Lux URP Essentials

Lux URP Essentials provide a growing collection of manually written and optimized HLSL shaders and custom nodes for Shader Graph. It reduces the gap between HDRP and URP by adding missing advanced lighting models and ships with a ton of various rendering features to cover a wide range of use cases.

I tried to somehow cluster the provided shaders using hopefully meaningful categories as listed below — nevertheless i just encourage you to check out *all* included demos to get the full picture.

The new standard Lit
Lit extended and Lit Uber with full stencil support and further functions.

Environment related shaders
Mesh terrain, (interactive) grass, foliage, rock, water, tree creator, height blended parallax terrain and terrain blend shaders.

Advanced Materials and Lighting
Skin, hair, cloth, transmission, clear coat, glass, fuzz, toon lighting, simple flat shading and lit and shadowed particles.

Effects
Fast outlines, rim based animated highlights, hidden surfaces, screen space decals, billboards and simple volumetrics.

Shader Graph
Custom nodes for cloth, transmission, clear coat, glass and toon lighting next to some handy helper nodes like procedural stochastic texturing.

Getting Started
Do not miss the introduction!

FAQ
Please make sure you have a look into the section FAQ where I will collect frequently asked questions as well as tips and tricks and visit the forum.

Compatibility

Lux URP Essentials have been successfully tested in Unity 2019.1.3, 2019.1.6 and 2019.1.10 using LWRP 5.16.1 and Unity 2019.2.0f1 using LWRP 6.9.1

Lux URP Essentials have been tested using Unity 2019.3.1f1 and URP 7.2. and Unity 2019.3.9 using URP 7.4.1 on macOS and Metal, Windows 10 and DX11 and Android (Mali and Adreno GPUs) using Vulcan. A package for URP 7.1.8 is included.

All shaders support the **SRP Batcher.**

All shaders support **Single Pass Stereo rendering** — this includes shaders, which sample the camera depth or opaque texture like water, particles, decals and glass.

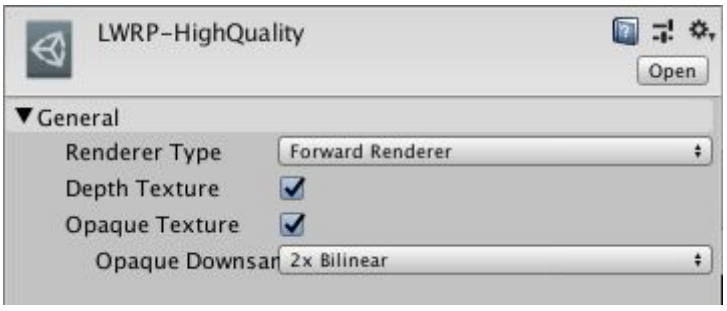
The documentation may still uses "Lux LWRP" here and there. Simply replace it with "Lux URP" when it comes to shader or script names.

In case you import one of the older packages some features described here may not be available as I usually do not backport new features to older versions.

Table of Content

Getting started

- Make sure you have the proper version of the *Core RP Library* and the *Lightweight RP / Universal RP* installed (5.16.1, 6.9.1, 7.1.8, 7.2 or 7.41 - according to the shader package you have imported) in the *Window → Package Manager*.
- Set *Color Space* to *Linear* in *Project Settings → Player → Other Settings*
- Open the *Scriptable Render Pipeline Settings* assigned under *Project Settings → Graphics* and check *Depth Texture* and the *Opaque Texture* as both are needed by e.g. the water or the glass shader. If you check *Opaque Downsample* or not is up to you.



- Lux URP/LWRP Essentials fully support the SRP batcher. So you should enable it in your *Scriptable Render Pipeline Settings*.
- Import the package – if you have not done already – and open the included "Environment Demo" scene (located in *Lux URP/ LWRP Essentials → Demos*).

Shader Overview

The new standard Lit

Lit extended and Lit Uber shader

While the first one simply adds support for stencil options and rim lighting the Uber version comes with a bunch of advanced features such as parallax mapping, specular anti aliasing, horizon occlusion, bent normals and lets you fade out geometry close to the camera.
These shaders are needed in case you want to use features such as the simple outline along with a "regular" material as they allow you to set up the stencil buffer properly.
Details →

Environment related shaders

Mesh Terrain Shader

The mesh terrain shader lets you shade any custom mesh using up to 4 detail textures (albedo, smoothness and normal) mixed together based on an RGB splat map. It usually samples all textures according to the UVs allowing you to texture even cliffs and overhangs without texture stretches – however the first detail layer may use top down projection to enhance mesh blending with objects using the *Top Down Projection shader* or hide texture seams at UV shell borders. Details →

Grass and Foliage Shaders

The grass and foliage shaders allow you to place individual patches of grass and foliage as single game objects within your scene whose bending is globally controlled by a built in wind zone and a custom render texture. In order to speed up rendering all grass and foliage patches in the demos are placed on the built in layers *TransparentFX* or *Water* which get culled at a distance of 30 / 50 meters using the simple LuxLWRP_LayerBasedCulling script . Details →

Top Down Projection Shader

The top down projection shader allows you to add one texture set on top of the regular texture set based on the up direction of the normal in world space to create effects such as moss on rocks, eroded sand or snow.

The projected texture set is sampled in world space and thus gives seamlessly textured surfaces even over multiple objects.

Normals are blended properly in world space using *Reoriented Normal Mapping*. The shader does not need to do expensive triplanar mapping. Details →

Water Shader

The water shader offers fast yet compelling water rendering for rivers or lakes and supports proper refractions, view depth based underwater fog, soft edge blending with the surrounding geometry and dynamically generated edge and slope based foam – both using perspective and orthographic projection. Details →

Tree Creator Shaders

The Lux LWRP/URP tree creator shaders are a port of the original tree creator shaders for the built in render pipeline and allow you to use tree creator trees along with LWRP and URP. Details →

Terrain Shader

The terrain shader supports proper(!) normals, height based blending, parallax extrusion and painted holes (Unity 2019.3. and above only). Details →

Terrain Blend Shader

The terrain blend shader allows you to smoothly blend meshes like cliffs or overhangs with your terrain. Details →

Versatile Blend Shader (experimental)

The versatile blend shader allows you to smoothly blend meshes with *any* other opaque geometry in your scene. Details →

Advanced Materials and Lighting

Skin Shader

The skin shader uses pre-integrated diffuse lighting and lets you create state of the art skin surfaces. Details →

Hair Shader

The hair shader supports anisotropic Kajiya-Kay specular hair lighting and transmission:
Details →

Cloth Shader

The cloth shader supports Charlie Sheen or GGX anisotropic lighting and transmission to cover a wide range of different fabrics. It allows properly lit double sided rendering using VFACE.
Details →

Clear Coat Shader

A versatile clear coat shader that lets you use duo coloring, add metallic flakes and even mix clear coat with standard lighting within the same material.
Details →

Transmission Shader

A versatile subsurface scattering or transmission shader, that lets you create materials like wax or thin plastics and supports transmission as well as wrapped diffuse lighting and can mix both with standard lighting within the same material.
Details →

Fuzz Shader

The fuzz shader lets you create materials such as e.g. moss – materials which do not follow a standard microfacet brdf model because they are a lot more porous and show up transmitted light.
Details →

Glass Shader

The glass shader allows you to add refractive, tinted and bump mapped glass like objects.

Details →

Lit and shadowed Particles Shaders

The lit particles shaders offer advanced lighting support such as additional lights, transmission and real time shadows. Details →

Flat shaded Shader

Add flat shading to any mesh regardless of its vertex normals. Details →

Effects

Fast Outline Shader

The fast outline shader allows you to highlight objects by adding a colored outline to them without having to use any expensive full screen image effect. The shader instead is based on the stencil buffer and needs the objects to be drawn twice. Details →

Toon Outline Shader

The toon outline shader is a rather simple shader which creates outlines by drawing the geometry a second time, extruding the vertices along the geometry's normals. Details →

Decal Shaders

The decal shaders let you add screen space decals which act more or less like deferred decals. Details →

Billboard Shader

The billboard shader lets you create fully camera aligned or camera facing upright oriented lit and unlit billboards suitable to create e.g. light halos, UI icons or distant trees and objects. Details →

Volumetric shaders

The volumetric shaders let you create fast volumetric light beams or simple box or sphere volumes. Details →

Lux URP/Lit Extended Shader

This shader is needed to prepare the stencil buffer in case you want to draw outlines as described in the chapter Highlight Shaders.

Special Shader Inputs

You will find all needed stencil buffer options in the foldout *Surface Options:*

ZTest Lets you tweak depth based face culling.

Stencil Reference The value to be compared against

Read Mask Bit mask which determines which bit will be read from.

Write Mask Bit mask which determines which bit will be written to.

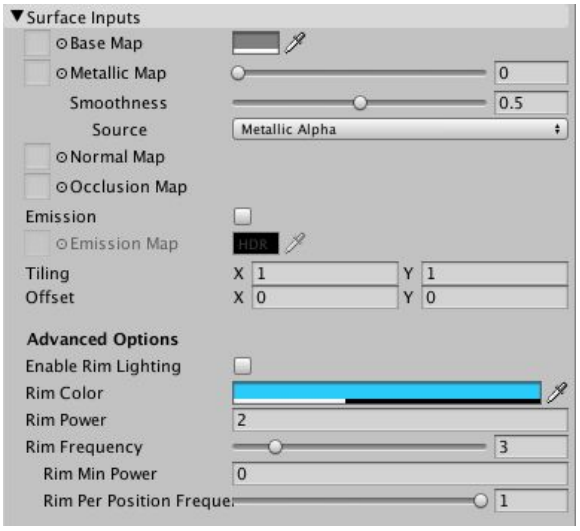
Stencil Comparison The function used to compare the reference value to the current contents of the buffer.

Stencil Pass Op What to do with the contents of the buffer if the stencil test (and the depth test) passes.

Stencil Fail Op What to do with the contents of the buffer if the stencil test fails.

Stencil Z Fail Op What to do with the contents of the buffer if the stencil test passes, but the depth test fails.

Next to exposing the needed stencil options it also allows you to simply highlight objects by adding Rim Lighting. You will find the corresponding settings in the foldout *Surface Inputs.*



Enable Rim Lighting Check to enable Rim Lighting

Rim Color The Rim Color

Rim Power Higher values will push the effect towards grazing angles only.

Rim Frequency Lets you add a simple sinus based animation. If set to values > 0.0 the shader will lerp between *Rim Power* and *Rim Min Power*.

Rim Min Power Second Power value used if Rim Frequency is > 0.0.

Rim Per Position Frequency Slightly offsets the animation based on the object'S pivot in worldspace and prevents objects all pulsating at the same frequency.

Lux URP/Lit Extended Uber Shader

This shader is derived from the Lit Extended one. But it adds support for additional features such as *parallax mapping, horizon occlusion, bent normals, geometric specular AA* or *camera fading*.

Special Shader Inputs

As you should be familiar with most of the inputs I will only cover the special/new inputs here.

Surface Options

ZTest Allows you to define depth testing.

Camera Fading If checked the shader will stipple out the geometry when the camera gets close *(only available if Alpha Clipping is enabled).*

Fade Distance Lets you specify at which distance the shader will start to fade out the object. *(When setting up this param make sure that the editor camera has a proper near clipping plane!)*

Fade Shadows Check this in case you want shadows to fade out as well. *(Please note that dithered shadows may look pretty odd especially if you have a low res shadow map)*

Shadow Fade Dist Distance at which shadows will start to fade out.

Please note that enabling Alpha Testing on an opaque object just to be able to fade it out when it gets close to the camera may be rather expensive: As soon as you enable Alpha Testing early depth testing will be disabled by the GPU which means that the GPU will evaluate all pixels – even those which are hidden by other geometry :(

So in case you use Camera Fading on e.g. walls consider swapping out materials when the camera gets close: Use a regular opaque material variant for any instance not close to the camera and only switch to the Fade variant if the camera already is pretty close.

Do so using Unity's LODgroup component or use triggers.

Advanced Surface Inputs

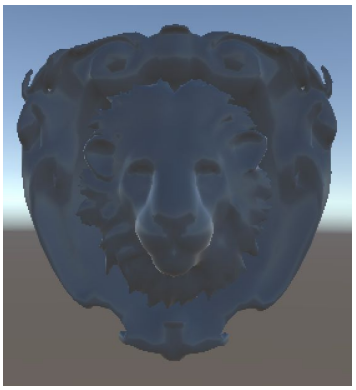
Height Map (G) RGB texture which stores height in the green color channel used by parallax extrusion.

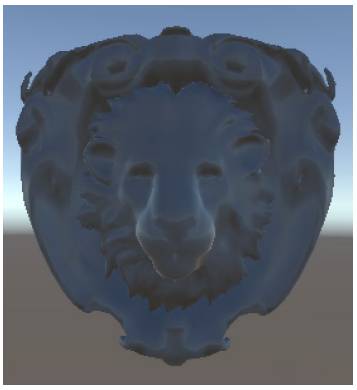
Extrusion Amount of parallax extrusion. *(The unnamed slider)*

Parallax Shadows If checked the shader will apply parallax mapping even in the shadow caster pass. This is somehow correct for directional shadows where we can derive the view direction from the (shadow) camera's forward vector but not in case we render spot lights. Furthermore even parallax directional shadow casters are quite unstable if you rotate the camera. So check if you really need this... *(If Alpha Clipping is enabled only)*

Bent Normal Map Cosine weighted Bent Normal Map in tangent space. If assigned the shader will tweak ambient diffuse lighting and ambient specular reflections.

It uses a rather simple and all but physically correct approach: Ambient lighting gets sampled in the direction of the bent normal and specular ambient lighting is reduced by the dot product between bent and regular normal in world space.


Regular shading


Bent normals


Bent normals and AO

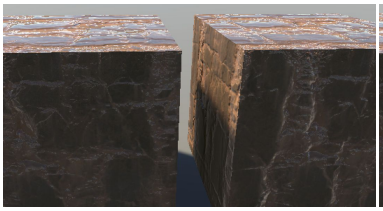
Horizon Occlusion Terminates light leaking caused by normal mapped ambient specular reflections where the reflection vector might end up pointing behind the surface being rendered: https://marmosetco.tumblr.com/post/81245981087. A value of 0 does not terminate anything.

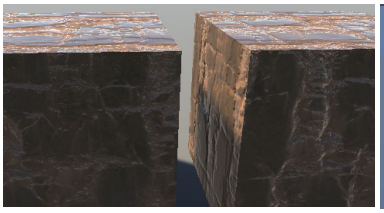
Geometric Specular AA When enabled the shader reduces specular aliasing on high density meshes by reducing smoothness at grazing angles.

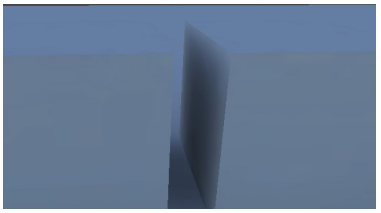
> **Screen Space Variance** Controls the amount of Specular AA. Higher values give a more blurry result.
>
> **Threshold** Controls the amount of Specular AA. Higher values allow higher reduction.

GI to Specular Occlusion In case you use lightmaps you may activate this feature to derive *some kind* of specular occlusion just from the lightmap and its baked ambient occlusion.

> **GI to AO factor** Controls the amount of specular occlusion. It acts as a factor to brighten the value sampled from the lightmap.
>
> **Bias** Adds a constant value to brighten the value sampled from the lightmap.


GI to AO enabled


GI to AO disabled Have a look at the reflections in the gap between the two cubes.


Lightmap

Advanced

Render Queue *Here i dropped the way the built in Lit GUI handles things and returned to the old school way. So instead of using Priority you will see the final Render Queue. As the built in Lit shader always offsets the predefined render queues by + 50 i did so as well. So you will get a render queue = Geometry + 50 = 2050 if you switch the material from transparent to opaque.*

Mesh Terrain Shader

Using Unity's built in terrain might be slightly over the top in certain cases or simply not feasible due to its high demands regarding memory consumption and CPU/GPU usage.

Here the *Mesh Terrain Shade*r kicks in which lets you mix up to 4 details textures – based on a splat map or vertex colors. Latter allows you to texture your terrain right in Unity using e.g. *PolyBrush*.

Using a mesh based terrain offers a lot of advantages as you can simply sculpt cliffs and overhangs and get proper texturing without using expensive solutions such as triplanar mapping. A mesh based terrain also lets you add vertex density only where you need it – where otherwise you would have to crank up the heightmap resolution to be able to add the details you need. Mobile? Something you should consider… Great games have been made using a technique like this. Just think of all games by Naughty Dog. It is labour intensive but will pay off.
Please check out the *Environment Demo* to find out more.

Shader Inputs

Surface Options

Receive Shadows If unchecked the material will not receive shadows *(faster)*.

Surface Inputs

Enable normal Turns normal maps on and off.

Enable Top Down Projection If checked the *first detail texture* will be sampled in world space using a top down projection. This feature is helpful in case:

- you add objects like rocks using the *Top Down Projection shader* and want to make these blend better with the terrain.
- your terrain mesh has several UV shells. Add the top down projected detail texture at the seams to hide them. *See: Path example in the Demo scene.*

Tiling in world space Tiling of the top down projected detail texture in world space. A value of 0.25 means that the texture will be repeated every 4 meters.

