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 (preview) allow you to get pixel accurate and seamless transitions between above and underwater rendering.

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

Compatibility

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

Although the water shader itself uses forward rendering Lux Water works best in deferred.

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.

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.

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. In case you would like to use linear or simple exponential fog and forward rendering you will have to edit the shader file and comment/uncomment the corresponding defines.

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
```

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. So you may disable it.

Culling Lets you specify which faces will be drawn:

- **Back** Don’t render polygons facing away from the viewer.
- **Front** Don’t render polygons facing towards the viewer.
- **Off** Disables culling – all faces are drawn.

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.

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.

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 und 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.

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 have 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 tho.

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.

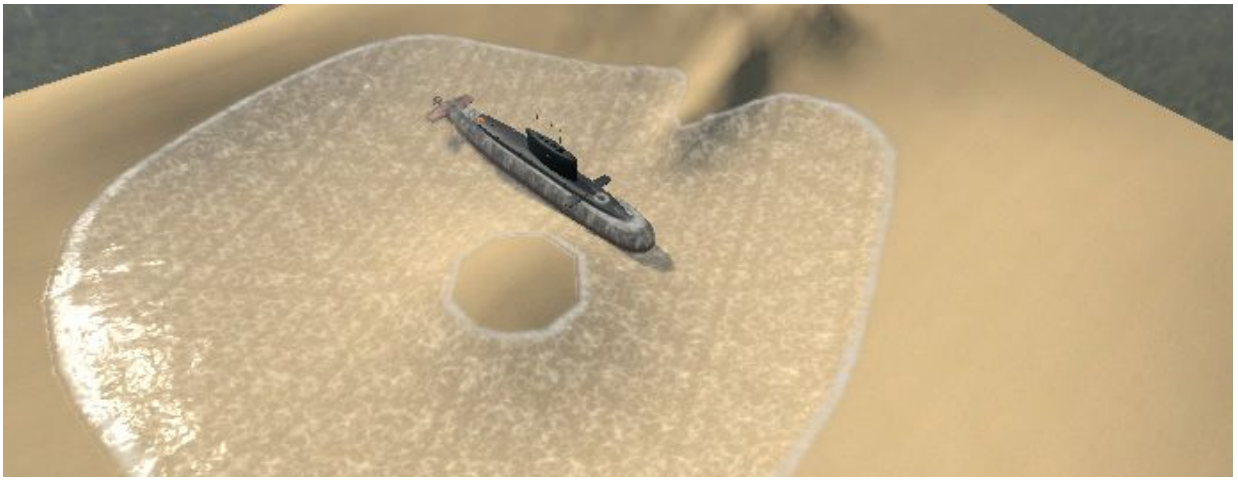
IOR 1.333 Internal reflections more or less cover to complete surface.

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

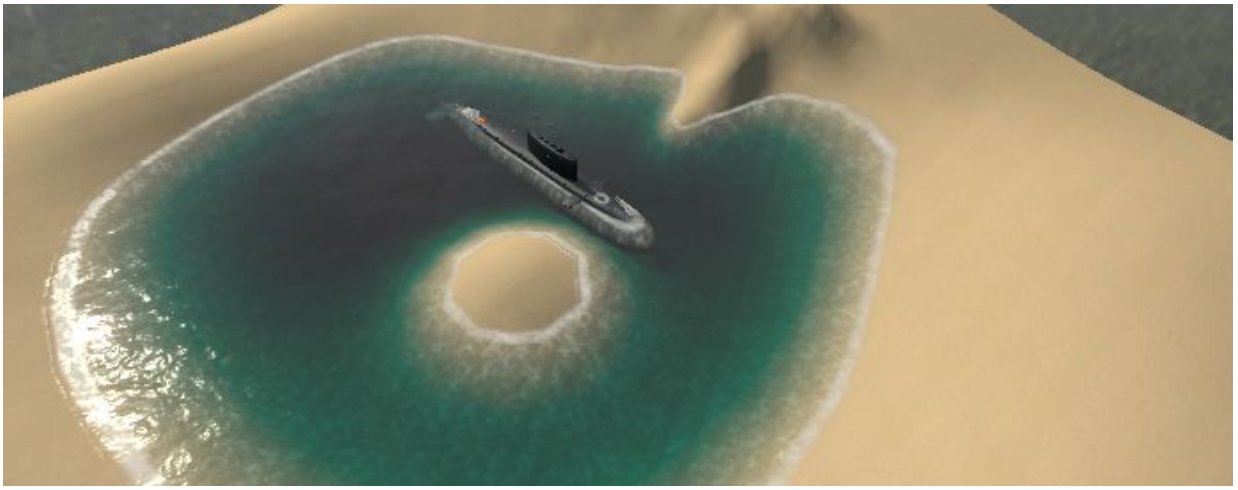
Underwater Reflection Tint As – when looking from below – the reflection will show up the bottom of the reflection cubemap, which most likely will not look like a reflection from below the water surface, you may specify a tint color here.

