lights when rendering the reflection.

Render Shadows If unchecked shadow will be dropped while the reflection camera renders.

> **Shadow Distance** Lets you overwrite the current shadow distance. If it is set to 0.0f the current shadow distance will be used.
>
> **Shadow Cascades** Lets you overwrite the number of shadow cascades.

Water Surface Offset Usually the reflection camera is positioned taking the water object's pivot into account. In case this does not match the water surface (as you are using a volume rather than a plane) you may adjust the position of the reflection camera using this offset.

Clip Plane Offset Lets you offset the reflection.

Using multiple water tiles

In case you use several independent game objects and meshes to compose your final water surface as you may be used from Unity's water, you should have the "LuxWater_PlanarReflection" script only once in your scene as otherwise you will create a vast amount of draw calls.

In order to do so:

- Create an empty gameobject, attach the "LuxWater_PlanarReflection" script to this object and check **IsMaster**.
- Add your water tiles as child objects and make sure that they have the "LuxWater_PlanarWaterTile" script attached.

- Make sure that the parent game object has the same y position as the water tiles using “GameObject” → “Center on Children”.
- Go back to the “LuxWater_PlanarReflection” script on parent object and assign the related water materials:
    - Set “Size” to 1.
    - Then drag the water material used by the water tiles to the slot “Element 0”.
    - In case you want to use more than one water material, increase the size of the array and assign the other water materials too.
- Check “Enable Planar Reflections” on all affected water materials.

Please have a look at the “LuxWater Fjord Demo” to find out more.

Troubleshooting

Particles do not show up in planar reflections or are partially hidden

Particles using the new *standard lit* and *unlit* particle shaders do not show up at all as they use back face culling which – in conjunction with the oblique projection matrix of the reflection camera – make them being culled by the gpu.

Legacy particle shaders most likely will show particles. But most of them incorporate "soft particles" which are based on the depth buffer and the clip space position. This does not work in case you have an oblique projection matrix like in case of the reflection camera.

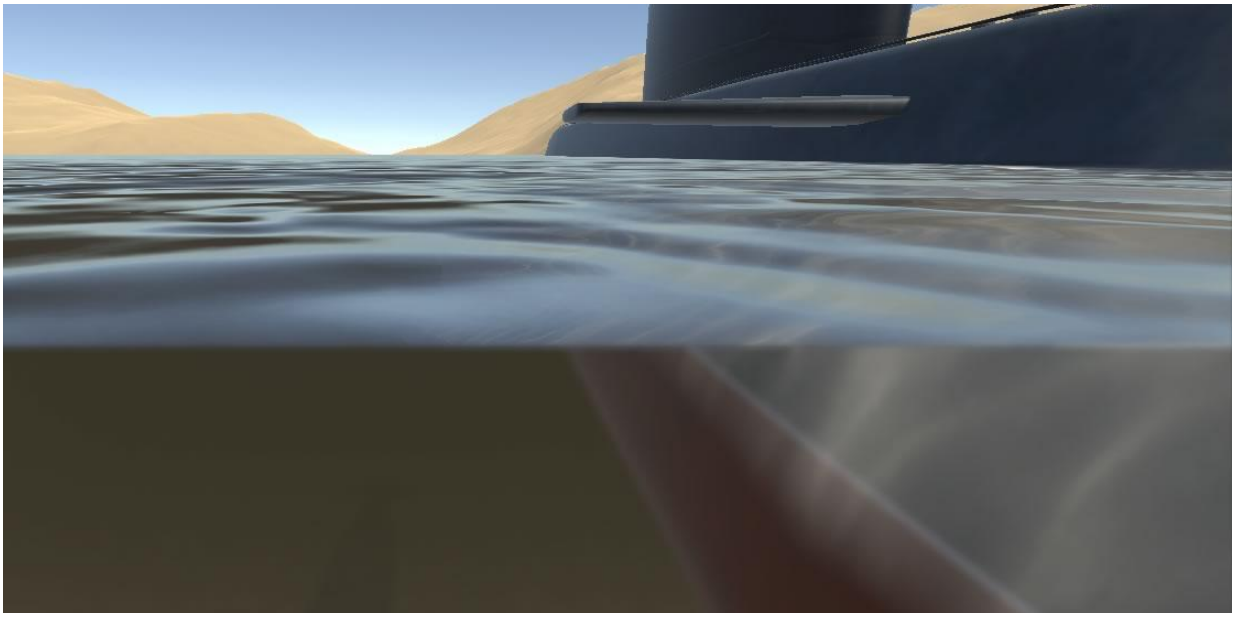
In order to be safe you should use a legacy particle shader which does not support *soft particles* like the particles/vertexlit blended shader.

You can also write your own particle shader but you have to make sure that the shader 1) uses “Cull Off” and 2) does not use the soft particle feature – at least not when the reflection camera renders.

Water Volumes

By introducing water volumes Lux Water lets you create seamless transitions from above to underwater rendering.

Lux Water does so by rendering a water mask and using it to combine above and below water rendering which is done as an image effect.



Adding a water volume

1. Add the “LuxWater_WaterVolume.cs” script to your water plane.
2. Assign a proper “Water Volume Mesh” to the corresponding slot in the script inspector.
3. Add a box collider to your water plane, make sure it has a proper size and check “Is Trigger”.
4. Make sure that your camera has a collider and a Rigidbody assigned. Check “Is Kinematic” in the Rigidbody inspector. Then add the “LuxWater_WaterVolume-Trigger.cs” to the camera.
   The collision between the assigned collider and the box collider of the water volume will trigger the underwater rendering. In order to prevent other colliders to trigger the water volume rendering the water volume script will check if the object that triggers the collision has a “LuxWater_WaterVolumeTrigger.cs” component.
   So if you already have a collider and rigidbody (like on your player) you may use this these as triggers by adding the “LuxWater_WaterVolumeTrigger.cs” component.
5. Add the “LuxWater_UnderwaterRendering.cs” script to your camera.
6. You may add the “LuxWater_UnderWaterBlur.cs” to your camera in case you want underwater to be blurred.

   Please have a look at the “LuxWater Water Volume Demo” to find out more.

The script components

LuxWater_UnderWaterRendering.cs

This script must be attached to the camera that will render the water. It will make sure that a proper water mask texture will be rendered and kicks off the image effect which will add underwater fog and caustics to the underwater part of the screen.

Inputs

Sun You have to assign the dominant directional light here which is used to lit underwater fog in the image effect. In case you can't do so look at:

Find Sun On Enable In case you spawn your player or camera from a prefab you won't be able to assign a sun. So check this to make the script look for a sun OnEnable.

> **Sun Go Name** Add the sun's gameobject name here if you want to search for it by name (slow).
>
> **Sun Tag Name** Add the sun's tag name here if you want to search for it by tag (fast). Do not add anything to the *Sun Go Name* if you want to look for it by tag.

Deep Water Lighting lets you attenuate light from the sun according to the depth below the water surface while it keeps local pixel lights intact. Find out more >

> **Enable Deepwater Lighting** If checked Deep Water Lighting will be enabled.
>
> **Default Water Surface Position** Default Water Surface Position which gets sent to all shaders if no water volume is active.
>
> **Directional Lighting Fade Range** The distance over which the directional lighting (sun) fades out below the water surface. This affects caustics and lets you fade them out.
>
> **Fog Lighting Fade Range** The distance over which underwater fog lighting will fade out below the water surface.
>
> **Please note:** Both values above will overwrite the corresponding values in all registered water materials.

Advanced Deferred Fog In case you are using deferred rendering and a properly tweaked fog shader Advanced Deferred Fog will improve the final rendering result. Find out more >

> **Enable Advanced Deferred Fog** Check this to enable Advanced Deferred Fog.
>
> **Fog Depth Shift** Lets you shift the depth below the water surface at which fog starts to fade out.
>
> **Fog Edge Blending** Lets you sharpen and smoothen the fading.

Managed transparent Materials In order to get somewhat proper results regarding the rendering of water and transparent objects and particle systems, the script may handle transparent materials and change their render queue when the camera moves from above to under water. See: Transparents and particle systems for further details.

> **Above Watersurface** List of managed materials to which you can add materials that are used by objects/particle systems above the water surface.
>
> **Below Watersurface** List of managed materials to which you can add materials that are used by objects/particle systems below the water surface.

Optimize

Prewarmed Shaders You may assign a Shader Variant Collection here of all shaders needed by underwater rendering. This will reduce hiccups when underwater rendering becomes active.

List Capacity Capacity of the internally used lists as set OnEnable. Make this match the number of max active water volumes to prevent garbage which otherwise would be created in case the lists had to be resized. Keeping the initial value of 10 should be ok and won't take much memory.

Enable Debug If checked the name of the current active water volume will be displayed in the game view (in play mode only) along with a preview of the generated water mask. In order to make the water mask show up you have to check "Gizmos" in the upper right corner of the game view window. Red colors in the water mask represent the water surface from above whereas green colors represent the water surface from below and the volume.

LuxWater_UnderWaterRenderingSlave.cs

In case you use split screens add this script to the 2nd, 3rd, … camera. Please make sure that the camera that has the LuxWater_UnderWaterRendering.cs attached to is rendered first by setting up the *Depth* properties of the cameras properly.