Detail [0-3] Albedo (RGB) Smoothness (A) Detail albedo in RGB and smoothness in Alpha.

Normal [0-3] The related normal maps.

Use Vertex Colors If checked the shader will ignore the *Splat Map* and mix the detail textures based on vertex colors. Just like in the spalt map red maps to detail 0, blue to detail 1, green to detail 2. If the vertex color is black detail 3 will show up.

Splat Map (RGB) The splat map drives the distribution of the detail maps. Red maps to detail 0, blue to detail 1, green to detail 2. If the splat map is black detail 3 will show up.
Please note: The splat map must be imported as linear texture and should be uncompressed., So uncheck "sRGB (Color Texture)" in the import settings.

Detail Tiling (UV) Tiling of all non top down projected detail texture according to the UVs.

Specular Overall specular color.

Occlusion Although the shader currently does not support explicit occlusion maps you may tweak occlusion to better make the terrain grounded.

Tips

Use common UV tricks to:

- break up tiling.
- make the textures follow the actual geometry like in the *path example* by creating several UV shells.

Grass and Foliage Shaders

The grass and foliage shaders allow you to place individual patches of grass or foliage as single game objects within your scene whose bending is globally controlled by a built in wind zone.

These are the most complex shaders/solutions as they involve proper meshes with baked bending information, a wind input, layer based culling and finally the shaders themselves.

Performance

The **grass** shader is the faster one as it performs simple bending and lighting. The **foliage** shader does a more complex bending animation and supports translucent lighting or transmission.

Please note: Both shaders support *Alpha Clipping* or *Alpha Testing* which disables *early depth testing* due to how GPUs work and may produce a lot of overdraw. In case you use solid geometry which does not need any alpha testing just disable *Alpha Clipping* set the shader to *Render Queue* = *Geometry* (2000).

If *Alpha Clipping* is disabled the shader will *scale* the instances over distance instead of tweaking the cut off value in order to make them fade out.

Using the SRP batcher combined with layer based culling → allows us to render even a vast amount of grass patches / foliage instances quite efficiently on the CPU. Of course 1 mio grass patches in a 1x1 km level are out of scope. But running 3K grass patches in a 300 x 300 meters levels isn't a problem at all.

When placing **grass** you do not want to place each grass instance separately but want to place patches containing several grass meshes instead: Doing so you will keep the number of game objects and renderers at a reasonable amount. On the other hand it makes it a bit more difficult to place the patches as outer grass meshes might not align up well with the given terrain. So create rather small patches of 1x1 – 3x3 meters in size and place them properly.

When it comes to **foliage** you most likely will place each instance individually.

In order to improve performance use layer based culling → so all grass or foliage instances will be skipped at a distance of e.g. 30 / 50 meters (lightweight!). The grass and foliage shaders support distance based fading so you can hide any popping.

LuxLWRP_LayerBasedCulling Script

This script is just a placeholder and offers to setup layer based culling for two layers. You most likely will extend it and connect it to your quality settings.

Layer based culling is an effective performance optimization as it is pretty fast. In order to make it work properly you should add a special layer on which you will place all objects which shall be culled at a distance smaller than the camera's far clip plane.

The inputs of the script are the following:

Small Details Layer Choose a layer and make sure that small objects actually are set to this layer.

Small Details Distance The distance at which all objects on the specified *Small Details Layer* will be culled.

Medium Details Layer Choose a layer and make sure that medium sized objects actually are set to this layer.

Medium Details Distance The distance at which all objects on the specified *Medium Details Layer* will be culled.

Wind Input

Bending is driven by a **global wind render texture** which contains a slightly modulated wind strength and some additional noise in order to create some more distinguished gusting wind.

This wind render texture gets updated each frame which enables us to rotate the wind direction at runtime without creating weird bending grass instances. Using a wind texture allows us to have spatial differences concerning wind strength and gust without having to implement any fancy per instance logic in the vertex shader.

The global wind render texture is created by the *LuxLWRP_Wind.cs* script, which has to be assigned to your main directional wind zone as it gets its main inputs like wind turbulence and direction from that wind zone:

- *Main* (wind strength) from the wind zone settings will be directly pushed to the shaders.
- *Turbulence* will effect the contrast of the global wind render texture.
- *The rotation* of the wind zone will drive the scroll direction.

Either add the *LuxLWRP_Wind.cs* script to your main wind zone or – in case you have none – drag the *PF Lux LWRP Wind* into your scene.

LuxURP_Wind Script

Update In Edit Mode If checked the wind render texture will be updated in editor each frame which helps you to adjust the settings. In case you do not work on tweaking the the wind just uncheck it to free resources in the editor.

Render Texture Settings

> **Resolution** Resolution of the created global wind texture. Keep it as low as possible to save resources like memory and GPU time.

Format Choose between ARGB32 (faster and less memory consumption) and ARGBHalf (more precise).

Wind Base Tex In order to be able to create a global wind texture you have to provide a proper base texture. The package ships with a default wind base texture (*Default Wind Base texture.psd*). This texture contains various grayscale perlin noise textures at different scales in each of the color channels (RGBA). While RGB will only be used to calculate the final wind strength the texture assigned to the alpha channel (A) will contribute to the wind strength but also determine the wind gusting.
Please note: The texture should be imported as linear texture, so uncheck "sRGB (Color Texture)" in the import settings. In order to get the best quality i recommend to set the *Compression* to *None* – at least to *High*.

Wind Composite Shader Shader which is used to render the final global wind texture. Assign the included *Lux LWRP WindComposite.shader* – or a proper custom one.

Wind Multipliers

Grass Global multiplier for the wind strength as retrieved from the assigned wind zone and applied in the grass shader.

Foliage Global multiplier for the wind strength as retrieved from the assigned wind zone and applied in the foliage shader.

Wind Speed and Size

Base Wind Speed Speed in meters per second at which the global wind texture will "scroll" in world space. The value is relative to the *Main* (Wind Strength) value of the attached wind zone.

Size In World Space Determines the area in world space the global wind texture covers before it gets tiled and repeats.

Speed Layer 0, Speed Layer 1, Speed Layer 2 These are multipliers for the overall set *Base Wind Speed*. In order to create a lively global wind texture you should use varying values here.

Noise

Grass Gust Tiling RGB channels of the assigned *Wind Base Texture* will be sampled using fixed UVs as they already contain perlin noise textures at various scales – however you may finetune gusting wind by changing this parameter. Higher values will create smaller waves of gusting wind.

Grass Gust Speed Just like the other Speed multipliers.

Layer To Mix With Lets you choose a Wind Layer you want the dedicated Gust sample to be combined with.

Jitter

When using the foliage shader and texture driven wind high frequent edge fluttering will be fed by the two jitter frequencies and

Grass Shader Inputs

Surface Options

Alpha Clipping If checked the shader will do alpha clipping or alpha testing which is the desired behaviour in case your texture contains any alpha.

Threshold If the alpha channel contains different shades of gray instead of just black and white, you can manually determine the cutoff point by adjusting the slider.

Please note: If you disable *Alpha Clipping* you should set the shader to *Render Queue = Geometry* (2000). If you re enable *Alpha Clipping* set back the shader to *Render Queue = AlphaTest* (2450).

Receive Shadows If unchecked the material will not receive shadows *(faster)*.

Surface Inputs

Albedo (RGB) Alpha (A) The grass texture (RGB) and the Opacity mask (Alpha)

Smoothness Overall smoothness factor. As grass should use upwards pointing normal specular lighting and ambient reflections will look pretty odd. So i recommend to use quite low values here like 0.1. This will make grass being grounded within the environment but not looking crazy.

Specular The specular color. **Smoothness** Overall smoothness factor. As grass should use upwards pointing normal specular lighting and ambient reflections will look pretty odd. So i recommend to use quite low values here like 0.1. This will make grass being grounded within the environment but not looking crazy.

Specular The specular color.

Occlusion Lets you adjust the ambient occlusion as sampled from the vertex colors.

Wind

Wind Strength acts as a multiplier on the global settings and lets you adjust bending without having to rework the vertex colors.

Normal Strength determines the impact of the bending on the normal. Any value > 0.0f will effect diffuse and specular lighting.
Please check the Normal Strength under various lighting conditions (light angle, view angle) to find the right value. Increasing smoothness and having a look at the ambient specular reflections can be a help here.

Sample Size Wind strength is sampled from the render texture at the world position of the given vertex offsetted by vertex.color.red. *Sample Size* scales the value of vertex.color.red. So setting it to 0.0 will fully flatten vertex.color.red and all vertices will sample the wind strength at their world position while setting it to 4.0 will allow certain vertices to use a world position + 4 (along the xz axis) as sample location.
So think of it being some kind of additional local turbulence factor.

LOD Level lets you specify the LOD level at which the wind render texture gets sampled. Using higher LOD levels will flatten out the sampled wind strength and smooth bending over all instances.
So think of it as some kind of negative local turbulence factor.

Distance Fading

Max Distance Determines the distance at which grass will be fully faded out. The distance should match the culling distance you use in the layer based culling.

Fade Range Determines the fade range.

Please note: Both values are "manipulated" by the material editor. *Max Distance* maps to _DistanceFade.x and *Fade Range* to _DistanceFade.y where *_DistanceFade.x = Max Distance * Max Distance;* and *_DistanceFade.y = 1.0f / ((Fade Range * 2) * (Fade Range * 2));*

Important: Please make sure that the grass actually gets culled by the layer based culling and not only fades out. Check it by setting *Max Distance* to a much greater value than the one you use in the layer based culling settings.

Advanced

Enable Blinn Phong Lighting Lets you switch to a cheaper lighting model.

Enable Specular Highlights If unchecked the shader does not calculate direct specular highlights which makes it cheaper.

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

Foliage Shader Inputs

Surface Options

Alpha Clipping If checked the shader will do alpha clipping or alpha testing which is the desired behaviour in case your texture contains any alpha.

Threshold If the alpha channel contains different shades of gray instead of just black and white, you can manually determine the cutoff point by adjusting the slider.

Please note: If you disable *Alpha Clipping* you should set the shader to *Render Queue = Geometry* (2000). If you re enable *Alpha Clipping* set back the shader to *Render Queue = AlphaTest* (2450).

Receive Shadows If unchecked the material will not receive shadows (faster).

Surface Inputs

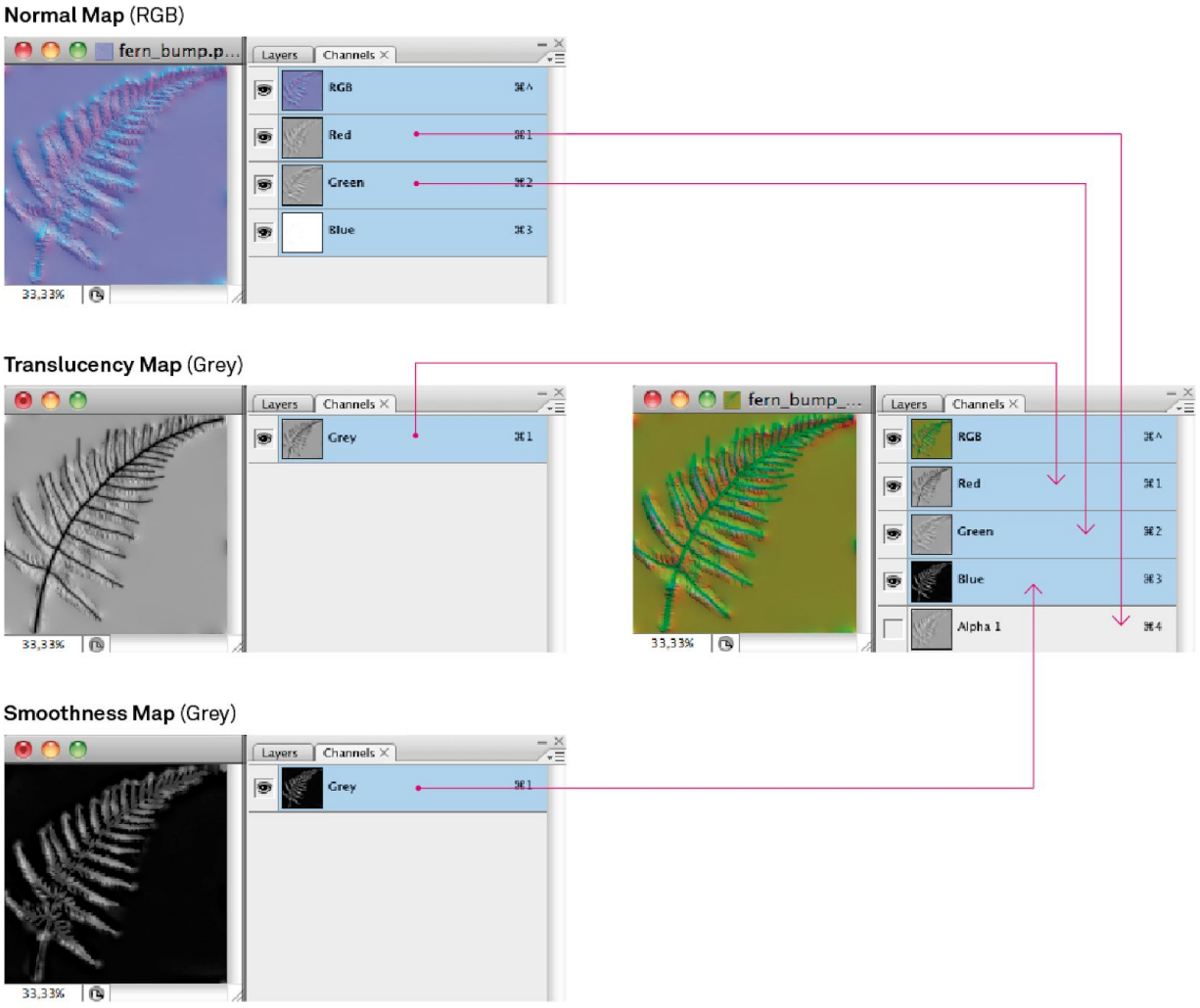
Albedo (RGB) Alpha (A) The foliage texture (RGB) and the Opacity mask (Alpha)

Smoothness Overall smoothness factor.

Specular The specular color.

Enable Normal Smoothness Trans Map If checked the shader will sample the according texture.

Normal (AG) Smoothness (B) Trans (R) Combined texture which stores the normal in AG, smoothness in B and translucency or thickness in R. The texture can be assembled like shown below:



Combined Normal/Smoothness/Translucency Texture Color Channel Layout

Please note: These combined textures do not contain a color or albedo texture – so uncheck "sRGB (Color Texture)" in the import settings.
As the texture contains a normal map you might consider setting the *Filter Mode = Trilinear*.

Smoothness Scale Lets you remap the sampled smoothness value.

Transmission

Power Determines view dependency. Higher values will make transmission kick in only when the view ray more or less directly points towards the light source.

Strength Lets you scale transmission.

Shadow Strength As transmission might be totally eliminated by self shadowing this parameter lets you suppress shadows when it comes to transmission. Other lighting features (diffuse, specular) are not affected.

Mask by incoming shadow strength Lets you suppress transmission according to the shadow strength as set on the given light source - or from point lights which do not cast any shadows.

Distortion When calculating transmission the shader distorts the inverted light direction vector slightly by the given normal to simulate the scattering. Default value is 0.01. Higher values will give you more scattering and break up the uniform look.

The subsurface color will be derived from albedo.

Wind

Wind Input Choose between *Math* or *Texture*. *Texture* should be a bit faster. *Math* a bit more predictable.

Primary Strength Lest you remap the primary or main bending strength as encoded in vertex color alpha.

Secondary Strength Lest you remap the secondary or detail bending strength as encoded in vertex color blue.

Edge Flutter Lets you adjust the edge flutter strength as encoded in vertex color green.

LOD Level lets you specify the LOD level at which the wind render texture gets sampled. Using higher LOD levels will flatten out the sampled wind strength and smooth bending over all instances. *Wind Input = Texture only.*

Sample Size Primary and secondary wind strengths are sampled from the render texture at the world position of the given vertex. Setting *Sample Size* to 0.0 will make all vertices fetch the wind strengths at the pivot of the given instance while setting it to 1.0 will make the shader sample strengths at the vertex positions in world space which will add some more noise and disturbance.

Distance Fading

Max Distance Determines the distance at which grass will be fully faded out. The distance should match the culling distance you use in the layer based culling.

Fade Range Determines the fade range.

Please note: Both values are "manipulated" by the material editor. *Max Distance* maps to _DistanceFade.x and *Fade Range* to _DistanceFade.y where *_DistanceFade.x = Max Distance * Max Distance;* and *_DistanceFade.y = 1.0f / ((Fade Range * 2) * (Fade Range * 2));*

Important: Please make sure that the foliage actually gets culled by the layer based culling and not only fades out. Check it by setting *Max Distance* to a much greater value than the one you use in the layer based culling settings.