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 [>]

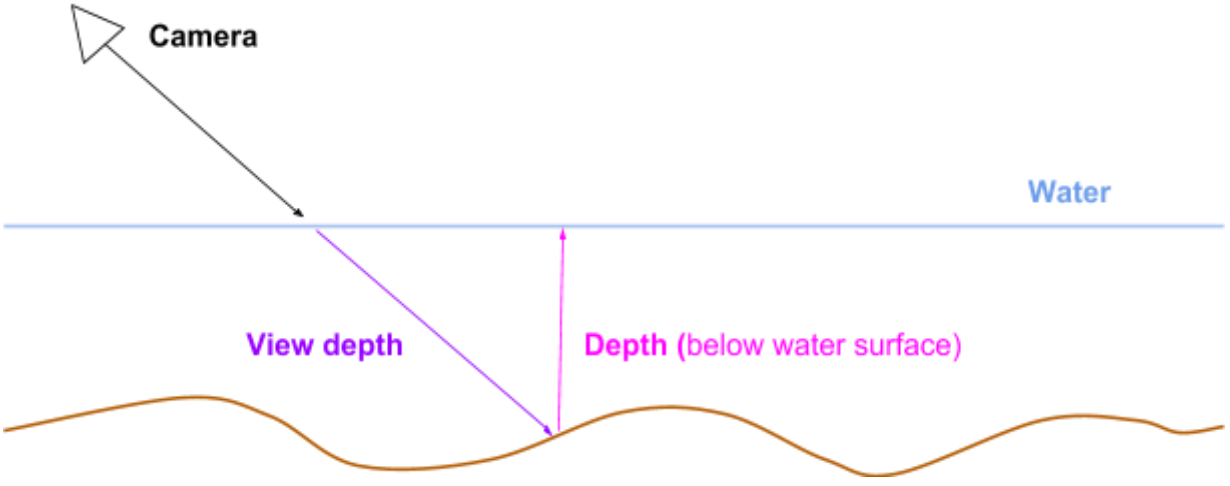
Adding underwater fog lets you tint the water: "*Dissolved and particulate material in water can cause discoloration. Slight discoloration is measured in Hazen units (HU).*[8] *Impurities can be deeply colored as well, for instance dissolved organic compounds called tannins can result in dark brown colors,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

Translucency Color Base color of the scattered light which gets multiplied by the light color. Please note that the translucency color internal is multiplied by 10 aswell in order to allow subsurface scattering even in case you use rather flat normal maps.

Scattering Power Specifies the view dependency of the scattering.

Normal Maps

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

So you may consider using a low res (in order to safe 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)

Fade (Y) Drives the fade between far normal and first detail normal. Smaller values will make the far normal kick in at greater distances.

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 water surface and background along the view ray — which makes fog 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.

Inputs

Enable Foam If checked form will be rendered.

Normal (RGB) Mask (A) The foam texture – where RGB stores the normal map and A 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.

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 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 Determines the size of the dynamically calculated foam border.

Foam caps are only supported if you have enabled Gerstner waves. So all relevant inputs are described there.