LuxWater_UnderWaterBlur.cs

This script adds a blur image effect to the parts of the screen which are below the water surface. It must be attached to the camera that will render the water.

Inputs

Blur Spread Defines the sampling radius. Larger values will make the result blurrier.

Blur Down Sample Factor by which the grabbed screen will be downsampled before it gets blurred. Larger factors will speed up the effect but might lead to quite blocky results.

Blur Iterations Number of blur iterations. The higher the values, the nicer the result – but the more expensive the effect as well.

LuxWater_WaterVolumeTrigger.cs

This script must be attached to the game object which holds the collider and rigidbody that shall trigger underwater rendering – most likely your camera.

Inputs

Cam In case you did not add the script to the camera itself but add it to your player's prefab instead, you have to assign your player's camera here. If no camera is defined the script assumes that it is assigned to a camera and will grab the camera component automatically.

Active If unchecked underwater rendering will not be triggered even on collision.

LuxWater_WaterVolume.cs

This script has to be attached to the water plane that shall be rendered as water volume. It will register the associated water plane with the UnderWaterRendering script, so that the latter can handle the rendering.

Inputs

Water Volume Mesh The mesh which is used to render the water mask.

Sliding Volume Check this if you use a huge water surface formed out of several water meshes. Find out more >

Grid Size The distance between the vertices of the water surface mesh at a size of 1.0. If the water surface mesh is scaled in your scene (either itself or inherits a different scale from its parent object) the scripts will adjust it automatically.

Listen to Collision Events

The script lets you subscribe to the OnTriggerEnter and OnTriggerExit functions using:

```
LuxWater.LuxWater_WaterVolume.OnEnterWaterVolume += YourEnterFunction;
LuxWater.LuxWater_WaterVolume.OnExitWaterVolume += YourExitFunction;
```

See *LuxWater_WaterVolumeListener.cs* for details.

OnEnterWaterVolume and OnExitWaterVolume are only fired if the collider which triggered the event has the *LuxWater_WaterVolumeTrigger.cs* attached to it.

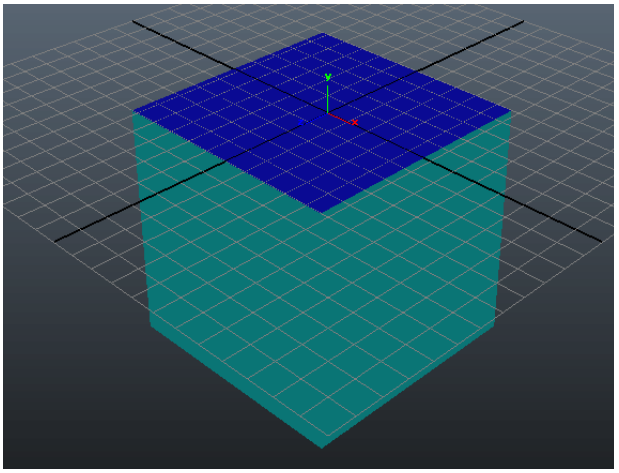
When using Gerstner waves and a slightly oversized collider *OnTriggerEnter* does not guarantee that your camera (or the trigger) actually is underwater. You will have to check the Gerstner height at the given position to make sure the camera is underwater or not.

Water Volume Mesh

When using water volumes, Unity has to know when to render water volumes and where to render them. *When* is determined by the collider and triggers. *Where* is derived from the position, rotation and scale of the original water surface plane. So the assigned “Water Volume Mesh” has to 100% match your water surface mesh – except for the fact that it describes a volume of course.

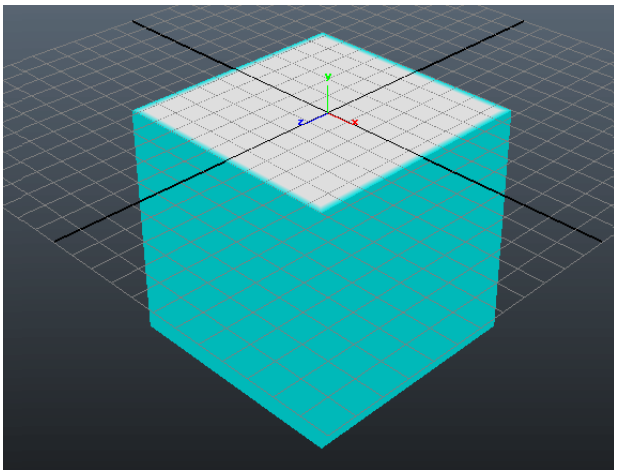
Submeshes and materials

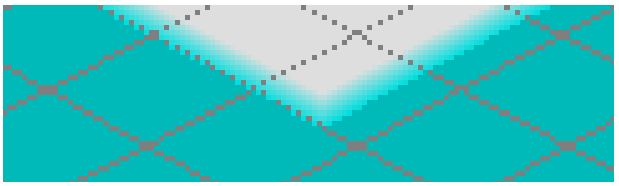
The water volume mesh needs 2 submeshes or materials: 1st one must describe the volume (cyan), 2nd one the actual water surface (blue).



Vertex Colors

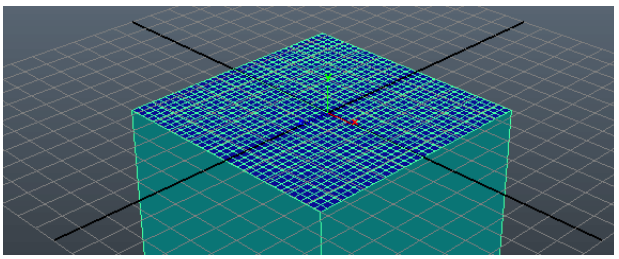
In case you want to use Gerstner Waves you should add vertex colors to the meshes (both the water volume mesh and the mesh used as water plane) in order to prevent the outer edges from getting disconnected. Simply add vertex color red = 0.0 to the outer vertices and you are fine.





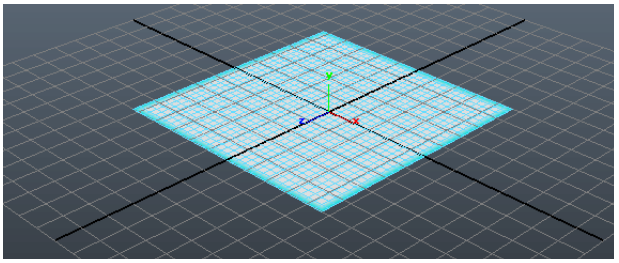
Tessellation

In case you want to use Gerstner Waves the water surface should have a decent amount of tessellation.



Water surface mesh

The mesh used for the water surface is just exactly the same – except from the missing volume faces.



Normals

The normals of the water volume mesh should be regular ones – so all of them should point outwards in case of the example cube. *In order to avoid micro cracks when using tessellation I recommend setting the normal angle to 180° so they all get smoothed out.*

UVs

The water volume mesh does not need any UVs.

Pivot

The pivot should be placed at the water surface as it is taken to compute the “depth below water surface” in the shaders. It must match the pivot of the plane used for the water surface.

Height of the volume

Make sure that the height or extent along the y axis is big enough to cover the entire space between water surface and ground. Actually you can simply make it “super” high as it does only produce a very little amount of extra costs.

Lux Water WaterSurface Shader

When using water volumes we can’t compute the depth below the water surface per pixel – as this would not work in the UnderWaterPost shader. For this reason the Water shader will fall back and pick up a float input instead (_WaterSurfaceYPos or "Water Surface Position (Y)").

Transparents and particle systems

As water is transparent and most likely pretty large Unity’s sorting for transparents will fail in most cases, so water might hide or overdraw a lot of your transparent materials and particles. For this reason Lux Water renders on RenderQueue = 2999 by default. This ensures that water will be rendered before all regular transparent materials including particles which usually render at RenderQueue = 3000.

Transparents *above* water surface

This is fine for all transparent materials which are placed above the water surface – as long as the camera is outside the water volume. However when the camera is inside the water volume these materials would be drawn on top of the underwater surface. A standard particle system not handled by script will be invisible from below the water surface.

So Lux Water introduces **managed transparent materials** whose render queues get adjusted whenever the camera crosses the water surface. This change of course is not pixel accurate but good enough in most cases.

Particles drawn after the underwater surface.
Particles may simply get drawn on top of the underwater surface. They are neither occluded by the total reflections nor refracted – suitable for particles under the water surface.

Particles drawn before the underwater surface.
The particles are properly occluded by the total reflections from the underwater surface and are refracted properly – suitable for particles above the water surface.

Transparents *below* water surface

Transparent materials below the water surface are a completely different case. They should not receive Unity's built in fog but the underwater fog and color absorption as defined in the water volume's material. At the same time they should render *after* the water in order to not be occluded using RenderQueue = 3001 – at least when the camera is below the water surface.

However this would make them stand out when looking from above the water surface as they would not be refracted. So these transparents should be handled as well by script and their Renderqueue should be set to 2998 to make sure that they get rendered before the water.

Doing so however will most likely make them disappear when looking from above the water surface. Just a little tradeoff we have to take here – which in most cases is barely noticeable. Especially if you use rather subtle underwater particle effects.