Advanced

Enable Specular Highlights If unchecked the shader does not calculate direct specular highlights which makes it cheaper.

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

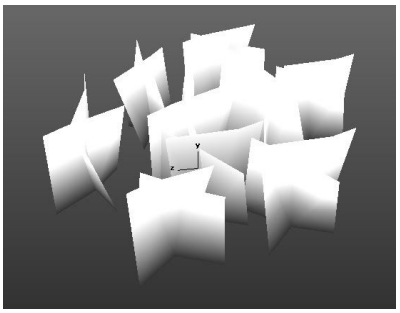
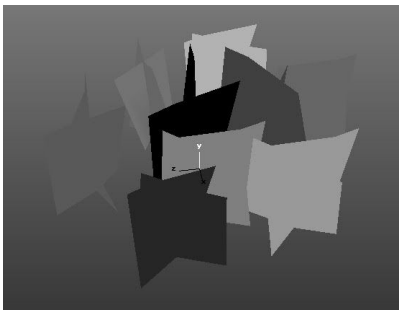
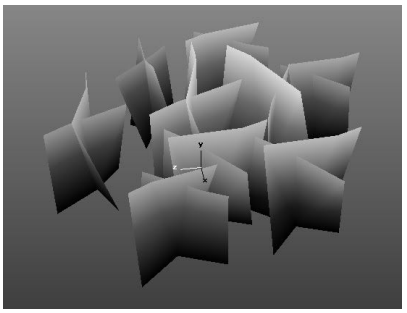
Custom Grass and Foliage Models

When using custom grass or foliage models you have to take care about:

- the fact that the grass shader expects single sided geometry.
- vertex colors properly set up as they drive the bending.
- the vertex normals.

Vertex Colors Grass Shader

The **grass shader** uses vertex.color.rgb to control ambient lighting and bending.

Red – Ambient Occlusion	**Green** – Phase Variation	**Blue** – Bending
		
Just some ambient occlusion added to the lower vertices.	Different grass "instances" have different green values to add some variation to the bending as the final sampling position for the wind texture will be displaced by the amount of the green color value.	The blue color channel contains a gradient from black at the bottom to a medium gray or white at the top. This will fix lower vertices while upper ones may bend.

The vertex color usage in the grass shader is similar to how Vegetation Studio expects vertex colors. So any grass mesh working with VS should work with Lux LWRP as well.

In order to apply bending (blue) on import please have a look into the section Adding Bending on Import →

Vertex Colors Foliage Shader

The **foliage shader** supports a more complex bending as you might already know from the built in tree creator shaders and is based on Crytek's original tree bending. It is controlled by the vertex colors applied to the model and consists of 3 blended animations:

1. **Primary or main bending** which animates the entire model along the wind direction.
2. **Secondary or detail bending** adding a higher frequent animation mostly along the y-axis.
3. **Edge fluttering** which is a high frequent bending animation originally invented by Crytek to simulate single leaves on large leave planes.
4. In order to make the whole blending more believable, there is a fourth factor: per leaf / per branch **phase variation** which controls the delay of the detail bending.

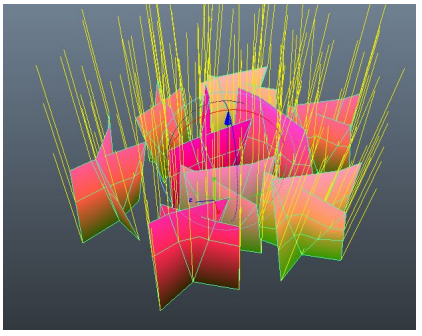
Red Phase Variation	**Green** Edge Flutter	**Blue** Detail Bending	**Alpha** Main Bending
			
Red stores phase variation. Each branch should always have one single red value.	Just some small **green** values (barely visible here) only at the outer tips of the leaves. Make sure that the pivot and the spine of the leaves are set to green = 0.	The **blue** color channel stores detail bending. It should be a gradient from black at the pivot of the branches to a medium gray or white at their outer tips.	**Alpha** stores main bending which is a gradient from black (bottom) to some brighter gray or even white at the upper vertices,

The vertex color usage in the foliage shader is similar to how AFS or ATG expect vertex colors. So you may use the AFS Foliage tool to apply or adjust vertex colors. Furthermore you can use the Custom Tree Importer to add vertex colors automatically on import.

In order to apply main bending (alpha) on import please have a look into the section Adding Bending on Import →

Normals Grass Shader

The vertex normals should more or less all point upwards so the final lighting of the grass patch is mainly driven by its overall orientation. This way the lighting of the grass matches the lighting of the underlying terrain.



Normals Foliage Shader

When using the foliage shader the vertex normals should match the given geometry: The shader will do wrapped around diffuse lighting by default and as it uses VFACE any smoothed normals will cause very harsh lighting on backfaces.

Adding Bending on Import

In case you do not want or can't apply a proper gradient in the blue color channel (grass) or the alpha channel (foliage) of the vertex colors you can make use of the *LuxLWRP_GrassMeshPostprocessor* script. This script will add the related vertex color automatically on import and whenever the grass or foliage mesh is updated.

In order to let the script find all models that should be processed you will have to name them properly: The script finds them by name which must contain “_LuxGM“ for grass meshes and “_LuxFM“ for meshes using the foliage shader.

Maximum Bending

You also have to specify the max bending value by adding it directly after “_LuxGM“ or “_LuxFM”. Bending values must be in the range of 0.0 – 1.0 and have to be written as 2 digits without the period:

“00“ -> 0.0 / “05“ -> 0.5 / -> “10“ -> 1.0

So a file named: “SM MyTallGrass_LuxGM05_test.fbx” would be processed using a max bending value of 0.5.

A file named: “SM MySuperfern_LuxFM03Testnew.fbx” would be processed using a max bending value of 0.3.

A file named: “SM MySuperfern_LuxFM1Testnew.fbx” would throw an error.

Power

In case you have a more complex model which consists of several triangles you may also specify a **power factor**. Adding a power factor will change the linear gradient which gets applied towards an exponential gradient and results in much livelier bending.

Do so by adding “_POW” + 2 digits to the file name e.g.:
“SM Grass Patch_LuxGM10_POW19.fbx”

Processing Details

Main bending (vertex color blue in case of grass meshes and vertex color alpha in case of foliage meshes) will be set according to the “local” y position of the given vertex, the max bending and the power value: Main bending = 0 for all vertices if vertex.pos.y < 0 and a lerped value for all other vertices according their y coordinate divided by the height of the bounding volume.

Please note: The postprocessor expects a single flattened mesh. It does not merge sub meshes and skips processing on models which have more than 1 sub mesh. It does not bake occlusion or phase but keeps the values stored in the vertex colors.

Setting up Wind

In order to give you a better understanding of how the wind works an help you setting it up i added the *Wind Setup* scene.

It contains the *PF Lux LWRP Wind Prefab* and a plane which uses the *M Lux LWRP Vegetation Wind Visualize material:* This actually lets you view the generated wind render texture.

In order to see how the wind render texture changes over time and reacts to changes in the settings of the script or the rotation of the game objects which holds the wind zone and script make sure you have checked *Update In Edit Mode.*

By default the plane shows the *Combined Wind* which is how the vegetation shaders will combine *Wind Strength* and *Wind Gust*.

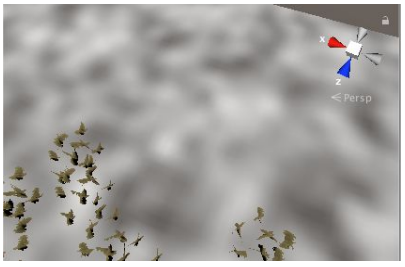
In order to visualize the original inputs as stored in the wind render texture you have to edit the visualisation material and change *Visualize* to *Wind Strength* or *Wind Gust*.

In case you get confused because the bending of the grass does not match the wind strength as visualized by the underlying plane: Due to "Sample Size" in the grass material and according to the baked vertex colors → red = phase grass may sample wind at a different location. If you set "Sample Size" to 0.0 it should perfectly match.

Wind Strength

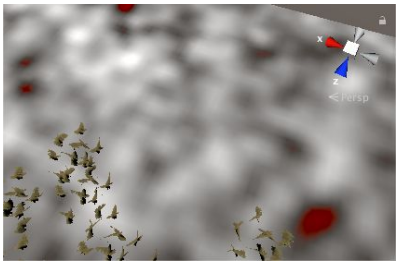
The combination of the 3 + 1 wind layers. Contrast will change according to the *Turbulence* settings in the wind zone.

Wind Gust



The combination of the dedicated gust layer and the selected *Layer To Mix With*. Contrast according to the *Turbulence* settings in the wind zone.

Combined Wind



The combination of *Wind Strength* and *Wind Gust* as calculated within the vegetation shaders. Red pixels indicate negative wind strengths.

Now you should be able to adjust speeds and tilings driving the render texture and adjust your materials.

Final bending on each prefab will be controlled by:

- the Main (wind) parameter in the wind zone settings
- the input from the wind render texture
- the according wind multiplier in the script
- the baked vertex colors
- the wind parameters in the used material

Bending quality

Bending quality depends on:

- the number of vertices in the mesh: Less vertices will result in harsher bending.
- the accuracy of the bending information stored in the vertex colors: Any discontinuities here will result in jagged bending.
- the accuracy/compression and resolution of the wind input textures: The one that gets sampled to create the render texture and the render texture itself.

Top Down Projection Shader

Shader Inputs

Surface Options

Receive Shadows If unchecked the material will not receive shadows *(faster)*.

Surface Inputs

Albedo (RGB) Smoothness (A) Albedo texture in RGB, Smoothness in Alpha.

Enable dynamic tiling If checked the shader will adjust the giving tiling values according to the scale of the object. This is a nice feature in case you care about texel density, use a tiling texture and add the related object at various scales.

Smoothness Scale Lets you remap the sampled smoothness value.

Specular Specular color.

Enable Normal Map If checked the shader will sample the assigned normal map(s).

Normal Map The normal map.

Normal Scale Scale of the sampled normal.

Mask Map

Enable Mask Map Check to make the shader sample the mask map.

Metallness (R) Occlusion (G) Height (B) Emission (A) This combined texture contains all other features supported on the base material. Height (B) in this case does not do anything to the base material but acts as detail mask for the top down projection. So it does not have to contain a height value but could also store a simple noise map.
Please note: The texture should be imported as linear texture, so uncheck "sRGB (Color Texture)" in the import settings.

In case you do not need Metallness, Occlusion or Emission always consider using *Get Mask from Normal* as described below because this will strip off one texture lookup.

Emission Color Tint color for the emissive lighting.

Bake Emission Must be checked in case you bake lightmaps.

Occlusion Lets you dampen the sampled occlusion.

Top Down Projected

Enable Top Down Projection Check this to make the shader add the top down projected texture set.

Albedo (RGB) Smoothness (A) Albedo texture in RGB, Smoothness in Alpha.

Smoothness Scale Lets you remap the sampled smoothness value.

Get Mask from Normal If checked the shader expects a tweaked normal map which also contains a mask map. See Get Mask from Normal for details.

Normal (RGB) or Normal (AG) Mask (B) The normal map. Either as regular normal map (if *Get Mask from Normal* is unchecked or as combined normal and mask texture)

Normal Scale Scale of the sampled normal.

Tiling Tiling in world space. A value of 0.25 means that the texture will be repeated every 4 meters.

Terrain Position Describes the offset for the sampling in worldspace. Useful in case you want to make the projected texture match a detail texture on the terrain.

Blending

Angle Limit Lets you limit the projection according to the y (or up) component of the world space normal.

Strength Will scale the blend amount.

Base normal Influence Lets you determine the impact of the base normal on the blend amount.

Base Normal Strength Lets you reveal the base normal – even if the projected layer fully covers the surface.

Height Influence Lets you determine the impact of the height (or noise) sample on the blend amount. This will only work if the *Mask Map* is enabled or you have checked *Enable Normal Map* and *Get Mask from Normal*.

Fuzz Lighting

Enable Fuzz Lighting If checked the shader will add simple fuzzy lighting to the top down projected texture which lets you create materials such as e.g. moss.

Diffuse Wrap The shader will apply smooth wrapped around diffuse lighting. This value lets you adjust the light wrapping. Setting it to 0 will give you built in Lambert lighting.

Fuzz Strength Fuzz color is derived from the albedo. So if the albedo is rather dark you may raise this multiplier.

Fuzz Power Determines how far the fuzz lighting travels according to the normals. The higher the value the less fuzz lighting will be applied.

Fuzz Bias Lets you add some fuzz lighting regardless of the given normal.

Ambient Strength Determines the Fuzz influence on ambient diffuse lighting.

Advanced

Enable Specular Highlights If unchecked the shader does not calculate direct specular highlights which makes it cheaper.

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

Get Mask from Normal

Usually the shader samples the blend mask from the assigned Mask Map – if any. This texture gets sampled based on the UVs of the given mesh. If the mesh has several UV shells the sampled mask will most likely show up some discontinuities which may be quite distracting.

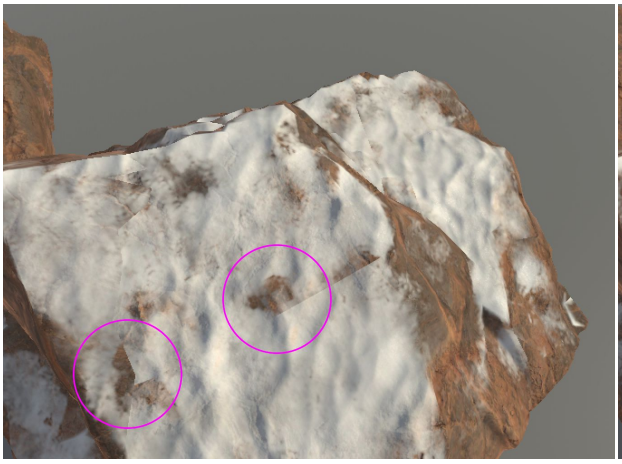
In case you enable *Get Mask from Normal* the shader will sample the mask from the top down projected normal in world space instead – which never has any discontinuities.

In order to use this feature you have to prepare the projected normal:

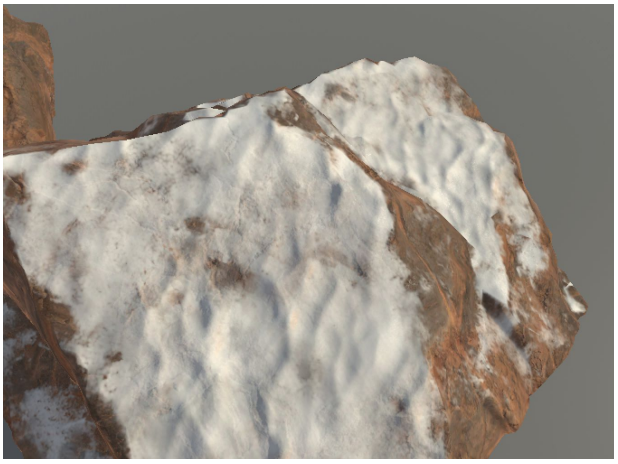
1. Take a regular normal map and add an alpha channel.
2. Copy/paste the red color channel into the alpha channel.
3. Fill the red color channel with pure white. Important!
4. Copy the mask texture into the blue color channel.
5. Import the tweaked normal mask texture as linear texture by unchecking *sRGB (Color Texture)* in the import settings. *Compression* should be set to *High Quality*.

Why do i do this to you? Because we want as less texture lookups as possible!

Have a look at the snow sample to find out more.



Masking artifacts from sampling the mask from the Mask Map using the UVs.



No artifacts when sampling the mask from the tweaked projected normal texture in world space.

Tips

As the projection heavily depends on geometry normals the final result on different LODs when using LODGroups most likely will not match. So consider using cross fading – at least on large objects.

Please note: The progressive CPU lightmapper may fail to calculate proper blending as it does not always?! calculate a fitting world space normal. In order to fix this disable lightmap static, rebake, then enable lightmap static and bake again… Enlighten just works fine tho.

Water Shader

Please note: Make sure your lightweight scriptable render pipeline settings have *Depth Texture* and *Opaque Texture* checked. Latter is only needed if *Refraction* is enabled.

In case your camera uses **orthographic projection** you have to check *Enable Orthographic Support* in the material inspector.

Shader Inputs

Surface Options

ZWrite Lets you choose whether the shader writes to the depth map or not. Usually it should write to the depth buffer

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.

Dest BlendMode This has to be set to *Zero* if *Refraction* is enabled or *OneMinusSrcAlpha* if *Refraction* is disabled. *Should be set properly by the custom ShaderGUI script LuxLWRPCustomWaterShaderGUI.cs.*

Receive Shadows If unchecked the material will not receive shadows *(faster)*.

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

Surface Inputs

The water shader mixes two normal samples to create the final bumpiness.

Water Normal Map The normal map used by both samples. Adjust *Tiling* and *Offset* to make the first sample fit your geometry.

Normal Scale Scale of the first sampled normal.

Speed (UV) Scroll speed in UV space. X equals U, Y equals V.

Secondary Bump This Vector4 combines all parameters for the second normal sample.