Caustics

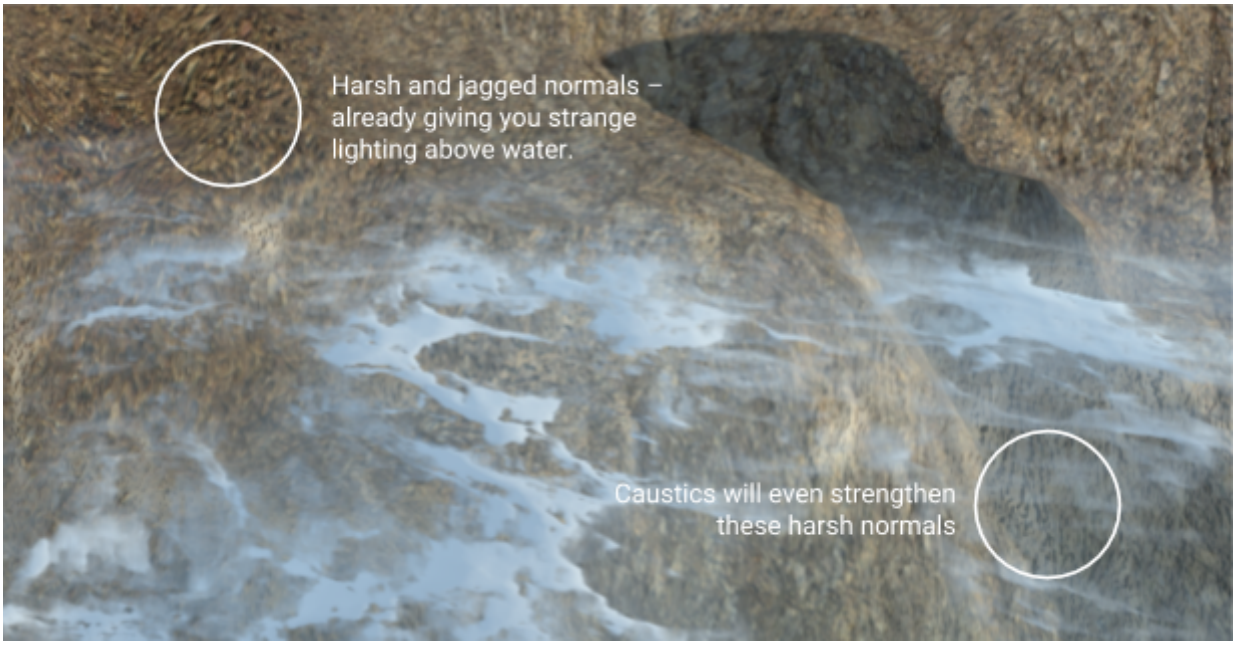
Lux Water does not use any projectors to render caustics but renders them together with the water surface directly to screen taking the depth texture and – in case you use deferred rendering – the deferred normals into account. So unlike other water solutions Lux Water does not have to draw your scene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

Beta: 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. So this will most likely change and become some sort of an image effect in future releases.

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 no be any caustics on steeply angled geometry.

Caustics are not cancelled by any magic number, but fully rely on underwager fog and light absorption.

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 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.

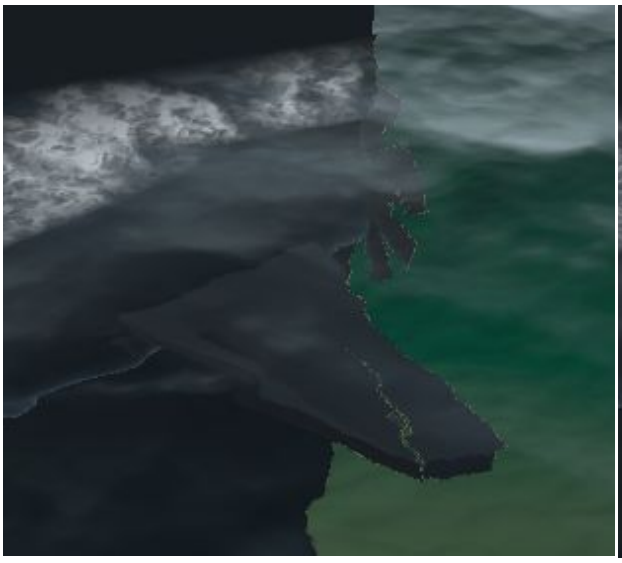
Advanced Options

Enable Snapping If enabled 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.