I swear I hate solving rendering problems using scripts, but in this case it was just the most performant solution. Doing it all by shaders alone would need two grab passes and the water being rendered twice.

Lux Water/Particles/UnderwaterParticles Alpha Blended shader

This shader is derived from the built in Particles Alpha Blended – but instead of adding Unity fog this shader adds underwater fog as specified by the active water volume.

This being said it should be clear that using this shader needs you to use water volumes. It needs the "LuxWater_UnderWaterRendering" script as this provides the shader with all needed variables.

Lux Water/Particles/Like Alpha Blended Premultiply shader

As Unity's built in Alpha Blended Premultiply particle shader does not support fog i added a shader which more or less acts similarly but supports fog. Texture input expects a regular alpha texture (RGB + A): You do not have to multiply alpha on top of RGB.

Fog

You do not want any built in fog on pixels which are underwater – but underwater fog instead. So we have to mask out the corresponding pixels.

This is rather easy when it comes to deferred rendering as fog is applied as an image effect.

Please note: Properly tweaked fog shaders for the PostProcessingStack v1 and v2 are added. You will find them in the folder "_FogShaders". Move them to the appropriate folder of your PostProcessingStack, backup the default fog shaders, then remove the "__" from their file name.

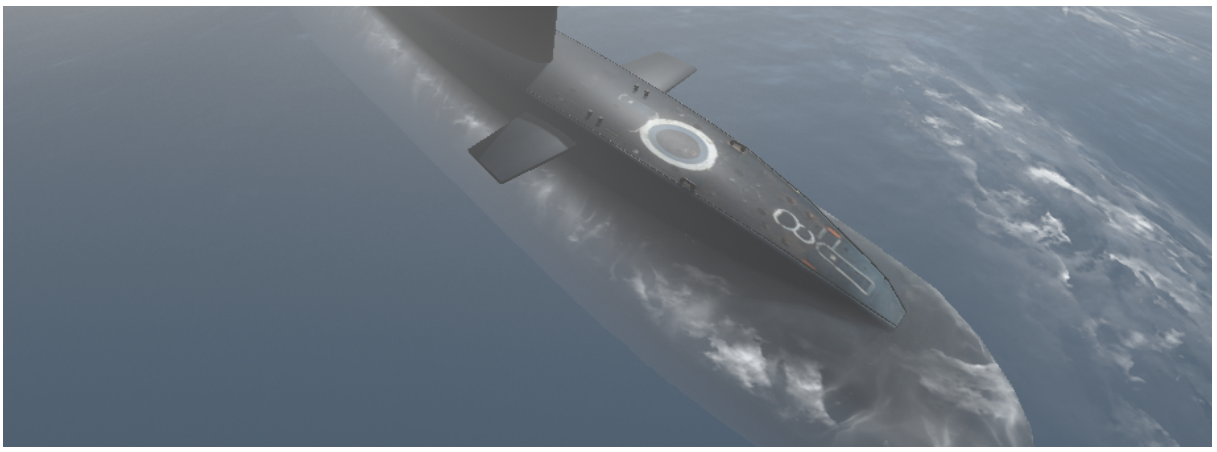
Fog rendered by **Azure** or **Enviro** is also supported. In order to make these work you will have to edit the "LuxWater_Setup.cginc" file (located in: LuxWater → Shaders → Includes) and comment/uncomment the corresponding defines.

However when using forward rendering this gets pretty tricky as usually fog is applied directly when the geometry gets drawn to screen. So when using forward we have to live with the fact that everything receives fog and have to pray that underwater fog and absorption will actually hide it :(

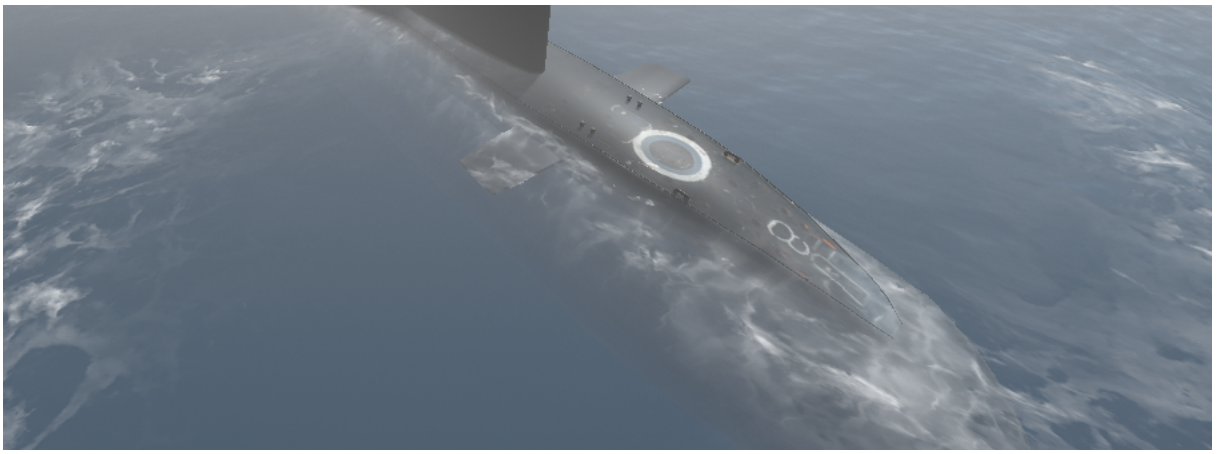
If Advanced Deferred Fog is enabled the (tweaked!) fog shader will sample the water mask and mask all pixels inside the water volume from being fogged.

Furthermore it compares the depth from the depth buffer against depth from the water surface and lets you fade out fog below the water surface even if the camera is not inside a water volume. This helps to prevent objects in water getting too bright (caused by the fact that they receive double the amount of fog: 1st from the object itself + 2nd from the water surface) and stops them standing out.

The result of this is illustrated in the following screenshots:



Fog Depth Shift: 1, Fog Edge Blending: 0.001 Here fog reveals the silhouette of the submarine as fog sums up: Both submarine and water surface get fogged.



Fog Depth Shift: 1, Fog Edge Blending: 0.4 If we sharpen the “Fog Edge Blending” by raising its value we can make fog fade out earlier on the opaque objects according to their depth below the water surface and get a nicer blending between objects and water.

Fog Depth Shift: 0, Fog Edge Blending: 8.0 However sharp fog edge blending may produce areas where there is no fog at all (the dark parts in the screenshot). Use “Fog Depth Shift” to lower the height where fog fading starts.

Actually we have to set **Fog Depth Shift** and **Fog Edge Blending** so that we get a little overlap between fog being faded out and the overall **Edge Blend Factor** of the water material.

Please make sure that you check your settings by entering playmode. Then find some intersections between water and objects and look if there are any unfogged areas like shown below. If so tweak **Fog Depth Shift** and **Fog Edge Blending.**

Tip: *Even if you do not want to use real water volumes you can use its components to get advanced deferred fog: Simply do not attach a trigger volume (collider) to your water mesh, then underwater rendering will never be triggered. Assign the water mesh itself as “Water Volume Mesh”.*

Limitations

Currently water volumes can’t be bigger than the camera’s far clip plane. So if the latter is set to 1000 the max size of your water volume would be 577 along each axis.

In order to have larger water surfaces you will have to split them into several chunks and use sliding water volumes – which from a performance point of view might be a good idea anyway.

Find out more >

Water Volumes and split screen rendering

In case you want to use water volumes and split screen rendering you have to:

- Specify a master camera. This camera should render first. In order to ensure this, set its “Depth” to “-1”. This camera must have the “LuxWater_UnderWaterRendering” script attached to.
- All other cameras should use higher “Depth” values and must have the “LuxWater_UnderwaterRenderingSlave” script attached to them.

Please note: Split screen rendering currently is in preview and only supports one single active water volume. Please have a look at the “LuxWater Water Volume Splitscreen Demo” to find out more.

Deep water rendering

What does it do

Previous versions of Lux Water let you calculate light and color absorption according to the reconstructed depth below the water surface which gave you way more stable results when rendering the water surface from above compared to only view depth based absorption. However when the camera was inside the water volume the same calculation took place simply attenuating ALL light according to the depth – no matter if these lights were above or below the water surface.

The problem here is to distinguish between sunlight from outside the water and lights within the water. The underwater post effect can’t handle this and simply darkens all pixels according to their eye depth and depth below the water surface.

Deep water rendering addresses this issue as it makes the (non water) scene materials take depth based attenuation of sunlight into account (see screenshot below). This is done in a custom Deferred Lighting shader thus Lux Water supports deep water rendering **only when using deferred lighting**, as deferred lighting lets us globally hack into the lighting pipeline.

Other issues are caused by underwater fog and caustics. Both are lit by the sun and react to absorption in case fog does not fully cancel it. Now you can specify a “Directional Lighting Fade Range” and “Fog Lighting Fade Range” over which caustic and fog lighting will be faded out. Caustic lighting will take the distance between water surface and the scene’s y position into account while fog lighting uses the distance between the water surface and the position of the camera.