Tiling (X) acts as multiplier on the given *Tiling* from the first sample

Speed (Y) acts as multiplier on the given *Speed (UV)*

Refraction (Z) The amount of refraction added to the UVs based on the first normal sample.

Bump Scale Scale of the sampled normal.

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 Blending Lets you fade out the shore line or borders of the water surface.

Enable Refraction If checked the shader will sample and refract the camera opaque texture. Water and refracted background will be mixed within the shader and the final result will be rendered to screen as if it was opaque.

If unchecked the shader will act like a transparent premultiplied shader. Thus the *Dest BlendMode* has to be adjusted properly. *Not using refraction saves two texture lookups.*

Refraction Specifies the strength of the refraction effect.
The shader does two refracted samples of the _CameraDepthTexture in order to occlude objects in the foreground from being occluded. It does not do a 3rd unrefracted sample which results in relatively unstable foam close to the camera if the Refraction value is high. So consider using small Refraction values in case you run into any trouble.

Reflection 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 Fog

Fog Color Color of underwater fog added.

Density Density of the underwater fog according to depth along the view vector.

Foam

Enable Foam Check to enable dynamic foam at shore lines

Foam Albedo (RGB) Mask (A) RGB will tint the foam mask. Alpha drives the opacity.

Foam Tiling Tiling relative to the base UVs

Foam Speed (UV) Scroll speed of the foam texture.

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.

Foam Edge Blending Determines the size of the dynamically calculated foam border.

Foam Slope Strength Lets you add foam according to the normal up direction of the water geometry.

Foam Smoothness The smoothness of the foam.

Advanced

Enable Specular Highlights If unchecked the shader does not calculate direct specular highlights which makes it cheaper.

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

Tips

Use common UV tricks to:

- speed up water at special locations (by shrinking the UVs).
- distort water normals and foam.

- In case you need more foam at special locations just lower the distance between the river bed and the river surface. The closer the river bed is to the river mesh the more foam you will get.

Tree Creator Shaders

The tree creator shaders are a port of the tree creator shaders for the built in render pipeline and allow you to use trees created using Unity's tree creator along with LWRP and URP.

Please note: These shaders do not support the SRP batcher by design. Enable GPU instancing to speed up rendering.

Usage

In order to make your trees use the Lux LWRP tree creator shaders you have to assign the *Lux LWRP Tree Creator Bark* and the *Lux LWRP Tree Creator Leaves* shader to the **base materials** you use to create the tree (the materials which are assigned e.g. in the *Branch Material* or *Break Material* slot of the tree creator). Then refresh the tree.

The final tree will be rendered using the *LWRP Tree Creator Bark Optimized* and the *Lux LWRP Tree Creator Leaves Optimized* shaders. These need the related "Rendertex" shaders which will be used to render the billboard textures.

Shader Inputs

The inputs of the *base* shaders match those of the original Unity shaders. So please have a look at Unity's documentation in case you need more information.

However if you look into the *optimized* shaders you will notice that these contain some LWRP related inputs:

Receive Shadows If unchecked the material will not receive shadows *(faster)*.

Enable Dithering for VR If you are rendering for a VR device Unity will totally change the rendering for trees and billboards. The original implementation solely relies on the keyword "BILLBOARD_FACE_CAMERA_POS" to be enabled — which gets enabled if you enable *Billboards Face Camera Position* in *Project Settings → Quality* — regardless if VR actually is enabled or not.
In order to prevent more expensive rendering if you have checked *Billboards Face Camera Position* but do not use VR i added *Enable Dithering for VR* as a safeguard. So check this if you are using VR and the shaders will do the expensive dithering. If you do not use VR keep it unchecked.

VR only

If VR is enabled Unity will use the built in *CameraFacingBillboardTree* shader to render the billboards. I could not locate or identify the shader, which is responsible for creating the billboard textures (albedo, alpha and normals) – nevertheless this shader sometimes (?!) seems to create *false* textures with leaves having a far too dark alpha. As the original *CameraFacingBillboardTree* shader uses AlphaToCoverage this will give you more or less transparent billboards :(

So i added a hacked *CameraFacingBillboardTree* shader which does not use AlphaToCoverage and applies a little different dithering. If you want to go with the built in one deactivate this shader by e.g. adding a "_" to its file name: "CameraFacingBillboardTree._shader".

Changes compared to the original shaders

I added some little improvements to the shaders:

- Point and spot lights, wind and shadows fade in and out over the billboard *Fade Length* as specified in the terrain settings.
- When calculating the wind based bending the vertex shaders preserve the length of the original vertex which will result in a little less distortion.
- Billboards will be lit using proper SH ambient lighting.

Terrain Shader

The terrain shader supports proper(!) normals, height based blending, parallax extrusion and painted holes (Unity 2019.3. and above only). Since version 1.35 it also support procedural stochastic texturing on the first terrain layer which automatically eliminates tiling artifacts.

In order to make it work with GL ES 2.0 it uses multiple passes to draw the splats in case you have assigned more than 4 layers – just like the built in terrain shaders.

Due to the texture limit of GL ES 2.0 *Mask Maps* are not supported. The heights for the height based blending and parallax mapping instead have to be combined into a single RGBA texture where red contains the height for layer 1, green the height for layer 2, … *Please note: This texture should be imported uncompressed using a rather low resolution. It must be imported as linear texture,*

Height based blending and parallax mapping are only supported for the first four terrain layers. All higher layers will be rendered using a copy of the built in shader.

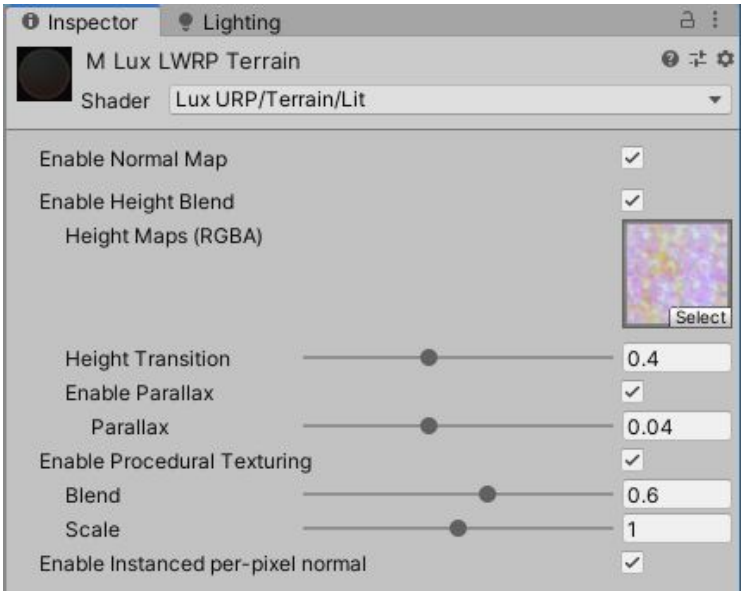
Procedural stochastic texturing is supported only on the first terrain layer. *Please note: The basemap shader does not support procedural stochastic texturing, so you may get a little pop as far as texturing is concerned. But usually this is not really noticeable at all.*

Usage

I recommend to setup your terrain just using the built in shader.
When done:

1. Create a new material using the *Lux LWRP Terrain* shader.
2. Select the terrain settings (gear icon), set *Material* to *Custom* and assign your material to the slot *Custom Material*.
3. Now the terrain should already be rendered using the *Lux LWRP Terrain* shader.
4. In order to fine tune the material and rendering select the material and edit it settings:

Material Settings



Enable Normal Map You have to check this in case you want to enable the normal maps as defined in the terrain layers.

Enable Height Blend Check this if you want to use height based blending and/or parallax mapping.

Height Maps (RGBA) The combined height map textures. The height map of each layer goes to the corresponding color channel starting with layer 1 = red channel.
This texture must be imported as linear texture. So uncheck *sRGB Texture (Color Texture)* in the import settings. It should use *Compression = High Quality* or even *Compression = None*. Please consider to import it with a rather low resolution like *Max Size = 512* or even *256* to speed up rendering.

Height Transition Lets you adjust the sharpness of the height based blending.

Enable Parallax If checked the shader will perform parallax extrusion.

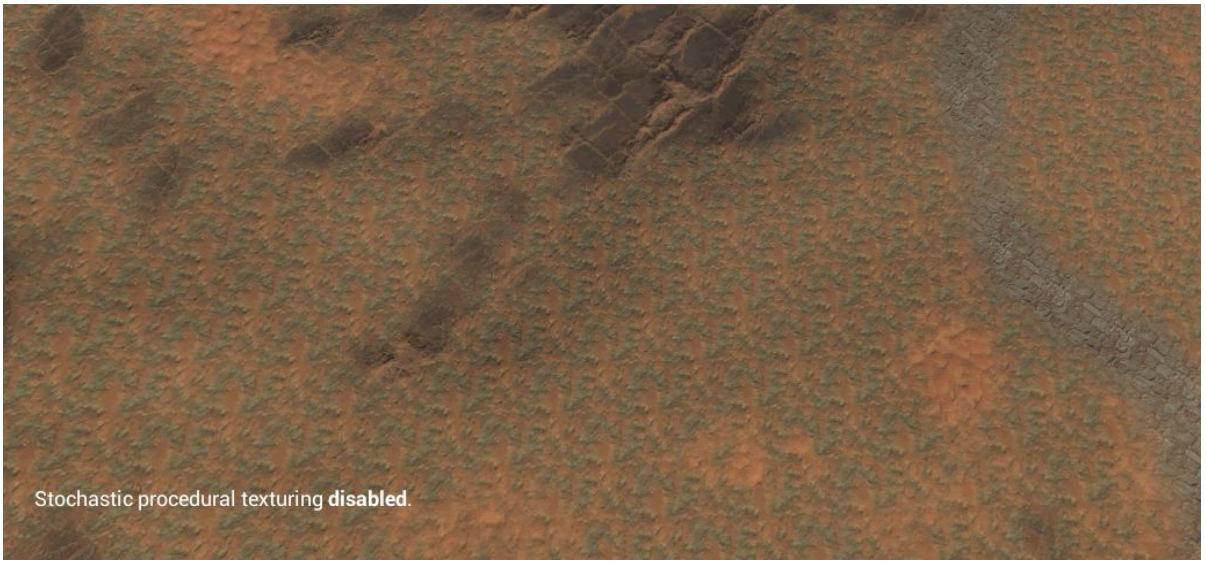
Parallax Determines the strength of the extrusion.

Enable Procedural Texturing If checked the shader will sample the first terrain layer using stochastic procedural texturing which hides all tiling artifacts automatically. *In order to do so the shader will have to to sample the first layer's textures (albedo and normal) 3 times which makes this feature handy but a bit expensive.* Due to the costs the shader supports this feature only on the first terrain layer. So assign your layers wisely and make sure that the first layer contains the texture that covers most of your terrain.

Find out more about stochastic procedural texturing here →.

Blend Lets you adjust the sharpness of the blending between the 3 samples.

Scale Scales the hexadecimal pattern used by the blending. *Keep this around 1.0 to avoid artifacts.*


Stochastic procedural texturing **disabled**.

Enable Instanced per-pixel normal If checked the shader will reconstruct the normal based on the internal terrain normal map if instancing is enabled. Checking this will reduce popping.

Now everything should be fine.

The demo terrain contains 5 layers wich actually is pretty stupid: The fifth layer is the path which is only applied at a pretty small region of the terrain – but forces the entire terrain to be drawn twice. Furthermore the path would really benefit from height based blending but it does not because height based blending is only supported on the first 4 layers.

Quick take away:

- Either use 4 or 8 layers, anything between will waste resources.
- Chose the first 4 layers wisely and assign textures to these where height based blending and parallax mapping will make most sense.

Height Maps

Please make sure that your height maps do not contain pure black. Otherwise the texture will never show up there even if it is painted with full strength.

Terrain Layers

Not all features of Unity's terrain layers are supported due to performance and compatibility reasons.

Diffuse Should contain albedo in RGB and smoothness in A.

Normal Map A regular normal map.

Normal Scale Scales the normal.

Mask Map Is not supported and should be empty.

Specular Is not supported by the terrain engine…

Metallic Metallness

Smoothness Multiplier on the smoothness as sampled from the Diffuse.a channel.
Seems to be broken… and has no effect.

Terrain Blend Shader

The terrain blend shader allows you to smoothly blend meshes like cliffs or overhangs with your terrain. It is still in development and thus flagged as experimental.

The provided solution is still not perfect and suffers from visual glitches as well as some shortcomings regarding LWRP and URP. Feedback is welcome!

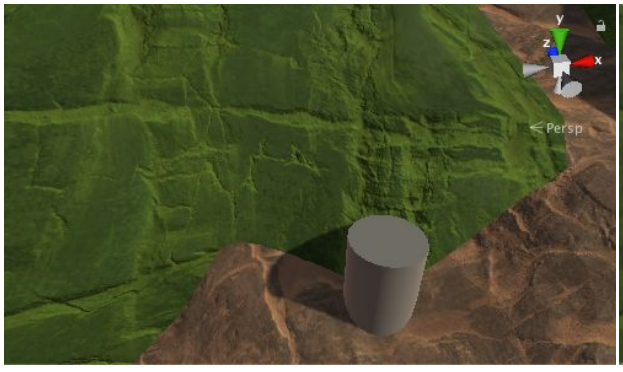
There are different techniques to accomplish a smooth blending – like probably the most common one which draws the mesh object twice using the terrain material for the first draw and projecting it on top of your mesh, which may be rather expensive.

The terrain blend shader works differently and adds a soft intersection between terrain and mesh using alpha blending: Although the mesh is rendered as opaque object (~~in order to benefit from screen space shadows~~ *(these are dropped in URP >= 7.2.)* and proper front to back sorting) it uses alpha blending controlled by the distance between the mesh and the terrain.

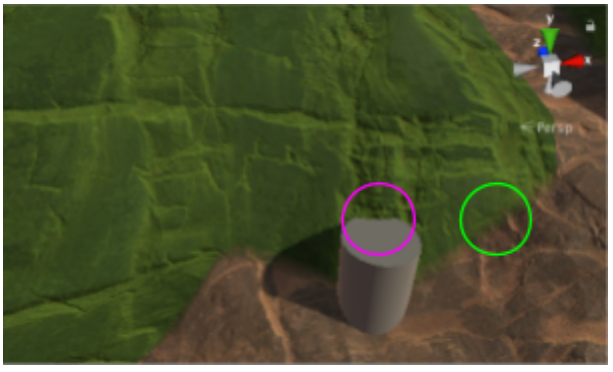
Usually you would use the camera depth texture to calculate the distance, but as we use Render Queue = Geometry -/+ x this texture is not available. For this reason we have to provide a terrain height and normal map from which the shader then can calculate the terrain height at a given position in world space.

When drawing the blended mesh on top of the terrain, the mesh usually is clipped according to the terrain's depth, so our blending could only happen above the terrain's surface which would result in slightly floating meshes and a small gap. How can we apply the blend *below* the terrain's surface?

In order to draw the mesh even when below the terrain's surface the shader pulls the vertices (actually the clip space depth) slightly towards the camera. Doing so however will introduce shading artifacts because other meshes close to the blended one might be clipped like shown in the images below.



Vertices not pushed towards the camera The blending between mesh (tinted green) and the terrain happens below the terrain and thus gets clipped.



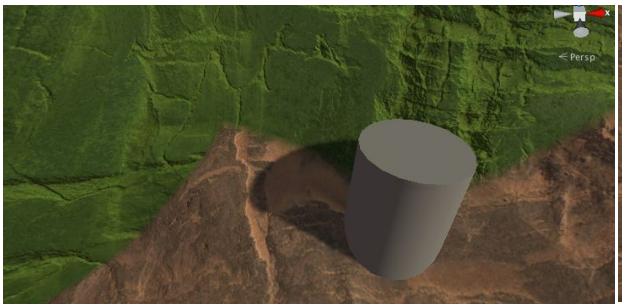
Vertices pushed towards the camera Now the mesh draws on top of the terrain and the blending gets visible (green circle). But it also renders on top of the gray cylinder and clips parts of it (magenta circle).

The solution to handle other objects close to the mesh is:

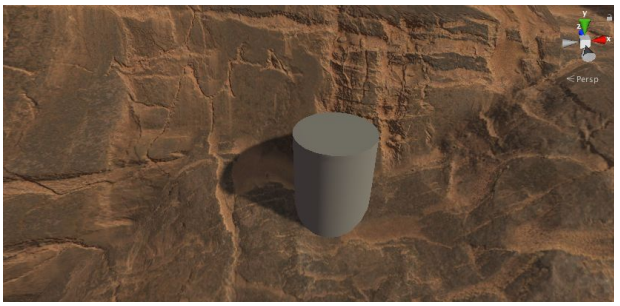
A. Only slightly pushing the vertices towards the camera. *This will result in a fairly small blend range.*

B. Disable *ZWrite* in the Blend Shader and use a second material to write to depth which does not push the vertices. *This will need us to draw the mesh twice.*

Depending on your terrain and the mesh you want to blend you may use A. or B.



Using a 2nd material to write proper depth This solves the problem on the cylinder.