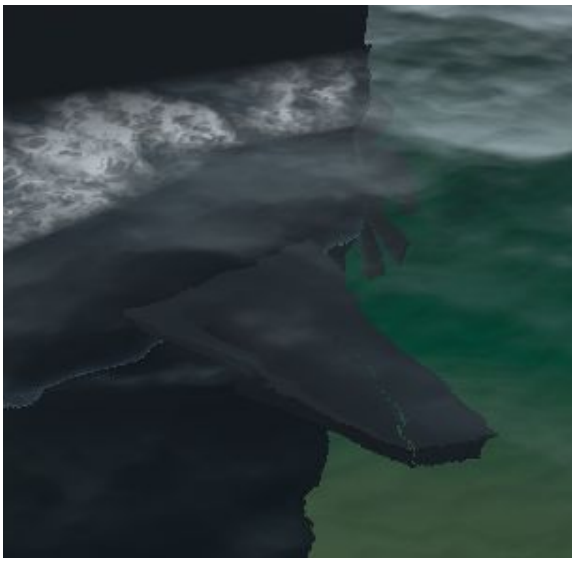
This artifact is barely visible on platforms using a reversed zBuffer like dx11, but it still exists.

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 the terrain (sand), but fog and absorption may be too weak as the depth texture lookup returns a values somewhere between the submarine and the terrain.

Enabling snapping will add some more math tho 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.

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.

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

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.

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:

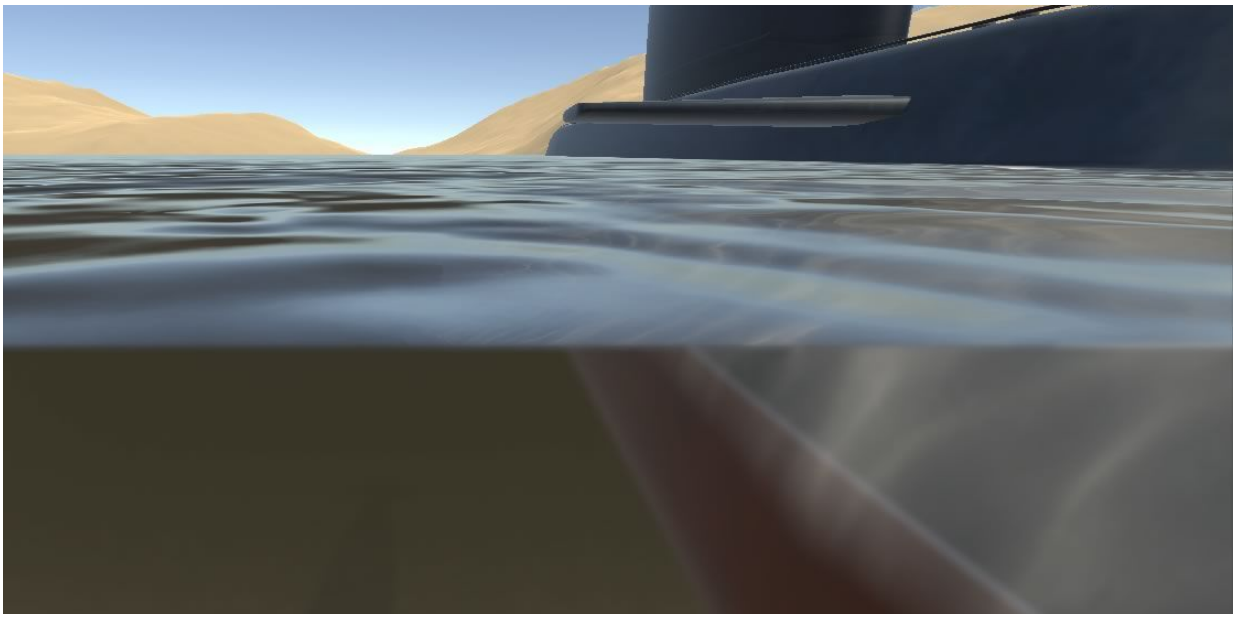
- Create an empty gameobject, attach the “LuxWater_PlanarReflection” script to this object and check **IsMaster**.
- Add your watertiles as child objects and make sure, that they have the “LuxWater_PlanarWaterTile” script attached.
- Make sure that the parent game object has the same y position as the 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.

Water Volumes (Preview)

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 of 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.

Managed transparent Materials In order to get somewhat proper result 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.

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_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 it 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 collider and rigidbody which shall trigger the underwater rendering.

Inputs

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 letter can handle the rendering.