Setting it up

In order to make deep water rendering work you have use a water volume of course, set your camera to deferred and assign the *LuxWater_DeferredShading* in *Edit → Project Settings → Graphics*:

- Under the *Built-in shader settings* change *Deferred* to *custom*, then assign the *LuxWater_DeferredShading* shader.
- Add the *Under Water Rendering* script to your camera and check *Enable Deep Water Lighting*.
- Set *Default Water Surface Position* to the y position of your water plane then adjust the *Directional Lighting Fade Range*.

- In case you want to render deep waters and use local pixel lights under the water surface at the same time you must **NOT** use any **depth based Light Absorption** at all – as this would cancel lighting from the directional light (sun) as well as from any other pixel light. So set *Light Absorption → - Depth*: 0.0 in the water material.

The result should somehow look as shown in the screenshot below: The directional lighting fades out starting at the *Default Water Surface Position* (absolute in world space) over the range defined by *Directional Lighting Fade Range* (relative value).

This happens automatically on all deferred rendered materials.

Attenuating sun light below the water surface.
When using deferred rendering we can easily attenuate the light from the sun (the directional light) below the water surface.

Please note: All opaque/deferred materials automatically get shaded according to their position relative to the water surface. Ambient lighting currently is not handled so you still have some light even at the very bottom. Local spot lights (these tiny little spots next to the big submarine…) just lit the surfaces as they would normally do.

In order to set up the underwater fog lighting and the *Fog Lighting Fade Range* you will have to enter play mode and dive into your water volume as it does not render in edit mode.

Fog lighting will be darkened according to the camera's position relative to the *Water Surface Position.*

Ambient lighting currently is not handled. Theoretically it could be done like the fog lighting – but as there are a lot of assets out there changing ambient lighting like any Time of Day solution i skipped this.

Deep water lighting and transparent materials

As mentioned earlier the attenuation of the directional lighting below the water surface is handled automatically on all deferred materials. Forward materials like materials using a custom lighting function (skin, hair, car paint) and transparent materials will not pick up the lighting. You have to assign special shaders or tweak the existing ones to make it work.

Lux Water ships with a small collection of tweaked shaders like two general purpose transparent shaders, alpha blend and additive particle shaders and – as it looks nice – a volumetric light beam shader. Their lighting will be driven by global shader variables set by the *Under Water Rendering* script.

Underwater transparent materials

Underwater transparent materials do not show up if the camera is above the water surface as they do render at queue = 3000 and therefore are occluded by the water surface which uses 2999 and most likely writes to depth.

If they were set to queue = 2998 they would be drawn before the water plane but be more or less invisible as they do not write to depth and therefore underwater fog and absorption would make them vanish.

Furthermore the shaders themselves already add fog and absorption. So these would double up…

For this reason you have the possibility to smoothly fade in transparent underwater materials according to the camera's position below the water surface height.

The shaders

Lux Water comes with two transparent shaders: the "Lux Water/Underwater/Transparent Premultiply" and "Lux Water/Underwater/Transparent Blend" shader.

Premultiply shader Use this (mostly) on transparent geometry like glas. It offers full PBS lighting including reflections on the transparent parts.
You may use this shader on particles as well but should enable particle shading and assign a final alpha texture. Examples are the "Underwater Bubbles" in the "LuxWater Water Volume Deep Water Demo".

Blend shader This shader is suitable for e.g. particles which need normal mapping and full lighting options.
Examples are the "Black smokers" in the "LuxWater Water Volume Deep Water Demo".

Shader Inputs

Color Color multiplier

Albedo (RGB) Alpha (A) The albedo texture in RGB and transparency in alpha.

Particle Options

> **Enable Particle Shading** If checked the shader will add soft particle blending (like built in particle shaders)
>
> **Final Alpha** (only available and needed in the premultiply shader) Premultiplied alpha blending usually would add specular reflections regardless of the given transparency as needed e.g. by glass. This however would reveal the particle quads. So adding a final alpha mask here will make sure that reflections are masked out properly too.
> *Examples are the "Underwater Bubbles" in the "LuxWater Water Volume Deep Water Demo".*

Specular Gloss

Enable Spec Gloss Map If checked the shader will sample the assigned Specular Gloss map.

Specular (RGB) Smoothness (A) The combined Specular Gloss map.

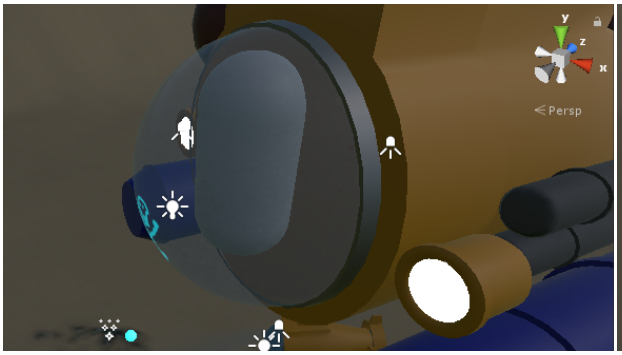
Smoothness Lets you set smoothness just from the slider input. In case “Enable Spec Gloss Map” is checked it acts as a multiplier on the smoothness sampled from texture input.

Specular The specular color in case “Enable Spec Gloss Map” is unchecked.

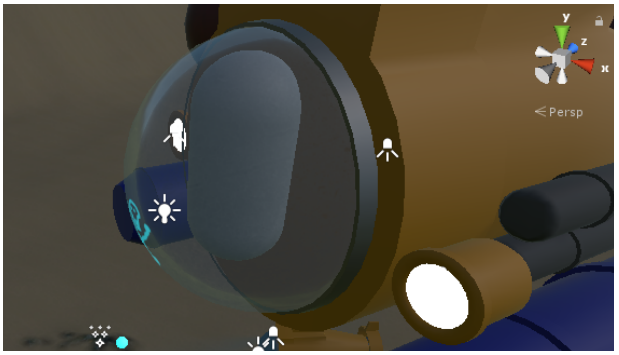
Use proper dielectric Fresnel Unity’s built in fresnel calculation is based on a dielectric/air or conductor/air interface. However in our case (underwater rendering) we have to deal with a dielectric/water or conductor/water interface. So if checked the shader will calculate the proper but more expensive dielectric/water fresnel.

Proper dielectric Fresnel is only needed on materials which use a pretty dark specular color (< RGB (44, 44, 44)) as otherwise it is barely noticable.

Colored fresnel reflections like from copper are not supported as they are way too expensive. But they are not really needed anyway as the brighter the specular color gets (metals) the less difference the both fresnel functions will produce.



Regular Fresnel
Regular fresnel driven ambient reflections. Specular color is 24,24,24.



Proper dielectric Fresnel
Proper fresnel driven reflection for the same specular color. Please note the pretty harsh edge between “no reflections” and “total inner reflections”.

Normal

Enable Normal Map if checked the normal map will be sampled by the shader.

Normal Map A regular normal map.

Scale Scale factor for the sampled normal.

Emission

Enable Emission Map If checked the shader will sample the assigned emission map.

Emission The emission map.

Color Color modifier for the emission as sampled from the emission map.

Caustics

Enable Caustics If checked the shader adds caustics which fit the caustics as added by the underwater post shader. Only needed on materials you use close to the water surface.

Fade

Watersurf Distance Fade Distance to the water surface over which the shader shall fade in.

Underwater light beams

The underwater light beams use a simple shader derived from Epic's light beam shader. Follow the link to find out further details.

As they are rendered using the transparent render queue and path they will more or less be fully hidden by water fog and absorption when the camera is outside the water volume. For this reason the beams will fade in when the camera is below the water surface.

When the camera is below the water surface the shader will calculate under water fog and absorption based on the global values passed from the *LuxWater_UnderWaterRendering* script.

The mesh geometry for the cone has to be double sided.

Shader Inputs

Fall Off (G) Light fall off mask along the beam in the green color channel. Red and blue should be set to 0 (black) to improve the texture quality.

Spot Mask (G) Light fall off masks perpendicular to the beam in the green color channel. Red and blue should be set to 0 (black) to improve the texture quality.

Detail Noise

Enable detail noise If checked the shader wil sample the assigned Detail Noise Texture two times.

Detail Noise (G) Detail noise texture. Noise in the green color channel. Red and blue should be set to 0 (black) to improve the texture quality.

Strength Lets you specify the strength of the detail noise.

Scroll Speed 1:(XY) 2:(ZW) Determines the scroll speed of the two texture samples. XY controls the scroll speed for the first sample whereas ZW controls the scroll speed for the second sample.

Fog Density Lets you lower the density of underwater fog.

Cone Width and **Spot Mask Intensity** Use these params to shape your light beam and hide sharp and ugly edges.

Watersurf Distance Fade Distance between Camera and water surface over which the beam shall fade in.

Camera Distance Fade Distance to camera at which the beams shall start to fade out.

Soft Edge Factor acts like Unity's Soft Particle Factor.

Water Projectors

Water Projectors allow you to project additional foam and custom normals on top of the water surface in order to create e.g. foam trails or water ripple effects.

You can use water projectors also to add some kind of interaction with the water surface (like splashes) or simply spice up the overall look.