Final result.

Draw Order

Solution b) depends on a proper drawing order: The terrain has to be drawn first, then the depth only, the blended mesh and finally all other objects.

Unfortunately LWRP and URP do not always respect the *Render Queue* as defined in the shader (?!) so we may have to set them manually. The following settings are used in the demo and work with LWRP 5.16.1. and SRP Batcher:

- Terrain: 2000
- Depth only: 2001
- Blend: 2002
- Other Materials which are close to the blended mesh: 2003+
  When using the Lit shader this would need you to lower "Priority" to e.g. -4.

Usage

In order to use the shader you first have to create the needed terrain height normal map.

1. Assign the *GetTerrainHeightNormal* script to your terrain and hit *Get Height Normal Map.* This will let you save a combined height and normal texture to disc.
2. Create a new material and assign the *"Lux LWRP/Terrain/Blend"* shader.
3. Assign the created combined *height and normal texture* to the material and set *Terrain Position* and *Terrain Size*.
4. Now you can assign the material to an object and should get some kind of soft intersection with the terrain. If not please check *Terrain Position*, *Terrain Size* and the *Render Queue* used by the material and the terrain: The terrain must render first.
5. If the soft intersection shows up, you now may tweak the blending by adjusting *Depth Shift*, *Alpha Shift* and *Alpha Contraction*.

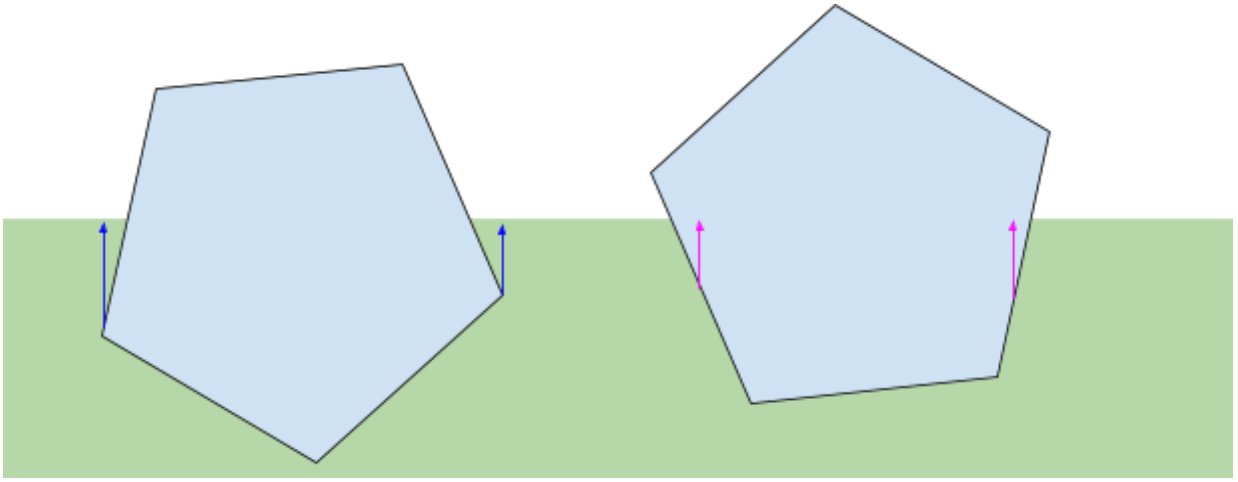
6. By default the blend material writes to depth. In case you use “large” *Depth Shift* values you should add a 2nd material to the *Mesh Renderer* component and assign the *M Terrain Blend Depth Only* material. Then set *ZWrite* to *Off* in your blend material. This will prevent the blend material from clipping other objects close to it.
   Please note: When it comes to lightmapping you have to make sure that the *depth only* material is the 1st one and the “real” material is the last one so the lightmapper will pick up the correct material when it renders the meta albedo. Check it by switching to the *Baked Global Illumination → Albedo* preview in the scene view.
7. Check various view angles and adjust the settings. You may even blend the normals.
   In order to tweak the normal settings i recommend to disable shadows.

Shadows

Since URP 7.2. there are no screen space shadows anymore which made handling shadows quite easy. So now we have to prevent the terrain from shadowing the parts of the blended mesh below it manually.

We do so by simply raising the shadow sampling position in world space by the portion the mesh is below the terrain. This leads to small discontinuities regarding the shape of the shadow which however are barely visible.

Furthermore this only works properly for slopes < 90° as otherwise we pull the shadow sampling coordinates into the mesh itself as shown below. For this reason the shader also lets you pull the shadows sampling coordinates along the view ray.



Shader Inputs

Surface Blending

Depth Shift Amount the vertices get shifted towards the camera. *Keep this as small as possible to avoid shading artifacts – or disable ZWrite as described above.*

Terrain Height Normal The terrain height normal (as generated with the provided script)

Terrain Position Position of the terrain in world space.

Terrain Size Length, height, width of the terrain like in the terrain settings. *Please mind the order!*

Alpha Shift Lets you shift the start of the blending range above or below the terrain surface.

Alpha Contraction Higher values will shrink the range over which the alpha gets blended resulting in a sharper transition.

URP >= 7.2.

Shadow Shift Threshold In order prevent the terrain from shadowing the parts of the blended mesh below the terrain we have to lift the position in world space the shadows are sampled at. *Shadow Shift Threshold* defines the distance to the terrain surface in cm where this lifting kicks in. As we can not 100% precisely reconstruct the terrains height values around 0.05 (= 5cm) should be fine in most cases.

Shadow Shift Factor which determines the final lift of the shadow sampling position. Usually a value of 1.0 should be fine.

Shadow Shift View Pulls shadow sampling position towards the camera.

Normal Shift The shader will blend the geometry normals towards the normals from the terrain to smooth the transition. *Normal Shift* lets you define if the blending shall start above or below the terrain surface.

Normal Contraction Higher values will shrink the range over which the normals get blended.

Normal Threshold Lets you reduce the influence of the terrain normal according to the angle between geometry and terrain normal.

Surface Options

Culling Lets you specify if the shader shall cull back or front faces.

ZWrite Lets you choose whether the shader writes to the depth buffer or not. *When using "large" Depth Shift values you should set this to Off and assign an extra depth only material.*

Receive Shadows If unchecked the material will not receive shadows.

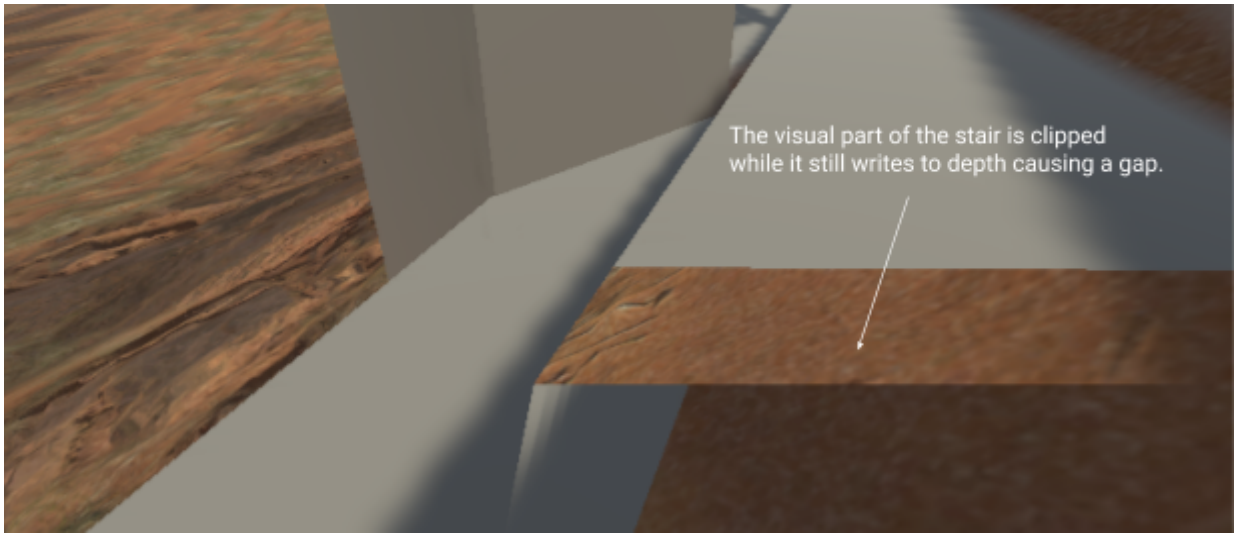
Surface Inputs

More or less common surface inputs. So i will not explain these in detail here.

Known Issues

As the effect works in view or clip space it changes according to the view angle. *I recommend using rather tight alpha blending ranges.*

Due to the fact that we push vertices towards the camera they will get clipped – while the vertices of the depth only draw still gets rendered. This may cause gaps if the camera gets really close to the surface like shown below. *You may improve this a bit by lowering the camera's near clipping plane but this will never solve it. So make sure that your camera never gets that close to the blended meshes.*



It is difficult to impossible to solve all shading artifacts. The screenshot below shows a foot which actually is clipped because of the depth written by the terrain. However as the stair blends with the terrain it looks as if the foot blends with the stair as well.

You can not stack blended meshes on top of blended objects tightly. Take the stairs for example: It is a single mesh which is fine as it gets drawn with a single call. If it was made out of 3 meshes — one for each step — you would get a lot of shading artifacts depending on the order they were rendered.

As you will most likely set the blend shader to not write to depth and use a depth only material instead you may get some artifacts on your blended mesh where parts of faces lying in the background poke through. Large, smooth and rather convex meshes do not suffer from this.

Versatile Blend Shader (Experimental)

The versatile blend shader allows you to smoothly blend meshes with *any* other opaque geometry in your scene. So compared to the terrain blend shader it does not only blend with the terrain.

It shares the principle idea of the *Terrain Blend* shader and lets you pull vertices slightly towards the camera so that pixels which are actually below the pixels already on screen will still pass the depth test and allow us to add the blend. *So i recommend to read the description of the Terrain Blend shader first.*

This flexibility however comes at some costs and drawbacks:

- The shader relies on the camera depth texture to determine the distance to the surrounding geometry (like e.g. soft particles do as well). As the camera depth texture is sampled in screen space the final blending is quite **view dependent**.
- In order to be able to sample a frame accurate camera depth texture, the shader has to be set to render queue = **transparent**. This will make Unity sort all objects using this shader back to front. So we will not benefit from early depth testing if several objects using this shader overlap. Regarding opaque objects occluding the blended objects early depth testing still works.
- ~~As transparent objects do not receive proper screen space shadows (from the directional light)~~ we have to sample and blur **shadows** from the original cascades and offset the shadow sampling position slightly to avoid shadows at the intersection: Using the real sampling positions of course would end in all blended pixels to be in shadow as they are supposed to be beneath the geometry they blend with.

The shader still is rather fast :)

Compared to a regular opaque shader It does an extra camera depth texture lookup and calculates the shadow cascades split per pixel ~~instead of per vertex.~~ Since URP 7.2. this is the behaviour of all shaders anyway…

Nevertheless I do not recommend to add this shader to any single stone you add to your scene. And if you have a house whose foundation you would like to blend nicely just make sure that the foundation uses its own material and only this material uses the versatile blend shader.

Unlike the *Terrain Blend* shader it renders on the transparent queue. This means that its tweaks to the depth buffer only affect objects which render afterwards - namely all other transparent materials. So clipping issues as caused by the *Terrain Blend* shader are kept to a minimum by design and we do not have to add an additional material just to write proper depth. Anyway: Using high *Depth Shift* values may introduce artifacts.

Orthographic projection is not supported. Does it make sense? *Just let me know.*

Shader Inputs

Surface Blending

> **Depth Shift** Amount the vertices get shifted towards the camera. The more you shift the more pixels below the surface the geometry intersects with will get visible. So this defines the maximum zone you may blend within.
> *Nevertheless: Keep this rather small to avoid shading artifacts.*

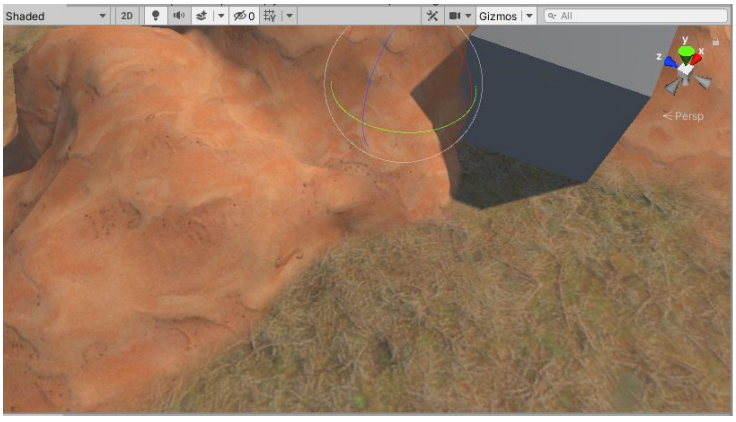
Alpha Shift Lets you shift the start of the blending range above or below the intersecting surface.
This will affect the shadows sampling offset too. So keep an eye on the shadows when tweaking this. High values will influence self shadowing and most likely eliminate too many shadows ...

Alpha Contraction Higher values will shrink the range over which the alpha gets blended resulting in a sharper transition.

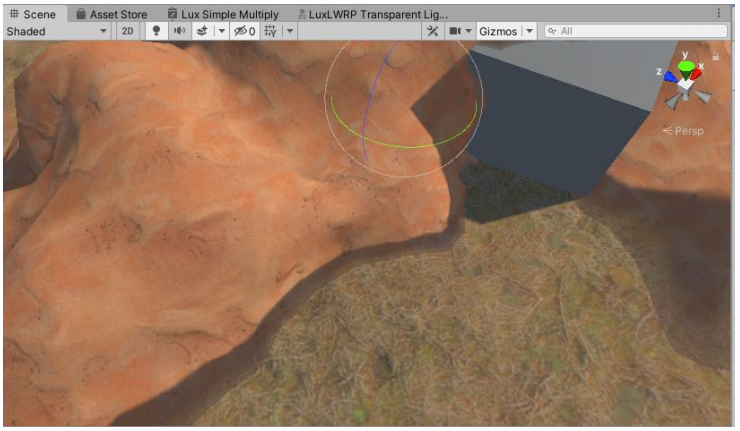
URP >= 7.2.

Shadow Shift Threshold In order prevent the scene already on screen from shadowing the parts of the blended mesh below the scene we have to lift the position in world space the shadows are sampled at. *Shadow Shift Threshold* defines the distance to the scene's on screen surface in cm where this lifting kicks in. As we can not 100% precisely reconstruct the scene's height values around 0.05 (= 5cm) should be fine in most cases.

Shadow Shift Factor which determines the final lift of the shadow sampling position. Usually a value of 1.0 should be fine.



Proper settings: We get a nice blending.



Improper settings: The blended pixels are shadowed by the geometry they intersect with because the *Alpha Shift* value and/or *Alpha Contraction* are too high and *Shadow Shift Threshold* and *Shadow Shift* are not set properly.

I admit: it is a pain to adjust the settings… and I have to work on them.

Surface Options

Regular surface options you know from other shaders. ZWrite should be set to on tho.

Surface Inputs

Regular surface inputs and nothing fancy here: Right now the shader only supports albedo/smoothness and normal textures.

Known Issues

Blending objects with other objects also using the *Versatile Blend* shader is not supported.

Sometimes shadows vanish when you switch your pipeline settings to *"No Cascades"*, which actually does not make much sense… Going forth and back between *"No Cascades" and "Two Cascades" fixes this for me.*

Make sure you check shadows from directional and spot lights.

Skin Shader

The skin shader uses pre-integrated skin lighting based on the work of Eric Penner and adds transmission from a static thickness map on top.

Diffuse and specular lighting use different normals giving you smooth diffuse lighting while keeping all details when it comes to specular highlights.

Using a skin mask it allows you to mix standard lighting and skin lighting within a single material. You can also fade out skin lighting over distance.

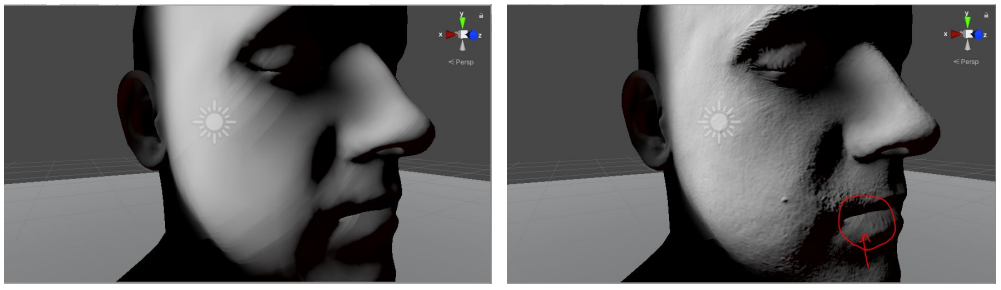
In order to make it match the *Lux LWRP Lit Extended* shader it supports (animated) rim lighting and all stencil buffer features.