Inputs

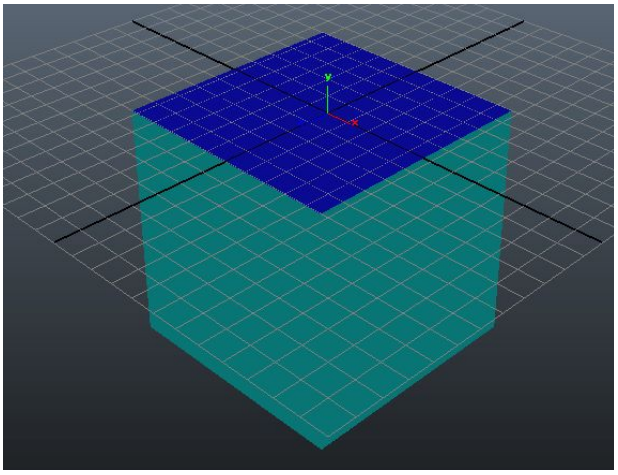
Water Volume Mesh The mesh which is used to render the water mask.

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.

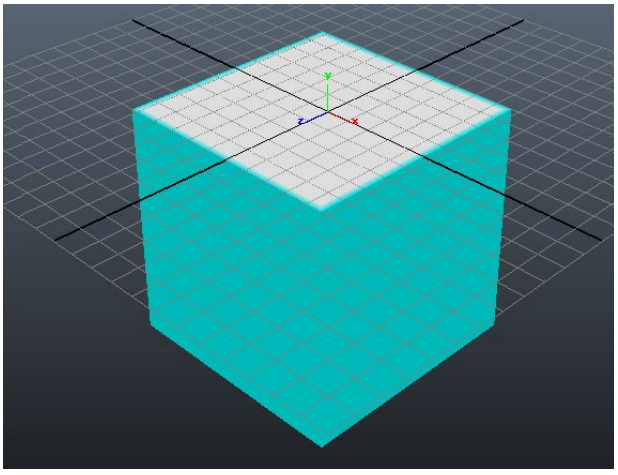
Submeshes and materials

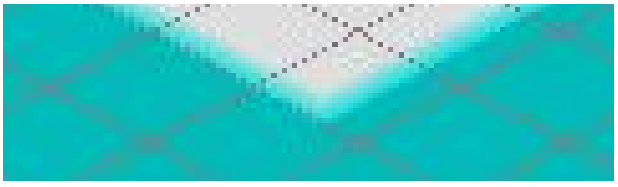
The water volume mesh needs 2 submeshes or materials: 1st one must describe the volume (cyan), 2nd one the actual water surface (blue).



Vertex Colors

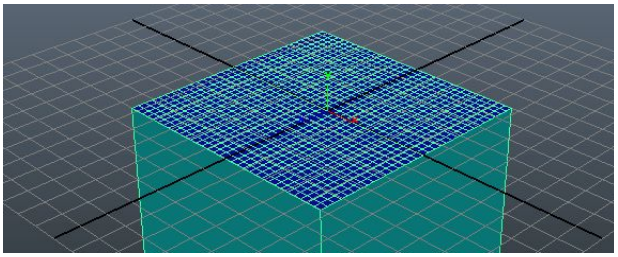
In case you want to use Gerstner Waves you should add vertex colors to the meshes (both the water volume mesh and the mesh used as water plane) in order to prevent the outer edges from getting disconnected. Simply add vertex color red = 0.0 to the outer vertices and you are fine.





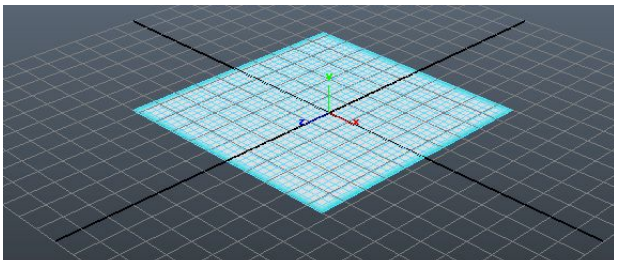
Tessellation

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.

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 crosse 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 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 textur (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.

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 :(

Limitations

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

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

Water Projectors (Preview)

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 needs 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 simple 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.