Projectors are rendered into two different render textures (foam and normal buffer) using the current camera's position and perspective – which means they are rendered in screen space. When the water surface is rendered later in the frame these buffers get projected on top of the water surface.

This screen space projection needs you to carefully align your projectors (particle systems, planes) with your water surface. Usually they should be laid out as horizontal billboards or flat planes and their y position should match the water's y position pretty closely. The more accurate this is done the less artifacts you will get.

Please note: Projectors aren't rendered properly in scene view but only in game view.

Adding water projectors

To get you up and running you have to:

1. Add the *LuxWater_ProjectorRenderer.cs* script to your camera.
2. Simply drag one of the provided "Demos -> Prefabs -> Water Projectors" prefabs into your scene and position them properly.
3. Enter playmode and explore the final result.
4. Create your own water projectors according to the information provided below.

Under the hood

The *LuxWater_ProjectorRenderer.cs* script actually drives the rendering of all water projectors.

Each water projector needs the *LuxWater_Projector.cs* assigned. This script will register the related projector to the *LuxWater_ProjectorRenderer.cs* script according to the chosen *Type* (Foam Projector or Normal Projector), which then will take care of proper rendering (including that the renderer won't be rendered by the default camera in playmode).

Foam Projectors

Foam projectors let you add foam either to special places where the default screen spaced foam just is not enough or dynamically in order to add e.g. foam trails.

Regardless of if you use a simple plane or a particle system, foam projectors need to have the *LuxWater_Projector.cs* script attached to them (*Type* has to be set to *Foam Projector*) and should always use the *Lux Water/Projectors/Foam Projector* shader.

Please note: Foam projectors need you to check *Enable Foam* in the water surface material. Otherwise they will not show up.

Foam Projector Shader

All foam projectors should use the *Lux Water/Projectors/Foam Projector* shader.

Foam projectors simply write a grayscale mask into the foam buffer. The projected foam will then pick up the color from the foam color as specified in the water surface shader.

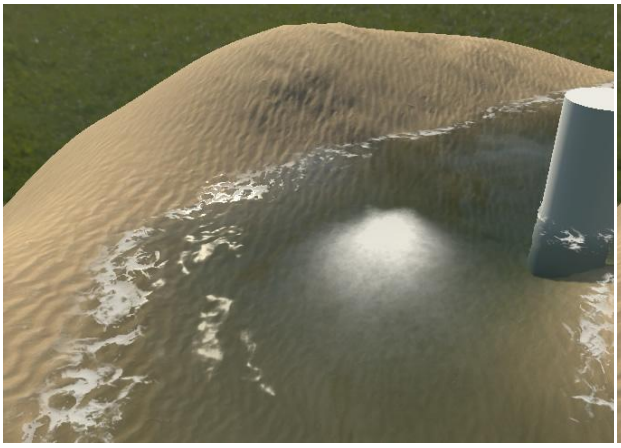
You may choose between two different **Overlay modes:**

Simple Will add the foam only to the foam's alpha value in the water surface shader. This foam will not pick up the foam's normals nor will it be influenced by the foam texture assigned to the water surface shader.

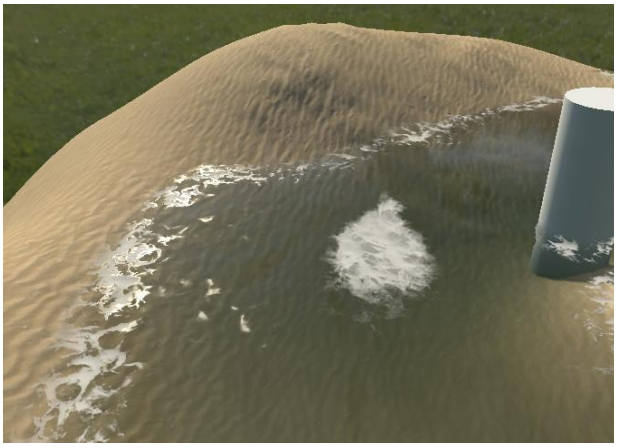
If this mode is selected the foam projector will be tinted green in the editor and debug mode.

Foam This mode will add the foam buffer to the procedurally generated foam so that it picks up all its properties like normals, animation speed etc.

If this mode is selected the projector will be tinted red in the editor and debug mode.



Overlay mode: Simple
The projected foam looks more like a classical particle system.



Overlay mode: Foam
Here the projected foam matches the procedurally generated foam as on the shoreline.

Please note: When using *Foam* instead of *Simple* usually you will get way less visible foam as the shader combines the mask from the projector and the mask from the foam defined in the water surface shader. You may have to tweak the emission rate (in case you use a particle system) or opacity parameter in the material to get the desired result.

Inputs

ZTest Should be set to *Disabled* (default), which means that the shader will not perform any depth testing. You may set it to *LessEqual* in the editor while positioning your normal projectors to get a better sense for its depth tho.

Culling Should be set to *Off* so that the projector will be visible from above and below the water surface. Change this only on purpose.

Blending

> *SrcFactor* and *DstFactor* drive the blending in the foam buffer. By default they are set to match the standard particle add blending (SrcAlpha/One). But you may play around with other blend modes of course.

Opacity Lets you tweak the opacity without having to tweak the texture.

Mask (R) The foam mask is stored in the red color channel. *Please have a look at the Foam Projector Textures section to find out more.*

Overlay Mode Lets you choose between Simple and Foam. See above to find out more.

Foam Projector Textures

Foam projector textures are simple grayscale images. As we only need the red color channel the texture **Format** should be set to “R compressed BC4” (desktop) in the import settings.

sRGB (Color Texture) may or may not be checked.

Normal Projectors

Normal projectors will output *“some kind of”* tangent space normal to the normal buffer, which then gets sampled, combined with the given water normals and finally transferred to world space in the water surface shader.

The result is not 100% accurate but looks pretty convincing and is fast to compute, so I stuck with the current approach.

Normal projectors must use the *Lux Water/Projectors/Normal Projector* shader, which currently only supports additively blended particles/normals.

Please note: In order to create smooth edges between water and projected normal it is crucial that the normal texture applied to the normal projector fades out smoothly towards RGB = 128, 128, 255.

Please note: Normal Projectors rely on RenderTextureFormat.ARGBHalf which might not be supported on all platforms. In case you encounter any problems please come back to me.

Normal projectors also need the *LuxWater_Projector.cs* script attached to them (*Type* must be set to *Normal Projector*).

Normal Projector Shader

All normal projectors must use the *Lux Water/Projectors/Normal Projector* shader, which currently only supports additively blended particles/normals.

Inputs

ZTest Should be set to *Disabled* (default), which means that the shader will not perform any depth testing. You may set it to *LessEqual* in the editor while positioning your normal projectors to get a better sense for its depth tho.

Culling Should be set to *Off* so that the projector will be visible from above and below the water surface. Change this only on purpose.

Normal (RG) Mask (B) Height (A) Holds the texture which contains:
- the red and green color channel of a regular normal map in red and blue.
- in blue you may add a mask texture in case you spot harsh edges concerning the final water normals. If you do not need a mask, simply set blue to pure white.
- In alpha you may add a height map. This is used (and needed) by the tessellation shader in case you want any extrusion.

Please have a look at the Normal Projector Textures section to find out more.

Normal Strength Lets you scale the normal.

Normal Projector Textures

As the normal texture which is used by the normal projector shader should fade out smoothly to the normals of the water it is mandatory that at least the outer edges have a color value of RG = 128, 128.

Texture Import settings

sRGB (Color Texture) *Uncheck* this option as we do not want any Gamma correction.

Filter Mode might be set to *Trilinear*, as we deal with a normal here.

Compression You should set it to *Height Quality* – if not even to *None*.

Please have a look at the provided demo content to find out more.

Particle Systems

I assume that most Water Projectors just will be a particle system – although you can of course also use custom meshes. So this section will give some general tips on how to setup particle systems.

Position

Make sure that the particle system is placed slightly above the water's y position. An offset of 0.01 just should be fine.

Renderer

Render Mode Horizontal Billboard

Normal Direction 1 *(needed by normal projectors)*

Material Make sure that your material either uses the *Lux Water/Projectors/Foam Projector* or the *Lux Water/Projectors/Normal Projector* shader.

Max Particle Size You may have to raise this value in order to avoid particles from being scaled down.