Shader Inputs

Surface Options

Receive Shadows If unchecked the material will not receive shadows *(faster)*.

Due to the smooth diffuse normals skin is prone to self shadowing artifacts. Usually Unity shifts the vertices along the surface normals in the shadow caster pass according to the Normal Bias of the light. This may not always look well, especially if you use smooth normals. Harsh normals would hide most artifacts as shown in the screenshots below.



So the properties below allow you to finetune self shadowing.

Shadow Caster Bias Lets you determine the amount of "mesh inflation" when it comes to the shadow caster pass. Unity default is 1.0. *This effects how the skin casts shadows.*

Shadow Sample Bias Lets you shift the position where shadows are sampled along the surface normals. A value of 0.01 would shift the position by 1 cm. *This effects how the skin samples shadows.*

Surface Inputs

Albedo (RGB) Smoothness (A) Albedo texture in RGB, Smoothness in Alpha.

Smoothness Lets you remap the sampled smoothness value.

Specular Specular color.

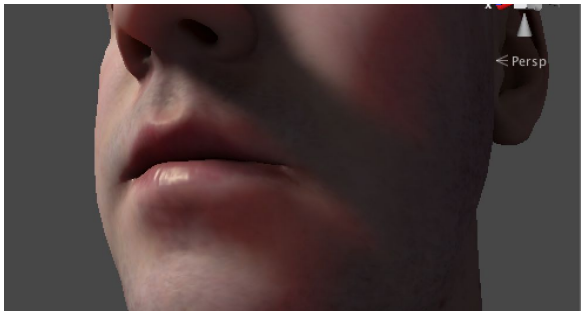
Enable Normal Map If checked the shader will sample the assigned normal map.

Normal Map The normal map.

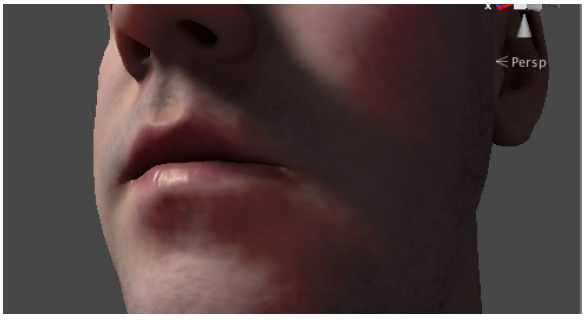
Normal Scale Scale of the sampled normal.

Enable Diffuse Normal If checked the shader will do a 2nd sample of the assigned normal map and blur it according to the chosen *Bias* value.

Bias Amount of blur added to the sampled diffuse normal (by sampling a lower mip level) In case you want to use subtle blur amounts the normal map must be set to *Filter Mode = Trilinear.*



***Enable Diffuse Normal Sample* disabled:**
The shader will simply use the vertex normals when it comes to diffuse lighting so diffuse lighting will not show up any details. This saves one texture lookup in the shader.



***Enable Diffuse Normal Sample* enabled:** The shader will sample a "blurred" version of the normal map according to the *Bias* and use it when it comes to diffuse lighting. Please note the subtle changes in lighting on the cheek or chin.

Skin lighting

Skin Mask (R) Thickness (G) Curvature (B) Occlusion (A) Texture which contains skin mask in red, thickness in green and ambient occlusion in alpha.

Skin Mask White pixels in the mask define the parts which shall be lit using the skin lighting. Black pixels define the parts where regular lighting shall be applied.

Thickness Bright pixels define thin parts where there will be a lot of subsurface scattering and transmission like on the nose or the ears.

Curvature Grayscale curvature map (optional).

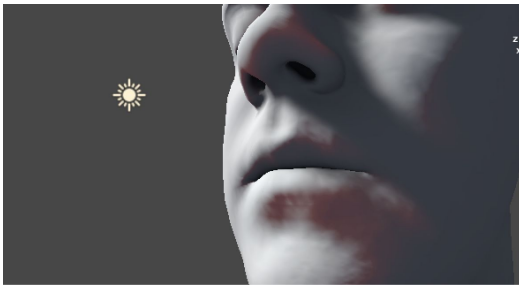
Occlusion Ambient occlusion map.

Please note: The texture should be imported as linear texture, so uncheck "sRGB (Color Texture)" in the import settings. *Compression* should be set to *High*.

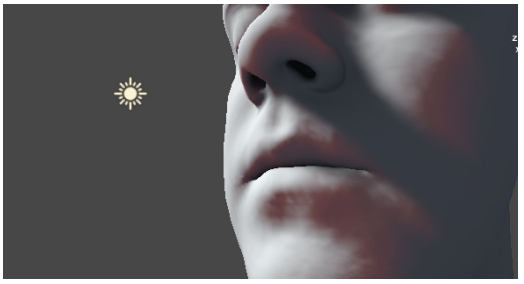
Sample Curvature In case this is checked the shader expects a *curvature map* in the blue channel of the *Skin Mask (R) Thickness (G) Curvature (B) Occlusion (A) texture.*

Curvature Drives the pre-integrated diffuse lighting and lets you control small scale light scattering. If *Sample Curvature* is checked this will be driven by the given curvature map. *Curvature* will act as multiplier. Otherwise the shader will take the given thickness map into account and *Curvature* will lerp the value from the thickness map towards 1.0.

Basically the pre-integrated diffuse lighting adds some kind of reddish tint (based on the Skin LUT) according to the dot product of the normal and the light direction to simulate the small scale light scattering within the human skin.



Curvature = 0 – rather subtle light scattering



Curvature = 0.3 – stronger light scattering.

Subsurface Color Color used to tint transmission.

Transmission Power Drives the view dependency of the transmission effect. Larger values will make it more view dependent.

Transmission Strength Lets you scale the transmission effect.

Shadow Strength As transmission might be totally eliminated by self shadowing this parameter lets you suppress shadows when it comes to transmission. Other lighting features (diffuse, specular) are not affected.

Mask by incoming shadow strength Lets you suppress transmission according to the shadow strength as set on the given light source - or from point lights which do not cast any shadows (here shadow strength is 0.0).

Transmission Distortion When calculating transmission the shader distorts the inverted light direction vector slightly by the given normal to simulate the scattering. Default value is 0.01. Higher values will give you more scattering and break up the uniform look.

Ambient Reflection Strength Lets you adjust the intensity of the ambient reflections.

Skin LUT Holds the results of the pre-integration and acts as a lookup table in the lighting function. For *caucasin* like human skin please use the provided LUT “DiffuseScatteringOnRing”.

Please note: This texture is absolutely mandatory. If it is missing lighting will look totally odd. But the shaderGUI should always load the default “DiffuseScatteringOnRing” texture from the Resources folder if the slot was empty.

Please note: If you want to assign your own LUT make sure that its *Wrap Mode* is set to *Clamp*. *Compression* should be *None*. The texture should be imported as linear texture, so uncheck “sRGB (Color Texture)” in the import settings.

Please note: The smaller the texture the faster its lookup will be. The default import size is set to 64px.

Distance Fading

Enable Distance Fade If checked the shader will fade out skin related lighting such as transmission and pre-integrated diffuse lighting. This lets you swap out the shader at lower LODs so you might use the built in "Lit" shader or even create a unified material for your characters.

Max Distance Determines the distance at which skin lighting will be fully faded out.

Fade Range Determines the fade range.

Please note: Both values are “manipulated” by the material editor. *Max Distance* maps to _DistanceFade.x and *Fade Range* to _DistanceFade.y where *_DistanceFade.x = Max Distance * Max Distance;* and *_DistanceFade.y = 0.1f / (Fade Range * Fade Range);*

Rim Lighting

Enable Rim Lighting Check to enable Rim Lighting

Rim Color The Rim Color

Rim Power Higher values will push the effect towards grazing angles only.

Rim Frequency Lets you add a simple sinus based animation. If set to values > 0.0 the shader will lerp between *Rim Power* and *Rim Min Power*.

Rim Min Power Second Power value used if Rim Frequency is > 0.0.

Rim Per Position Frequency Slightly offsets the animation based on the object’s pivot in worldspace and prevents objects all pulsating at the same frequency.

Advanced

Enable Specular Highlights If unchecked the shader does not calculate direct specular highlights which makes it cheaper.

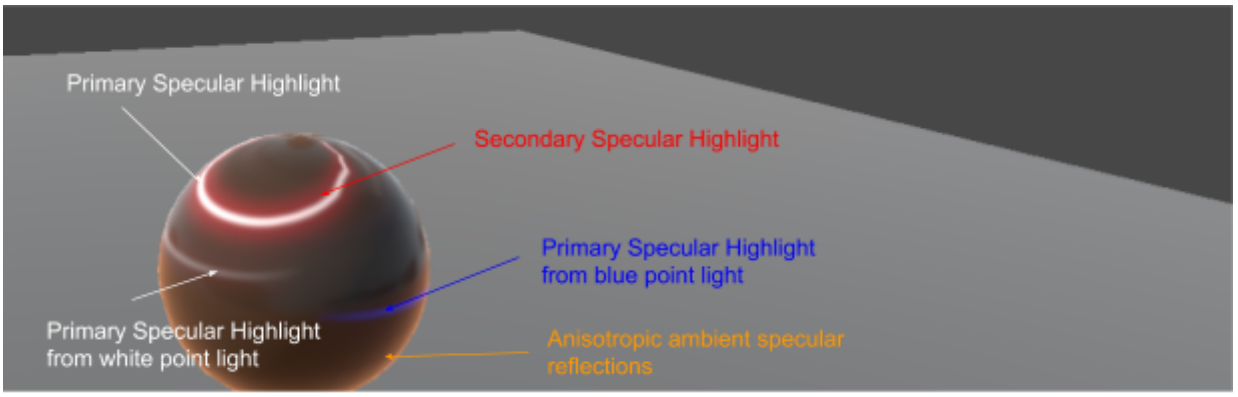
Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

Hair Shader

The hair shader supports hair specific Kajiya-Kay anisotropic lighting and multiple specular highlights: one representing the reflection at the outer shell of the hair fibre with the color of the light (primary) and another that represents the backscatter from the other side of the hair fibre's shell and is a mix of hair color and light color (secondary).

The shader is derived from Unity's implementation in the HDRP.

~~*It does not support "correct" backface lighting (VFACE) but always lits the surfaces according to the given geometry normal in order to achieve smooth shading.*~~



In order to make it match the *Lux LWRP Lit Extended* shader it supports (animated) rim lighting and all stencil buffer features.

The hair shader comes in two flavors: *Lux LWRP/Human/Hair* and *Lux LWRP/Human/Hair Blend*.

Lux LWRP/Human/Hair uses alpha testing and alpha to coverage (optional). It creates rather harsh edges (which gets softened/dithered if *Alpha To Coverage* is set to *On*) but writes to depth and receives proper shadows. Furthermore it does not cause any depth sorting issues.

Use this shader for hair which may not always be close to the head. Enabling MSAA will help to get nicer edges and is rather cheap on mobile devices nowadays.

Lux LWRP/Human/Hair Blend uses alpha blending and gives you nicer edges. However it does not write to depth (as it is a transparent shader), thus it will suffer from depth sorting issues.

Use this shader variant only on hair cards close to the skin, where proper depth sorting is not really required – such as eyebrows or beards.

You may also add a 2nd material using this shader in case you really need HD hair. Make sure that Depth Testing uses "Less" (not "LessEqual") to only render the blended material where there is no alpha tested hair already. This will save a lot of fill rate. Please have a look at the included demo to find out more.

Make sure you have checked "Alpha preserves coverage" in the import settings of the hair texture and play around with the "alpha cutout value" while you zoom in and out from the texture to make sure that hair does not clump nor fade out over the different mip levels. Keep the values below in the range between 0.1– 0.9, as 1.0 won't do anything.

Shader Inputs

Surface Options

ZTest Lets you tweak depth based face culling. In case you use two stacked materials for the hair, set it to "Less" for the blended version to save fill rate *(blended only)*.

Culling Lets you specify if the shader shall cull back or front faces.

Enable VFACE If Culling is set to off the shader will also render back faces – using the normals from the front faces. This will prevent back faces from getting too dark in case you have assigned smooth normal and the hair cards are very close to the head *(like the challenger's head in the demo)*. However if you have "hair tubes" you should enable VFACE to get proper lit back faces as otherwise they will be way too bright.

Shadow Culling Lets you specify if the shader shall cull back or front faces or none during the shadow caster pass. *You may set it to back to safe some fill rate.*

Alpha To Coverage In case your hair's alpha contains very little and fine details hair may just dither out over distance where lower mips are sampled. Try to work against this by checking "Alpha preserves coverage" in the import settings of the hair texture and playing around with the "alpha cutout value". Also try to adjust the contrast of the alpha channel. If none of this helps turn off *Alpha To Coverage*.

Receive Shadows If unchecked the material will not receive shadows *(faster)*.

Surface Inputs

Base Color Color which tints the sample from the albedo texture.

Secondary Color Secondary tint color. The shader lerps between base and secondary color based on vertex color alpha: Vertex color alpha = 1 → Base Color will be applied. *Use this feature to create brown hair with some gray strands (like on the Challenger in the demo) or to lerp from base to tip color.*

Albedo (RGB) Alpha (A) The Albedo (RGB) and the Opacity mask (Alpha)

Alpha Cutoff If the alpha channel contains different shades of gray instead of just black and white, you can manually determine the cutoff point by adjusting the slider.

Enable Normal Map If checked the shader will sample the assigned normal map.

Normal Map The normal map.

Normal Scale Scale of the sampled normal.

Enable secondary highlight If unchecked the shader will skip the secondary highlight, which makes it faster.

Shift (B) Occlusion (G) Texture which contains *Shift* values in the blue and *Ambient Occlusion* in the green color channel. The sampled *Shift* value breaks up the uniform look over hair patches as derived from the *Primary Specular Shift* and *Secondary Specular Shift*. *The texture should be imported as linear texture, so uncheck "sRGB (Color Texture)" in the import settings.*
Please note: The sampled shift value from the texture will be ***multiplied*** with the shift

values from the sliders *primary and secondary shift*. So if your blue color channel (shift) is black the sliders will not do anything.

Specular Specular color which drives fresnel on ambient reflections.

Smoothness Lets you remap *Primary* and *Secondary Smoothness*.
Ambient Reflections are calculated based on the Primary Smoothness.

Hair Lighting

Strand Direction Lets you choose between *Bitangent* (hair strands are laid out from top to bottom) and *Tangent* (hair strands are laid out left to right).

Primary Specular Shift Shift the primary specular highlight towards the hair tip

Primary Specular Tint Specular color used to calculate the primary reflection. Should be grayscale.

Primary Smoothness Smoothness used to calculate the primary reflection.

Enable Secondary Highlight If unchecked the shader will skip the secondary highlight, which makes it faster.

Secondary Specular Shift Shift the secondary specular highlight towards the hair root.

Secondary Specular Tint Specular color used to calculate the secondary reflection. Should somehow match the hair color.

Secondary Smoothness Smoothness used to calculate the secondary reflection.

Rim Transmission Intensity Lets you adjust the intensity of transmitted rim lighting.
This rim lighting has nothing to do with the Rim Lighting further down in the inspector.

Ambient Reflection Strength Lets you adjust the intensity of ambient specular reflections.

Rim Lighting

Enable Rim Lighting Check to enable Rim Lighting

Rim Color The Rim Color

Rim Power Higher values will push the effect towards grazing angles only.

Rim Frequency Lets you add a simple sinus based animation. If set to values > 0.0 the shader will lerp between *Rim Power* and *Rim Min Power*.

Rim Min Power Second Power value used if Rim Frequency is > 0.0.

Rim Per Position Frequency Slightly offsets the animation based on the object's pivot in worldspace and prevents objects all pulsating at the same frequency.

Advanced

***Enable Specular Highlights** is missing here, as it simply does not make any sense: If you disabled specular highlights you would not need a hair shader.*

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

Stencil

Stencil Reference The value to be compared against

Read Mask Bit mask which determines which bit will be read from.

Write Mask Bit mask which determines which bit will be written to.

Stencil Comparison The function used to compare the reference value to the current contents of the buffer.