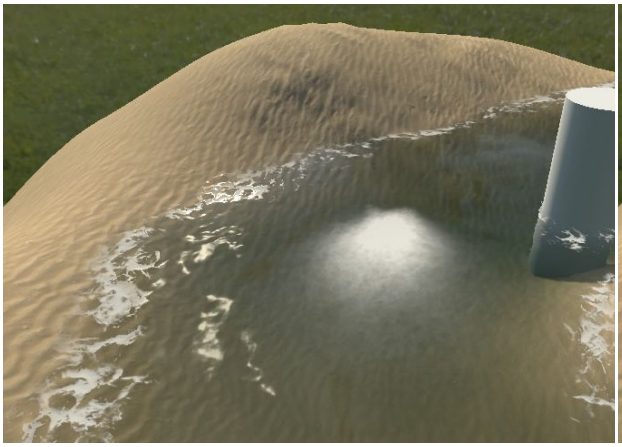
You may choose between two different **Overlay modes:**

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.

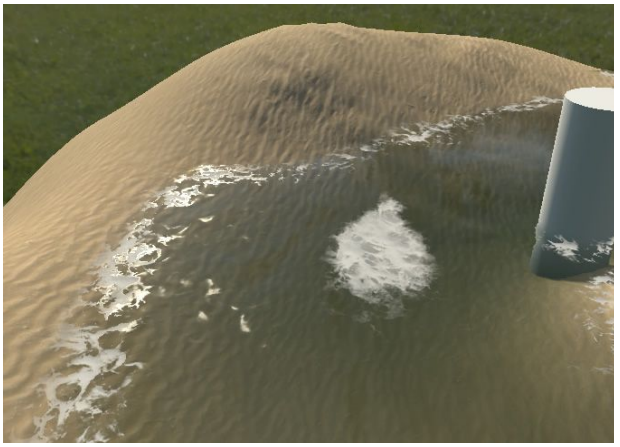
If this mode is selected the foam projector will be tinted green in the editor and debug mode.

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 as 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 Holds the texture which contains a regular normal map. *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 RGB = 128, 128, 255.

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:

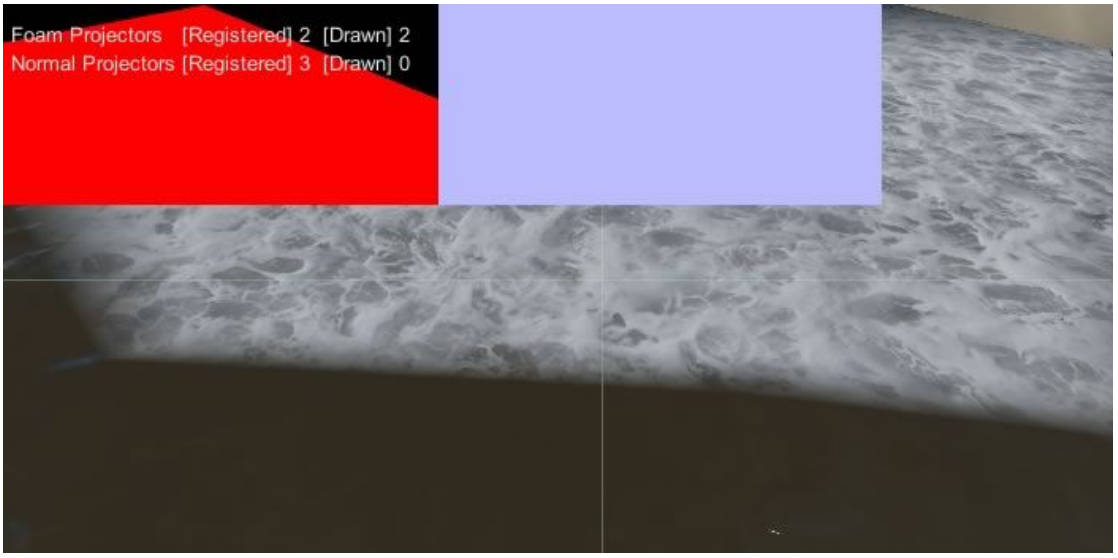
- Position
- Normal
- Color
- UV
- Tangent *(needed by normal projectors)*

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 :(

You can however avoid most artifacts by using gently displacement values.

The script components

LuxWater_ProjectorRenderer.cs

This script must be attached to you 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 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 i 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.

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.

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 Shader:
    - find:
      `//#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.