Custom Vertex Streams As we might deal with normals here we need some custom vertex streams:

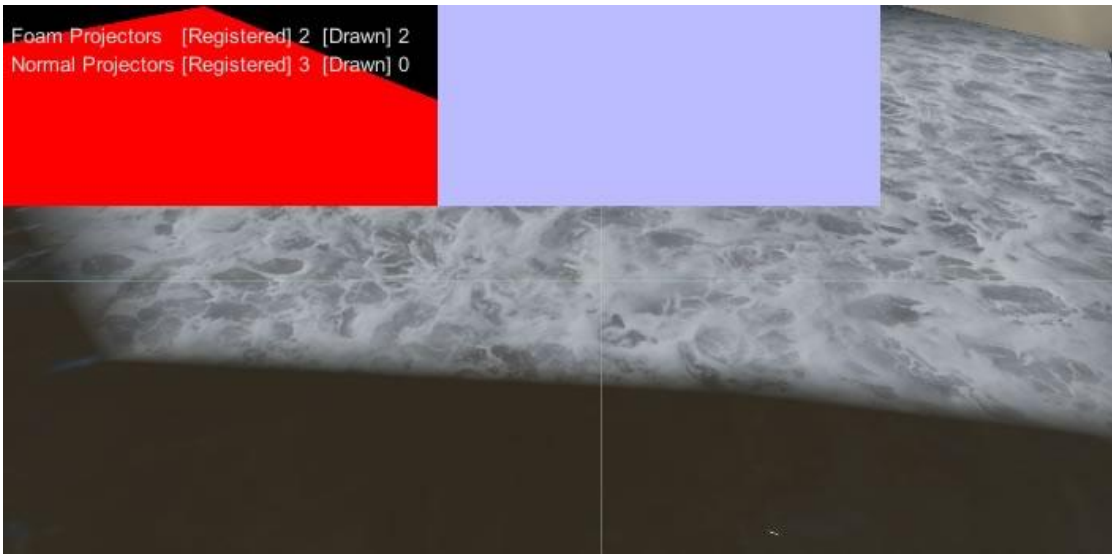
- Position
- Normal
- Color
- UV
- Tangent *(needed by normal projectors)*

Water Projectors and Gerstner Waves

Using a screen space projection will give you almost perfectly fitting results in case the water is just a flat surface. When using Gerstner waves however things get a bit more complicated.

Actually the current approach displaces the projection according to the displacement from the gerstner waves and makes foam actually swim on top of the surface, which is nice. Nevertheless gerstner waves introduces some more artifacts like:

- *All* projected foam and normals will be displaced by gerstner waves. This might lead to e.g. foam trails getting slightly disconnected from the emitter = boat.
- If the displaced screen space coordinates get out of the range of the area the render texture covers foam and normals would be stretched. In order to address this the water shader will simply fade out projected foam and normals instead of project stretched textures:



The difference between the red shape in the buffer preview and the actually rendered foam on screen is caused by vignetting in order to hide stretched foam as described above.

- If the camera is “inside” the displaced waves, the projection may just go crazy :(
- Attaching projectors to objects which get displaced themselves by gerstner waves will produce artefacts.

You can however avoid most artifacts by using gentle displacement values in the Gerstner Waves Settings.

Water Projectors, moving objects and Gerstner Waves

Please have a look at the “LuxWater Ocean Demo” scene which contains 4 different approaches to combine projectors, moving objects and strong Gerstner waves.

1. **Parented Particles** Here the Normal Projector is parented under the game object which uses the simple “LuxWater_SetToGerstnerHeight.cs” demo script. As the object gets displaced by the script the particle system will be too. On top of this the particles will be displaced in screen space by the water shader. This leads to particles just floating around...
2. **Unparented Particles** Here the Normal Projector is not parented under the sphere, but vice versa, so it has a stable position. The displacement of the particles solely is calculated within the water shader. As in this example displacement from script and displacement from shader do not sum up, particles look quite ok. Nevertheless there is a small delta between the position of the particles and the position of the sphere as the sphere uses damping which the particles do not use (not supported by the shader).

3. **Following Particles - Simulation on Local Space** You may of course move and animate your objects. In this case you make the particles follow your object which here is done using “LuxWater_SetToGerstnerHeight.cs” demo script: This moves the projector to the “original” position of the object which does not contain the displacement from the gerstner waves.
4. **Following Particles - Simulation in World Space** Using World Space for the particle animation lets you easily create trails.

Takeaways

- Do not parent Foam and Normal Projectors under objects which are displaced according to the gerstner displacement of the water waves.
- Parent the objects under the projectors.
- Or use script to make the projectors follow the objects.

The script components

LuxWater_ProjectorRenderer.cs

This script must be attached to your camera. It handles the drawing of both the foam and normal buffer which later will be projected on top of the water surface to create the desired effect. It will look through all registered projectors and check their visibility using an AABB frustum check. If a registered projector is visible the script will draw it to the corresponding buffer.

Inputs

Foam Buffer Resolution Lets you downsize the resolution of the normal buffer in order to save fill rate.

Normal Buffer Resolution Lets you downsize the resolution of the normal buffer in order to save fill rate.

Debug

> **Debug Foam Buffer** If checked a preview of the foam buffer will be outputted to the scene and the game view in playmode. In order to see it in game view you have to check “Gizmos” in the upper right corner of the game view window.
> *Projectors using the simple overlay mode will show up in green, projectors using the foam overlay mode will show up in red.*
>
> **Debug Normal Buffer** If checked a preview of the normal buffer will be outputted to the scene and the game view in playmode. In order to see it in game view you have to check “Gizmos” in the upper right corner of the game view window.
>
> **Debug Stats** If checked the number of registered and actually drawn foam and normal projectors will be printed to scene and game view.

LuxWater_Projector.cs

You have to attach this script to all particle systems or game objects/renderers which shall render into the foam or normal buffer. If the game object which it is attached to does not have

a renderer it will simply do nothing. Otherwise it registers itself to the *LuxWater_ProjectorRenderer.cs* and disables the renderer in playmode.

Inputs

Type Choose between *Foam Projector* and *Normal Projector* according to the kind of projector you want.

Please note: You have to manually assign a proper material using the *Lux Water/Projectors/Foam Projector* or the *Lux Water/Projectors/Normal Projector* shader.

Lux Water Utils

LuxWater_Utils.cs

This script contains a bunch of helper functions which – right now – allows you to sample the displacement of the water surface caused by gerstner waves in order to e.g. make objects follow the movement of the waves.

These functions are used in the *LuxWater Gerstner Displacement Demo* to make the spheres sit on top of the water surface.

struct GersterWavesDescription

A struct which holds all parameters to describe the gerstner waves for a given material.

GetGersterWavesDescription (ref GersterWavesDescription Description, Material WaterMaterial)

Fills the passed GersterWavesDescription with the parameters from the passed material.

Vector3 GetGestnerDisplacement (Vector3 WorldPosition, GersterWavesDescription Description, float TimeOffset)

Returns the displacement as Vector3 of a water surface (described by the GerstnerWavesDescription) at a given world position according to the current time + the passed TimeOffset.

LuxWater_SetToGerstnerHeight.cs

In order to get a better idea of how it works you should have a look at the *LuxWater_SetToGerstnerHeight.cs*, which is used to place the spheres according to the wave displacement.

Inputs

Water Material You have to assign the water material here used by the water mesh the objects should follow.

Damping Lets you define the strength of the displacement along each axis.

Time Offset Lets you calculate the displacement at a different time than the current time. Usually you would use negative values here in order to create an effect of the inertia of masses.

Update Water Material Per Frame If checked the Gerstner Waves settings of the assigned material will be read each frame (expensive…). Only check this if you change the material settings over time.

Add circle Anim If checked the script will add a circular movement using Radius and Speed. For testing purposes.

Under the hood

The script uses the functions and structs defined by LuxWater_Utils.cs.
It declares:

```
private LuxWaterUtils.GersterWavesDescription Description;
```

in order to create the needed struct which holds all information about the gerstner waves.

Then on *Start* it retrieves the Gerstner Waves settings of the assigned material:

```
LuxWaterUtils.GetGersterWavesDescription(ref Description, WaterMaterial);
```

Then it calculates the Offset of the Gerstner Waves at the given location and time in *Update* using:

```
Vector3 Offset = LuxWaterUtils.GetGestnerDisplacement(trans.position,
Description, TimeOffset);
```

The Offset then finally is used to reposition the object.

LuxWater_SetMeshBounds.cs

When using Gerstner Waves vertices might get heavily displaced so they leave the bounds of the original mesh. Therefore Unity might cull the mesh altho it should still be visible. In case you come across water meshes that suddenly disappear you may add this script and scale up the mesh's bounds.

The script works on the sharedMesh. So if you use several water tiles to create a huge water surface you have to attach the script only to one of the tiles.
Please have a look at the LuxWater Fjord Demo.

In the editor you will see a red wired box which represents the bounds. Just make sure that the selection wire preview never leaves these bounds.

Inputs

Expand_XZ Expand the bounds by increasing its size by the given amount along the xz axis.

Expand_Y Expand the bounds by increasing its size by the given amount along the y axis.

Creating large water surfaces and Oceans

Lux Water originally has not been created to render large or even infinite water surfaces. It does not use a projected grid or geomipmaps to automatically create the water surface but works on custom provided arbitrary geometry like oval lakes or curved rivers.

Nevertheless you can create large water surfaces using multiple water tiles.

The “LuxWater Ocean Demo” shows how you could create a wide water surface using several water planes and a sliding water volume.

It also shows how you should set up particle projectors in case you use strong gerstner waves as described here.

Sliding water volumes

The problem

As water volumes always have to be fully within the drawing range of your camera (far clip plane) you can not use one of the provided, simple water surface / volume meshes and scale it to e.g. 9000 x 9000 meters in case your camera’s far clipping plane is set to 3000. This would result in holes in the water mask giving you shading artifacts like shown on the right.

Water volume being clipped according to the far clipping plane of the camera. As the bottom of the volume is too far away you will get a hole in the water mask.