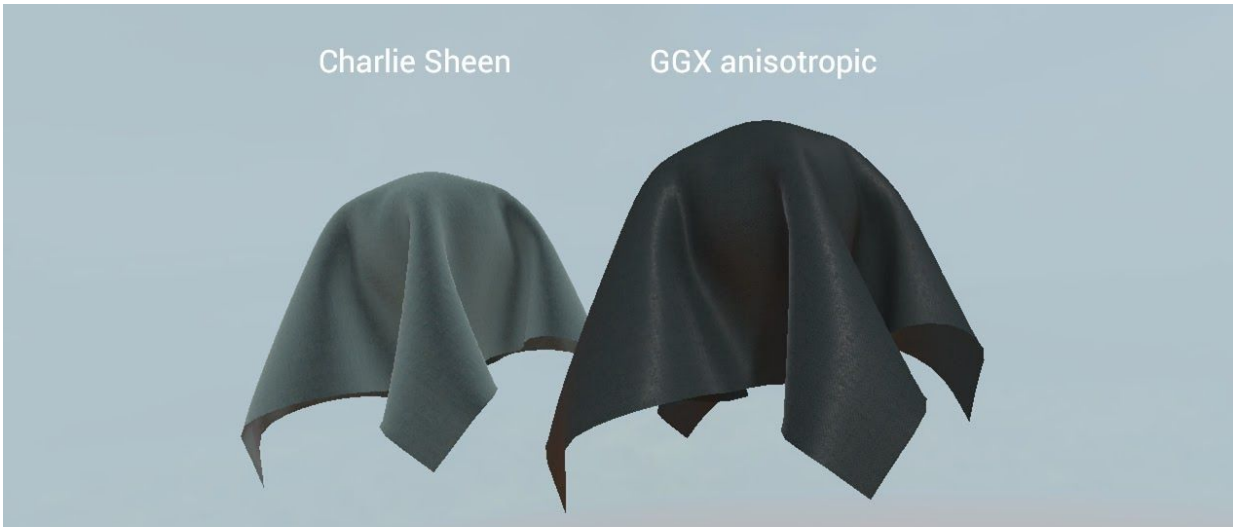
Stencil Pass Op What to do with the contents of the buffer if the stencil test (and the depth test) passes.

Stencil Fail Op What to do with the contents of the buffer if the stencil test fails.

Stencil Z Fail Op What to do with the contents of the buffer if the stencil test passes, but the depth test fails.

Cloth Shader

The cloth shader supports Charlie Sheen and GGX anisotropic lighting to cover a wide range of different fabrics and uses functions provided by the SRP Core.



Charlie Sheen lighting versus GGX anisotropic.
Please note that the GGX anisotropic material uses a darker albedo and less transmission.

The shader supports correct backface lighting using VFACE and transmission.

In order to make it match the *Lux LWRP Lit Extended* shader it supports (animated) rim lighting and all stencil buffer features.

Shader Inputs

Surface Options

Culling Lets you choose between rendering front faces, back faces or both.

Alpha Clipping If checked the shader will perform alpha clipping or alpha testing. *Enabling Alpha Clipping needs you to enable and assign the Mask Map as well, because alpha is stored in the Mask Map only.*

Threshold aka Alpha Cutoff: If *Alpha Clipping* is enabled this value determines where the alpha clipping starts.

Receive Shadows If unchecked the material will not receive shadows (faster).

Shadow Offset Lets you offset the casted shadows in order to prevent or minimize self shadowing.

Charlie Sheen Lighting

Enable Charlie Sheen Lighting If checked the shader will apply Charlie Sheen lighting which is suitable for materials like cotton or wool. If unchecked the shader will use GGX anisotropic lighting.

Sheen Color Color which tints the specular highlights.

GGX anisotropic lighting

If *Enable Charlie Sheen Lighting* is unchecked the shader will use GGX anisotropic lighting.

Anisotropy Controls the scale factor for anisotropy. 0 would be standard isotropic lighting, values smaller or greater 0 will shift the specular highlights according to the tangent or bitangent.

Transmission

Enable Transmission Check this to enable transmission.

Power Determines view dependency. Higher values will make transmission kick in only when the view ray more or less directly points towards the light source.

Strength Lets you scale transmission

Shadow Strength As transmission might be totally eliminated by self shadowing this parameter lets you suppress shadows when it comes to transmission. Other lighting features (diffuse, specular) are not affected.

Distortion When calculating transmission the shader distorts the inverted light direction vector slightly by the given normal to simulate the scattering. Default value is 0.01. Higher values will give you more scattering and break up the uniform look.

The subsurface color will be derived from albedo.

Surface Inputs

Color Color which gets multiplied on top of the base texture sample.

Albedo (RGB) Smoothness (A) Base texture which contains albedo (RGB) and smoothness (A). Tiling and Offset will drive the sampling of this texture and the assigned normal map.

Smoothness Lets you scale down the sampled smoothness from the base texture .

Specular The specular color.

Enable Normal Map If checked the shader will sample the assigned normal map.

Normal Map The normal map.

Normal Scale Scale of the sampled normal.

Enable Mask Map If checked the shader will sample the assigned mask map below.

Thickness (G) Occlusion (B) Alpha (A) The mask map which contains thickness (G) ambient occlusion (B) opacity (A). Thickness is needed by transmission, opacity is only needed in case you check *Enable Alpha Testing*.

***Please note:** The mask map gets sampled using unique tiling and offset values. This way it can store thickness, occlusion and alpha for the entire mesh while albedo, smoothness and the normal will use a higher tiling.*

Occlusion Lets you dampen the sampled occlusion.

Enable Alpha Testing If checked the shader will perform alpha testing. Alpha Testing needs the mask map to be enabled as well.

Alpha Cutoff If the alpha channel contains different shades of gray instead of just black and white, you can manually determine the cutoff point by adjusting the slider.

Rim Lighting

Enable Rim Lighting Check to enable Rim Lighting

Rim Color The Rim Color

Rim Power Higher values will push the effect towards grazing angles only.

Rim Frequency Lets you add a simple sinus based animation. If set to values > 0.0 the shader will lerp between *Rim Power* and *Rim Min Power*.

Rim Min Power Second Power value used if Rim Frequency is > 0.0.

Rim Per Position Frequency Slightly offsets the animation based on the object's pivot in worldspace and prevents objects all pulsating at the same frequency.

Stencil

Stencil Reference The value to be compared against

Read Mask Bit mask which determines which bit will be read from.

Write Mask Bit mask which determines which bit will be written to.

Stencil Comparison The function used to compare the reference value to the current contents of the buffer.

Stencil Pass Op What to do with the contents of the buffer if the stencil test (and the depth test) passes.

Stencil Fail Op What to do with the contents of the buffer if the stencil test fails.

Stencil Z Fail Op What to do with the contents of the buffer if the stencil test passes, but the depth test fails.

Advanced

Enable Specular Highlights *is missing here, as it simply does not make any sense: If you disabled specular highlights you would not need a cloth shader.*

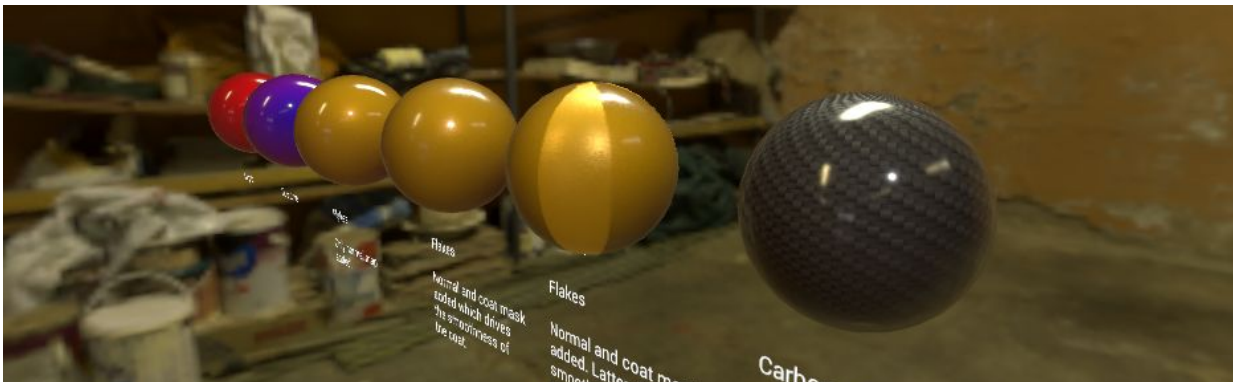
Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

Render Queue

Queue Offset Equals *Priority* in the standard Lit shader. As the material inspector sets the render queue based on *Enable Alpha Testing* you are not able to edit the *Render Queue* property at the bottom of the inspector manually. Use *Queue Offset* to adjust the final *Render Queue*.

Clear Coat Shader

Versatile clear coat shader that lets you use duo coloring, add metallic flakes and even mix clear coat with standard lighting within the same material.



In order to make it match the *Lux LWRP Lit Extended* shader it supports (animated) rim lighting and all stencil buffer features.

Please note: The clear coat shader effectively doubles the cost of specular computations.

Shader Inputs

Surface Options

Receive Shadows If unchecked the material will not receive shadows *(faster)*.

Clear Coat Inputs

Clear Coat Drives the thickness of the coat which influences the darkening of the albedo of the base layer towards grazing angles.

Clear Coat Smoothness Smoothness of the clear coat.

Clear Coat Specular Specular of the clear coat. Default is RGB(51,51,51). *Actually you should not really change it as some calculations are based upon the default value.*

Enable Coat Mask Map If checked the shader will sample the maske map.

Mask (G) Smoothness (A) Mask map which stores the clear coat mask in G and smoothness in A. The sampled mask value will be multiplied on top of the *Clear Coat* value and lets you adjust the thickness.

Enable Standard Lighting If checked and if *Enable Coat Mask Map* is checked as well the shader will fall back to standard lighting were the *Clear Coat* value (thickness) equals 0. *Expensive as the shader will use branching. It may however save you one more material.*

Base Layer Inputs

Color Primary albedo color.

Enable Secondary Color If checked the shader will mix *Color* and *Secondary Color* according to the view angle.

Secondary Color Color which will be applied at grazing angles.

Smoothness Smoothness of the base layer.

Metallic Metallness of the base layer.

Enable Normal Map If checked the shader will sample the normal map.

Normal Map The normal map.

Normal Scale Lets you scale the normal.

Enable Base Layer Mask Map If checked the shader will sample the maske map.

Metallic (R) Occlusion (G) Smoothness (A) Mask map which stores metallic in R, ambient occlusion in G and smoothness in A.

Occlusion Lets you scale the occlusion.

Enable secondary Reflection Sample If checked the shader will sample the given reflection probe twice: once for the coat layer and once for the base layer. By default reflections get only applied to the coat layer. *Expensive but worth it on certain materials such as coated carbon fibres.*

Rim Lighting

Enable Rim Lighting Check to enable Rim Lighting

Rim Color The Rim Color

Rim Power Higher values will push the effect towards grazing angles only.

Rim Frequency Lets you add a simple sinus based animation. If set to values > 0.0 the shader will lerp between *Rim Power* and *Rim Min Power*.

Rim Min Power Second Power value used if Rim Frequency is > 0.0.

Rim Per Position Frequency Slightly offsets the animation based on the object's pivot in worldspace and prevents objects all pulsating at the same frequency.

Stencil

Stencil Reference The value to be compared against

Read Mask Bit mask which determines which bits will be read from.

Write Mask Bit mask which determines which bits will be written to.

Stencil Comparison The function used to compare the reference value to the current contents of the buffer.

Stencil Pass Op What to do with the contents of the buffer if the stencil test (and the depth test) passes.

Stencil Fail Op What to do with the contents of the buffer if the stencil test fails.

Stencil Z Fail Op What to do with the contents of the buffer if the stencil test passes, but the depth test fails.

Advanced

***Enable Specular Highlights** is missing here, as it simply does not make any sense: If you disabled specular highlights you would not need a clear coat shader.*

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

Transmission Shader

Shader Inputs

Surface Options

Culling Lets you choose between rendering front faces, back faces or both.

Alpha Clipping If checked the shader will perform alpha clipping or alpha testing. *Enabling Alpha Clipping needs you to enable and assign the Mask Map as well, because alpha is stored in the Mask Map only.*

Threshold aka Alpha Cutoff: If *Alpha Clipping* is enabled this value determines where the alpha clipping starts.

Receive Shadows If unchecked the material will not receive shadows *(faster)*.

Shadow Offset Lets you offset the casted shadows in order to prevent or minimize self shadowing.

Surface Inputs

Color Color which gets multiplied on top of the base texture sample.

Albedo (RGB) Alpha (A) Base texture which contains albedo (RGB) and opacity (A). Tiling and Offset will drive the sampling of this texture and the assigned normal map.

Smoothness Smoothness or smoothness factor.

Enable Normal Map If checked the shader will sample the normal map.

Normal Map The normal map.

Normal Scale Lets you scale the normal.

Enable Mask Map If checked the shader will sample the mask map.

Mask (R) Thickness (G) Occlusion (B) Smoothness (A) Texture which stores a mask, thickness, occlusion and smoothness in the according channels.

Please note that mask in this case is not a metallic mask (the shader uses the specular setup). Instead it masks transmission lighting, so parts which are black in the mask will use standard lighting.

The mask map gets sampled using unique tiling and offset values. This way it can store mask, thickness, occlusion and smoothness for the entire mesh while albedo, alpha and the normal might use a higher tiling.

Transmission

Power Determines view dependency. Higher values will make transmission kick in only when the view ray more or less directly points towards the light source.

Strength Lets you scale transmission.

Shadow Strength As transmission might be totally eliminated by self shadowing this parameter lets you suppress shadows when it comes to transmission. Other lighting features (diffuse, specular) are not affected.

Mask by incoming shadow strength Lets you suppress transmission according to the shadow strength as set on the given light source - or from point lights which do not cast any shadows.

Distortion When calculating transmission the shader distorts the inverted light direction vector slightly by the given normal to simulate the scattering. Default value is 0.01. Higher values will give you more scattering and break up the uniform look.

The subsurface color will be derived from albedo.

Wrapped Lighting The shader will apply smooth wrapped around diffuse lighting. This value lets you adjust the light wrapping. Setting it to 0 will give you built in Lambert lighting.

Rim Lighting

Enable Rim Lighting Check to enable Rim Lighting

Rim Color The Rim Color

Rim Power Higher values will push the effect towards grazing angles only.

Rim Frequency Lets you add a simple sinus based animation. If set to values > 0.0 the shader will lerp between *Rim Power* and *Rim Min Power*.

Rim Min Power Second Power value used if Rim Frequency is > 0.0.

Rim Per Position Frequency Slightly offsets the animation based on the object's pivot in worldspace and prevents objects all pulsating at the same frequency.

Stencil

Stencil Reference The value to be compared against

Read Mask Bit mask which determines which bits will be read from.

Write Mask Bit mask which determines which bits will be written to.

Stencil Comparison The function used to compare the reference value to the current contents of the buffer.

Stencil Pass Op What to do with the contents of the buffer if the stencil test (and the depth test) passes.

Stencil Fail Op What to do with the contents of the buffer if the stencil test fails.

Stencil Z Fail Op What to do with the contents of the buffer if the stencil test passes, but the depth test fails.

Advanced

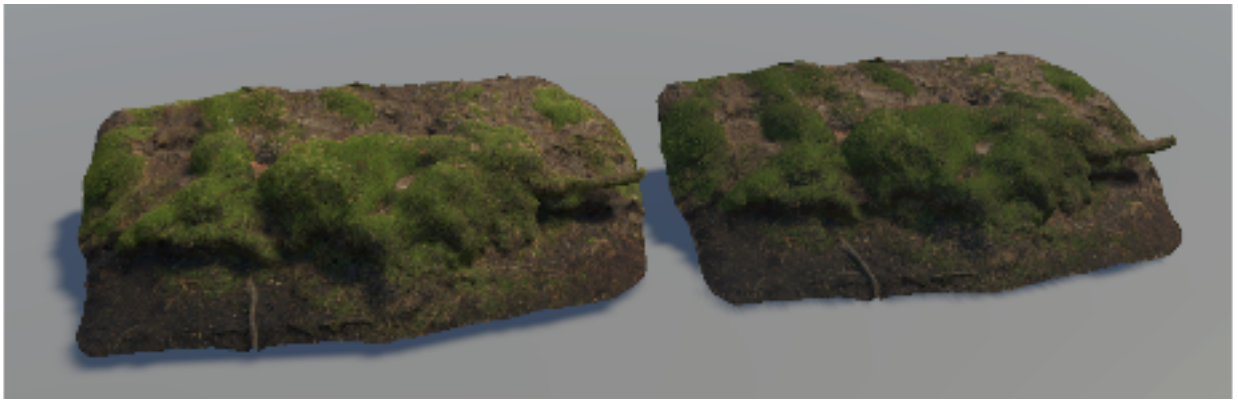
Enable Specular Highlights If unchecked the shader does not calculate direct specular highlights which makes it cheaper.

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

Fuzz Shader

The fuzz shader lets you create materials such as e.g. moss – materials which do not follow a standard microfacet brdf model because they are a lot more porous and show up transmitted light. It uses standard PBR lighting and adds fuzzy diffuse lighting on top. You even may add transmission. Specular lighting is not influenced by fuzz lighting.

Using a mask you are able to combine fuzzy moss and standard lit ground and wood within a single material.



Fuzz vs. standard lighting Fuzz lighting on the left, standard lighting on the right. The fuzz shader uses a mask to exclude non moss parts from fuzz lighting. Model and textures from Quixel.

Shader Inputs

Surface Options

Culling Lets you choose between rendering front faces, back faces or both.

Alpha Clipping If checked the shader will perform alpha clipping or alpha testing. *Enabling Alpha Clipping needs you to enable and assign the Mask Map as well, because alpha is stored in the Mask Map only.*

Threshold aka Alpha Cutoff: If *Alpha Clipping* is enabled this value determines where the alpha clipping starts.

Alpha To Coverage Lets you enable/disable *Alpha To Coverage* with dithers the alpha and produces less harsh edges than a simple cut off.