The solution

Let’s stay with the 9000 x 9000 meters example and a far clipping plane of 3000 meters. Then your volume’s max size would be 1732 x 1732 x 1732 meters according to Pythagoras, so you can look from one corner diagonally to the opposite one. In order to keep it simple here we raise the far clipping plane to 3465 meters so we can use planes of 2000 x 2000 meters :)

So we will create the water surface using several water planes e.g. 5 x 5 planes each 2000 x 2000 meters in size.

One might think that adding 5 x 5 volumes would do the trick, but unfortunately Lux Water does not support a seamless transition from one volume to the next. Furthermore doing so would mean that we had to draw up to 4 volumes to create the mask in case the camera is close to an edge – which sounds quite expensive.

So we use a sliding volume instead.

The sliding volume has a size matching the size of the water planes: 2000 x 2000 x 2000 meters and gets moved so that it is always centered on the camera. The important detail when it comes to gerstner waves: We have to make sure that the vertices of the volume always fall on vertices of the water surfaces to avoid gaps. So we can not simply center the volume but have to do it in steps.

Example

The provided **Ocean Demo** uses 5 x 5 water tiles each originally 500 x 500 meters in size. As the Ocean_5x5 prefab is scaled by 2 it covers an area of 5000 x 5000 meters – which would be the reachable area within the game’s level.

The used water surface mesh (and volume as well) has 50 subdivisions along each axis so our step size (or grid size as called in the script) is 10 meters (the scale of 2 will be factored in

automatically). With this info provided the UnderwaterRendering script is able to center the volume and make it snap to the grid formed by the vertices of the water surface planes.

We only have one single volume and collider – attached to the first water surface in the "Ocean_5x5" prefab. The collider is "static" and fills the entire area of 5000 x 5000 meters while our volume of 1000 x 1000 x 1000 meters gets moved.

According to the final volume's size the camera far clipping plane must be greater than 1732 meters (sqrt(pow(1000,2) * 3)).

The "Ocean_5x5" prefab is extended by a "skirt" which is made from simple planes using a simplified water material: no foam, no caustics, no pixel snapping, no gerstner waves, no tessellation. Culling is set to "Back".

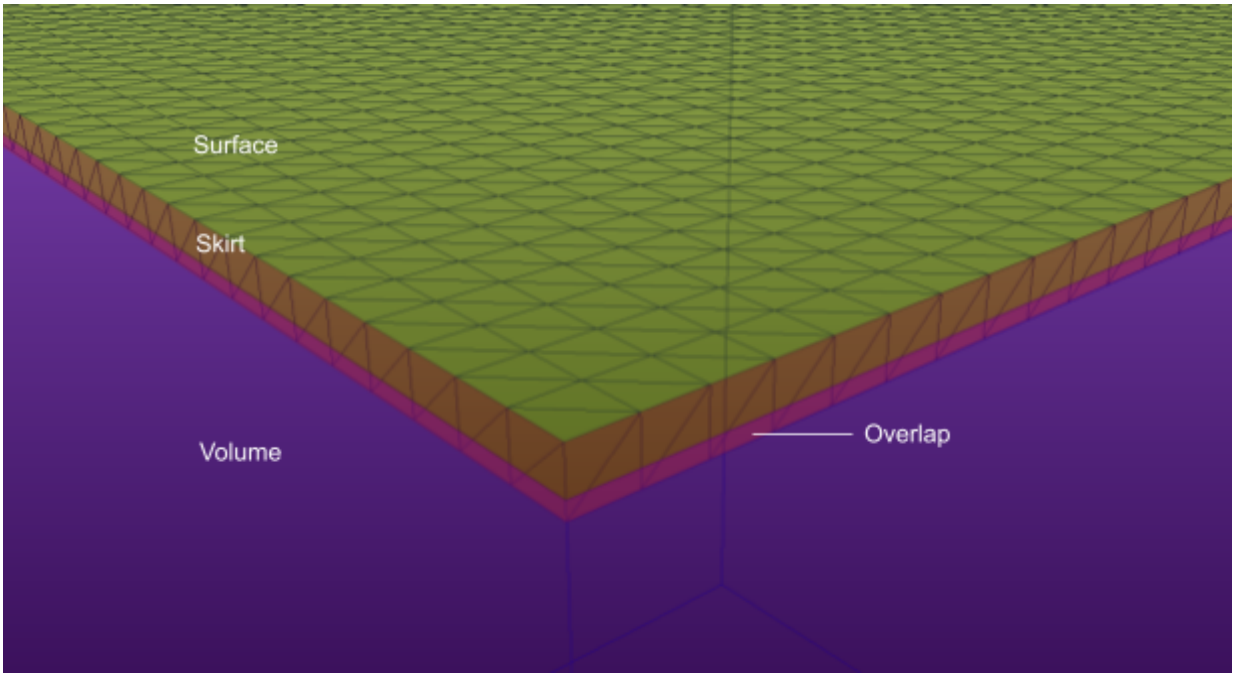
Sliding Water Volume Mesh

Compared to a "regular" water volume mesh – where you would mark the outer vertices using vertex color red to prevent these from being displaced by Gerstner waves the geometry of the mesh and the usage of vertex colors are slightly different.

Geometry

The mesh consists of 3 "parts" stored in 2 submeshes (material groups):

1. The highly tessellated water **surface** (as submesh 2).
2. A small **skirt** of quads as highly tessellated as the surface (in submesh 1).
3. A simple **volume** (in submesh 1).



The **skirt** is needed to fill any gaps between surface and volume when it comes to Gerstner waves and tessellation.

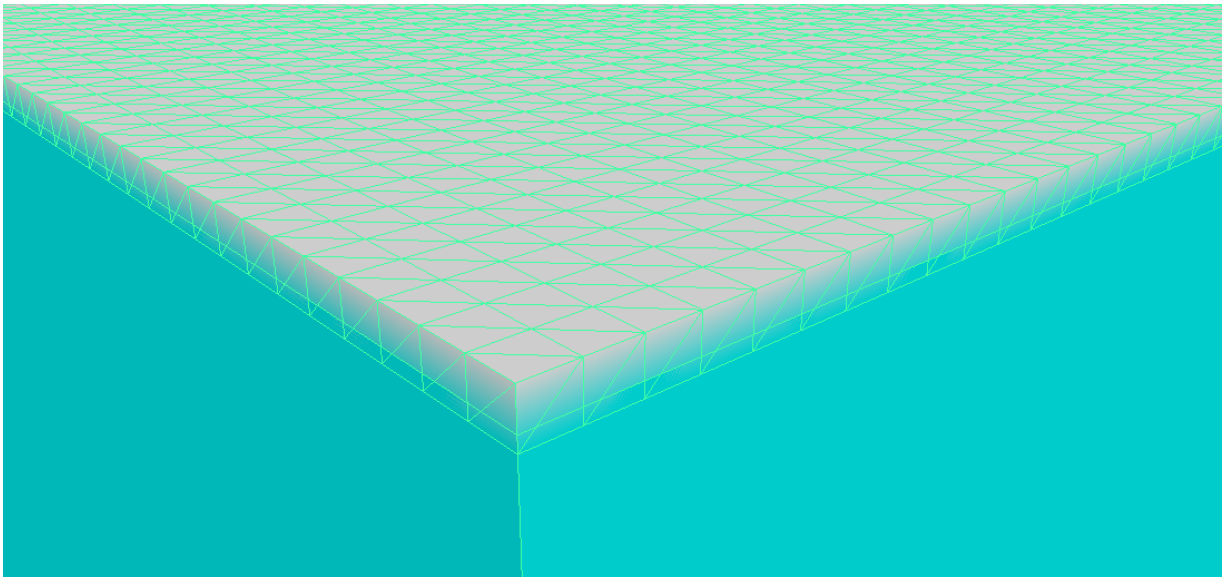
The **volume** slightly overlaps with the skirt.

Vertex Colors

The water **surface** is set entirely to vertex color red = 1. This will result in full tessellation and full displacement.

The **skirt** uses a gradient from vertex color red = 1 at the top to 0 at the bottom. This will result in full tessellation but displacement will be limited to the upper vertices.

All vertices of the **volume** are set to vertex color red = 0. Tessellation will be set to 1 and displacement will be fully suppressed.



Infinite Ocean

Now as you know how to create large water surfaces and volumes by using a sliding water volume and several water tiles it is only a small step to create an infinite ocean:

Simply use the same technique as for the sliding water volume and center all water surfaces on the camera. In order to prevent any hiccups in the animation and displacement of the surfaces we have to do this in steps as well and make the movement snap to the grid formed by the vertices of the water surface planes.

A simple example can be found in the “Lux Water infinite Ocean Demo”. Here the ocean prefab holds the simple “Luxwater_InfiniteOcean” script.

LuxWater_InfiniteOcean.cs

Use this script to move a set of water tiles along with the camera to create an “infinite” ocean. It must be attached to the game object which hold the water planes (like e.g. “Ocean_5x5 prefab”)

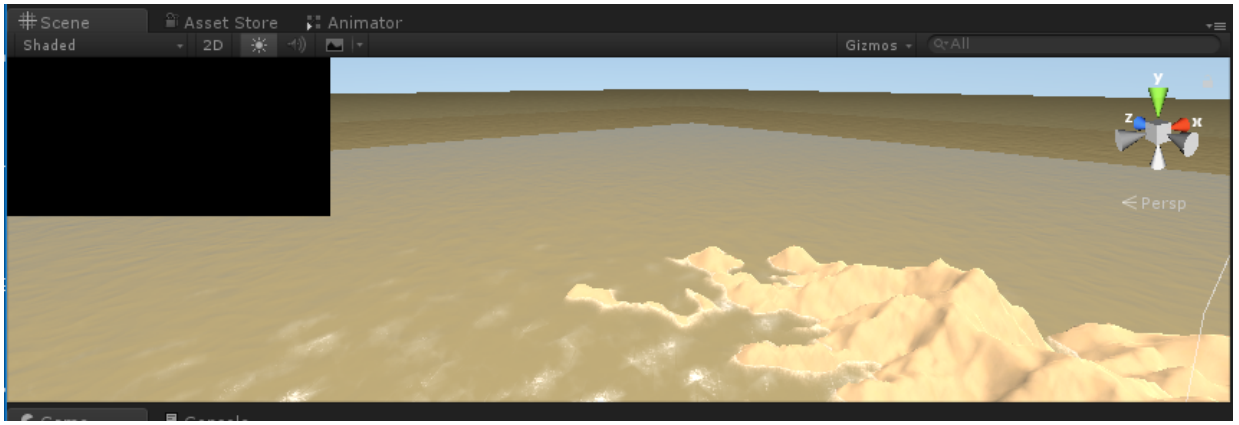
Inputs

Camera Assign your camera. If you leave this field blank the script will look for a camera tagged as “MainCamera”. If it does not find any it will simply return.

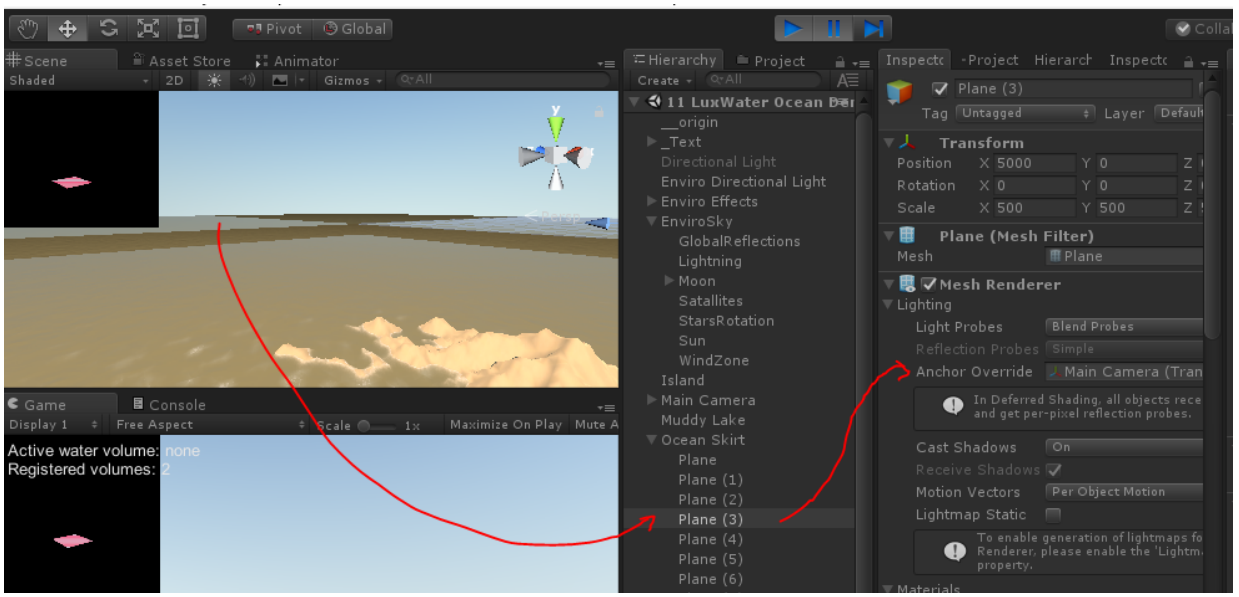
Grid Size The distance between the vertices of the water surface mesh at a size of 1.0. If the game object which holds the water tiles is scaled the script will adjust it automatically. Unlike the “LuxWater_WaterVolume” script this script does not take possible scaling of the water tiles inside the holder into account.

Reflection Probes

When using huge water surfaces made out of several tiles and custom reflection probes (like Enviro does) some of the water tiles may not be covered by the reflection probe’s volume, which leads to harsh edges as far as reflections are concerned:



To come over this you may consider overwriting the water tiles’ anchor which determines how probes are blended:



Metal and Deferred Rendering

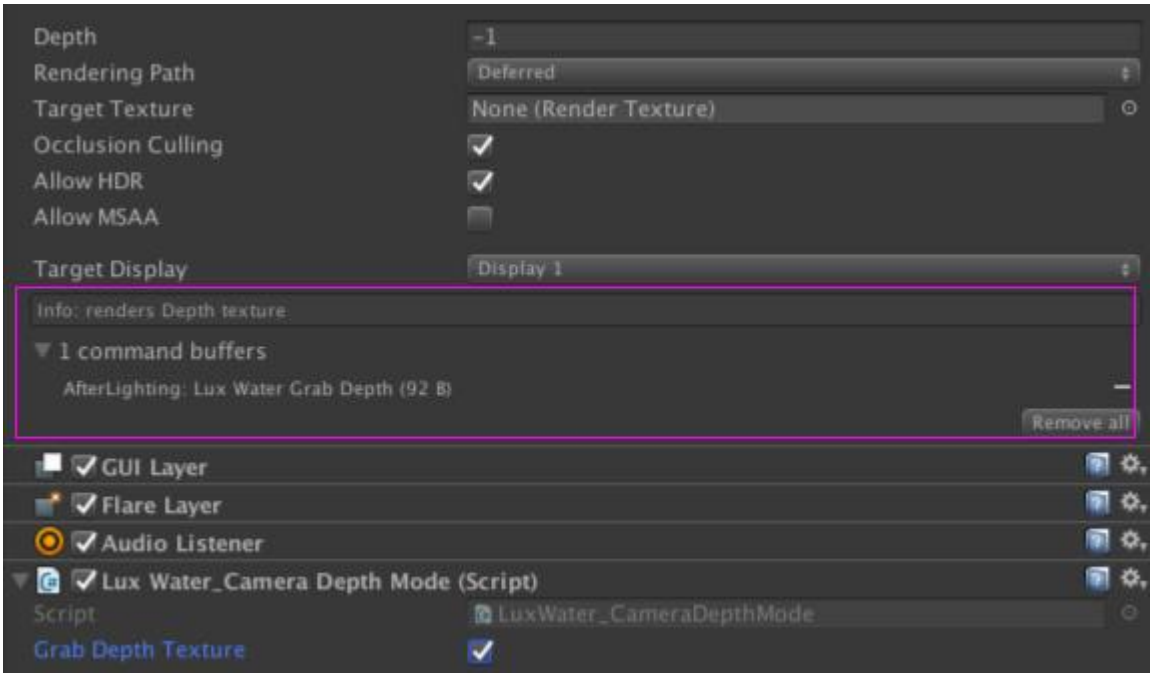
Due to the way Metal handles depth, you can not simply make the shader write to depth when using deferred rendering: Water would simply disappear as Metal reads from and writes to the depth buffer at the same time …

So you can just set **ZWrite** to off in the material inspector. This is fine in case you only use flat planes and you do not use Gerstner Waves.

But if you need proper depth writing then you will have to:

- add the *LuxWater_CameraDepthMode* script to your camera.
- check “Grab Depth Texture”.
- edit the LuxWater Shaders:
    - Find *(you will find it two times)*
      `//#define LUXWATERMETALDEFERRED`
    - and change it to:
      `#define LUXWATERMETALDEFERRED`
    - Then safe the shader.
- If you change your camera to forward you will have to comment `#define LUXWATERMETALDEFERRED` again.

By checking “Grab Depth Texture” you will add a CommandBuffer which grabs the depth texture after deferred lighting is finished and copies it into a custom texture which then is used in the modified shader.



Please note: In case you set your camera projection to orthographic the camera will always render in forward. In this case you will have to revert the shader and comment `#define LUXWATERMETALDEFERRED` again.

Optimizations

While it is quite convenient to be able to turn on and off special features like foam or caustics creating a lot of different shader variants is quite time and – most likely even more important – memory consuming, even in your build.

Do not solely rely on Unity's shader stripping, but manually activate/deactivate features you know you do not use by replacing the `#pragma multi_compile` directives.

In case you want support for water projectors e.g. replace:

```
//      Water projector support
        #pragma multi_compile __ USINGWATERPROJECTORS
```

with:

```
//      Water projector support
        #define USINGWATERPROJECTORS
```

In case you do not want support for projectors simply comment the #pragma:

```
//      Water projector support
        // #pragma multi_compile __ USINGWATERPROJECTORS
```