Receive Shadows If unchecked the material will not receive shadows *(faster)*.

Fuzz Lighting

Enable Fuzz Lighting If checked the shader will add simple fuzzy lighting which lets you create materials such as e.g. moss.

Diffuse Wrap The shader will apply smooth wrapped around diffuse lighting. This value lets you adjust the light wrapping. Setting it to 0 will give you built in Lambert lighting.

Fuzz Strength Fuzz color is derived from the albedo. So if the albedo is rather dark you may raise this multiplier.

Fuzz Power Determines how far the fuzz lighting travels according to the normals. The higher the value the less fuzz lighting will be applied.

Fuzz Bias Lets you add some fuzz lighting regardless of the given normal.

Ambient Strength Determines the Fuzz influence on ambient diffuse lighting.

Transmission

Enable Transmission On top of the fuzz lighting you may even add translucent lighting if needed.

Power Determines view dependency. Higher values will make transmission kick in only when the view ray more or less directly points towards the light source.

Strength Lets you scale transmission.

Shadow Strength As transmission might be totally eliminated by self shadowing this parameter lets you suppress shadows when it comes to transmission. Other lighting features (diffuse, specular) are not affected.

Distortion When calculating transmission the shader distorts the inverted light direction vector slightly by the given normal to simulate the scattering. Default value is 0.01. Higher values will give you more scattering and break up the uniform look.

The subsurface color will be derived from albedo.

Surface Inputs

Color Color which gets multiplied on top of the base texture sample.

Albedo (RGB) Smoothness (A) Base texture which contains albedo (RGB) and smoothness (A). Tiling and Offset will drive the sampling of this texture and the assigned normal map.

Smoothness Smoothness or smoothness factor.

Specular The specular color.

Enable Normal Map If checked the shader will sample the normal map.

Normal Map The normal map.

Normal Scale Lets you scale the normal.

Enable Mask Map If checked the shader will sample the assigned mask map.

FuzzMask (R) Thickness (G) Occlusion (B) Alpha (A) Texture which stores a fuzz mask, thickness, occlusion and alpha in the according channels.

The *FuzzMask* here lets you apply fuzz lighting only on certain parts of the model so you can easily mix standard and fuzz lighting.

Rim Lighting, Stencil and Advanced are described in other shaders already. So I won't go into details here.

Glass Shader

Limitations

The glass shader uses a non physically correct but eyeballed solution to simulate refraction.

As we do not use any fancy techniques such as ray tracing we can only refract what already is on screen. This means that a) we can only refract what actually is on screen already and b) will refract everything that is on screen.

a) If a refraction ray leaves the screen we will get heavy stretching on the refraction sample as the _CameraOpaqueTexture is clamped. This will most likely effect objects which are rather close to the screen border only.
In order to get rid of the stretching you may check *Enable Screen Edge Fade* which however will only make the refractions lerp towards the unrefracted sample.

b) By default the shader refract everything – even objects which are in front of the refracting surface.
In order to get rid of these you can check *Exclude Foreground* in the shader. Doing so however will add a lookup into the depth buffer (not free) and leave "holes" in the final refraction sample, as the shader simply falls back to the unrefracted sample.

Furthermore LWRP and URP do not support grabbing the framebuffer once for each draw – like the built-in render pipeline does. So we can not stack glass on top of glass: Glass in front of another glass will make the glass lying in the background fully vanish.

You will find help regarding issues using the glass shader and DOF here →

Alpha or transparency

Although the shader renders on the transparent queue, it actually is an "opaque" shader by design: Adding the refractions just overwrites anything in the background.

So alpha or transparency only control the mix between opaque parts plus the specular reflections and refraction.

Complex glass objects

Not being able to render "glass refracting glass" already makes a problem when it comes to a simple glass as – when looking at the bowl – you should see both outer and inner faces. However, only the outer faces get rendered (due to depth testing and writing). Disabling depth writing would lead to quite unpredictable results, because faces lying in the back may randomly overwrite faces lying in the front (depending only on the triangle order in the mesh).

So we have to split up the bowl into two materials: One used on the outer faces (Outer Material, the second one (Inner Material) used on the faces forming the inner parts of the bowl.

In case you want to use opaque "decals" on the outer faces of the bowl (like the glass with the Lux logo applied) you have to make the inner faces render first. Outer faces then get rendered on top using one of the two provided blending methods:

Inner Material

This material should render first.

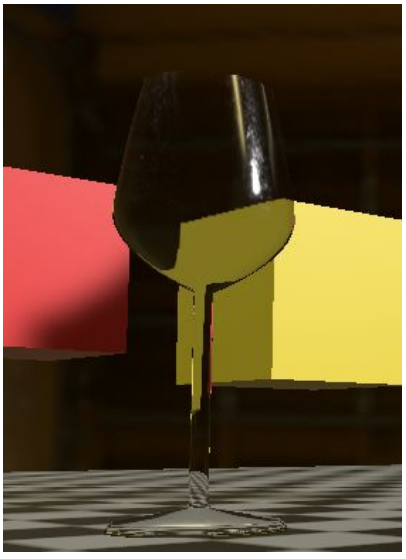
If *Culling* is set to *Back* we do not have to write to the Z Buffer as we do not have any overlapping front facing faces in a shape such as the bowl.

Outer Material

This material should render on top of the inner material. Do so by:
a) either tweaking the *Render Queue* to e.g. Transparent+1, *which is not ideal as it breaks proper back to front sorting...*
b) or make sure that the order of the materials (= submeshes) is setup properly from material which shall render first to the material which should render on top. *This keeps back to front sorting intact. As all materials can share the same Render Queue which should be: Transparent.*
As i am no expert in modelling i had to go with a), sorry.
Rendering the outer material on top of the inner material will make sure that any "decal" defined in the base map (Albedo Alpha) will fully overwrite the inner faces and thus not receive any glass specific lighting.

Using *Alpha Blending* will only reveal opaque parts and specular reflections if *Final Alpha* is set to 0. Raising *Final Alpha* will also reveal the refraction.

Using *Additive Blending* will reveal the inner faces even more – but of course will corrupt the final shading as the bowl and the refractions get too bright (additive!). Adjust the *Final Alpha* to reduce these artifacts.



Glass using one single material for the bowl Backfaces of the bowl are invisible.

Glass using two materials for the bowl Backfaces of the bowl are visible. The Outer Material uses *Transparent Blending* and a *Final Alpha* value of 0.4.

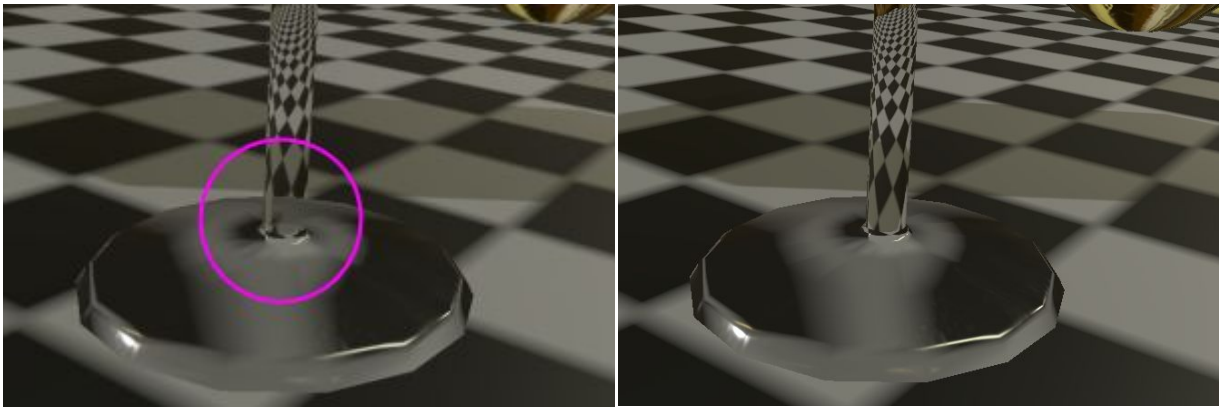
Glass using two materials for the bowl Backfaces of the bowl are visible. The Outer Material uses *Transparent Blending* and a *Final Alpha* value of 0.4. Base Map is enabled and adds tinting to the glass and the "Lux" logo decal.

In case you do **NOT** want to use opaque "decals" on the outer faces of the bowl you may swap the settings of the outer and inner material and render the outer material first. This way you

will ensure that the refraction on the outer material will be fully tinted. Otherwise it depends on the chosen *Final Alpha* value.

The importance of enabling ZWrite

The following screenshots demonstrate the importance of writing to the depth buffer:



ZWrite disabled Stem gets party overwritten by the foot. **Zwrite enabled** Stem is rendered properly.

Shader Inputs

Surface Options

ZWrite Lets you choose whether the shader writes to the depth buffer or not. Usually it should write to the depth buffer. In case you use multiple materials for certain parts of you glass object you may want want to set it to *Off*.

Culling Lets you choose between rendering front faces, back faces or both.

Blending

None Will make the shader simply overwrite the current frame buffer value. Use this for simple objects only using one glass material.

Transparent Will make the shader blend the shaded pixel with the frame buffer using “traditional transparency” or alpha blending.

Additive Will make the shader blend the shaded pixel with the frame buffer additively.

Final Alpha If blending is activated this lets you adjust how much the given pass will contribute to the final image. It uses a max operator, so it will lift the visibility of parts (like refractions), which do not have a corresponding alpha value.

Receive Shadows If unchecked the material will not receive shadows *(faster)*.
~~**Please note:** As glass does not contribute to the screen space shadows it may not receive proper shadows but simply pick them up from the underlying geometry (which may be very far away…).~~

~~In order to receive proper shadows you have to render an invisible object using the "Depth Only" shader. Please have a look at the "Glass Demo" where you will find a proper example.~~ Since URP 7.2. There are no screen space shadows any more.

Surface Inputs

Color (RGB) Alpha (A) Color which tints the refractions and opaque parts.
Alpha in this case does not drive the opacity of the material but the fade between glass lighting and standard lighting as used by the decals. In order to get fully transparent glass, alpha must be set to 0.0.

Enable Base Map If checked the shader will sample the base map.

Albedo (RGB) Alpha (A) Albedo (RGB) allows you to achieve multi colored glass or add decals. **Alpha** in this case does not drive the opacity of the material but the fade between glass lighting and standard lighting as used by the decals. In order to get fully transparent glass, alpha must be set to 0.0.

Tiling and Offset of this texture will drive sampling of the mask and bump map as well – even if it is disabled.

Smoothness Smoothness of the glass layer.

Smoothness Opaque Smoothness of the opaque parts as defined by alpha.

Specular Specular color of the glass.

Specular Opaque Specular color of the opaque parts.

Enable Mask Map If check the shader will sample the mask map.

Thickness Mask (R) Smoothness (A) Texture which contains the "invers" glass thickness (solid = 0, thin shell = 1) in the red color channel and the smoothness in the alpha channel. **Please note:** The texture should be imported as linear texture, so uncheck "sRGB (Color Texture)" in the import settings.

Enable Normal Map If checked the shader will sample the normal map.

Normal Map The normal map.

Normal Scale Lets you scale the normal.

Refraction

Enable geometric Refractions If checked the shader will refract the given background based on the normals of the geometry and the following settings:

Index of Refraction The material specific index of refraction which drives the amount of refraction. Glass has an IOR of roughly 1.5.

Thin Shell A solid glass sphere would use a value of 0.0, while a hollow sphere like a christmas ornament should use a value of 0.98. A value of 1.0 would give you no refraction. This parameter drives the final shape of the refraction.
See: Solid and Thin Shell Glass in the demo.

Enable Screen Edge Fade If checked the shader will fade out geometry driven refractions towards the screen edges to suppress stretchings.

Fade Width Lets you define a falloff used to fade out geometric driven refraction towards the edges of the screen.

Bump Distortion Distortion of the refraction based on the normal map.

Exclude Foreground If checked the shader will not refract objects lying in front of the given pixel but return a refraction sample from the actual screen position.
This will make the shader sample the _CameraDepthTexture (more expensive) and will cause holes in the refraction sample.

Rim Lighting

Enable Rim Lighting Check to enable Rim Lighting

Rim Color The Rim Color

Rim Power Higher values will push the effect towards grazing angles only.

Rim Frequency Lets you add a simple sinus based animation. If set to values > 0.0 the shader will lerp between *Rim Power* and *Rim Min Power*.

Rim Min Power Second Power value used if Rim Frequency is > 0.0.

Rim Per Position Frequency Slightly offsets the animation based on the object's pivot in worldspace and prevents objects all pulsating at the same frequency.

Stencil

Stencil Reference The value to be compared against.

Read Mask Bit mask which determines which bits will be read from.

Write Mask Bit mask which determines which bits will be written to.

Stencil Comparison The function used to compare the reference value to the current contents of the buffer.

Stencil Pass Op What to do with the contents of the buffer if the stencil test (and the depth test) passes.

Stencil Fail Op What to do with the contents of the buffer if the stencil test fails.

Stencil Z Fail Op What to do with the contents of the buffer if the stencil test passes, but the depth test fails.

Advanced

Enable Specular Highlights If unchecked the shader does not calculate direct specular highlights which makes it cheaper.

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

Lit Particles Shaders

The lit particle shaders are derived from the *LWRP Particle SimpleLit* shader and support Blinn Phong Lighting and all other options you may know from this shader. Lux LWRP lit particles shaders however enable all light types and add support for real time shadows and cheap transmission lighting next to some minor optimizations on top.

The particle shaders have been written with classic camera aligned alpha blended billboards in mind. Other billboard modes like *Horizontal Billboards* are only partly supported. Alpha tested or solid particles using a custom mesh are not supported but covered well with the built in shaders.

Additive blending like other blending modes is supported but most likely not suitable as it is mainly used to render emissive materials like flames which do not need to receive real time shadows.

Real Time Shadows

Classical billboarded particles are just flat surfaces pointing towards the camera. As they are transparent they do not write into the depth buffer and thus may not receive any screen space shadows from the sun. Furthermore Unity deactivated the support for additional lights in the built in particle shaders so they of course do not receive shadows from spot lights either.

URP 7.2. enabled shadows on particles, however these get always sampled per pixel.

Lux LWRP lit particles sample all shadows in world space, either per pixel – which may get pretty expensive – or per vertex.

Per pixel shadows suffer the most from the fact particles being a flat surface only while per vertex sampled shadows may already look smoother due to the interpolation from vertex to fragment shader.

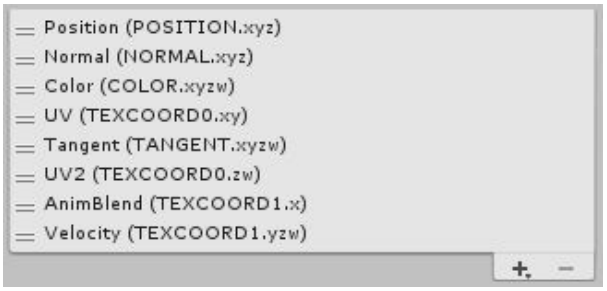
In order to make the per vertex sampled shadows even smoother the shaders offer two possibilities:

1. **Tessellation**: While being a bit expensive and not supported on all platforms this will just increase the number of vertices and thus give you more accurate shadows. Tessellation affects all light types and shadows.
2. Using **Sample Offset**: If this parameter is set to 0 the shader will only do a single look up into the shadow map of the directional light per vertex. As soon as you raise it, the shader will perform three lookups and mix the final result. The additional sample locations in world space are calculated according to the vertex's velocity and the chosen *Sample Offset*. ~~This only affects shadows from the main directional light.~~ In order to provide the shader with the *Velocity* you have to add the according vertex stream to your particle system.

Vertex Streams

Unity can output x? different combinations of vertex streams depending on their order and the selection. The shaders however only support two different vertex streams layouts when it comes to features such as *Flipbook Blending* and *Sample Offset*:

Supported Vertex Stream Layouts



Full feature Vertex Stream: Flipbook Blending and Sample Offset active If *Sample Offset* was deactivated the last stream *Velocity* would be obsolete.

Vertex Stream: Only Sample Offset active If *Sample Offset* was deactivated the last stream *Velocity* would be obsolete.

Of course if normal mapping is deactivated no *Tangent* stream is needed. Same goes with the *Color* in case your particle system did not write to any color – which however hardly ever happens.

In order to fix or check your streams you can use the button *"Check Vertex Streams"* at the very bottom of the material inspector. The particle system will also inform you about missing or obsolete streams.

Normal Direction

When it comes to particle lighting the *Normal Direction* of the particles as set in the *Renderer* module of the particle system plays an important role. Unity Docs >

A *Normal Direction* of 1 will give you quite accurate ambient lighting from the Skybox or a Gradient as the particle normals will more or less will fit the world normal of the particle's billboard. Per vertex lighting however might look a bit harsh. Values around 0.5 should be fine.

Shader Inputs

Tessellation *Tessellation shader only*

Tessellation Number of subdivisions created by the tessellator. Keep this number as low as possible for performance reasons. Make sure you always use an odd value (like 3, 5, 7, …) as even values will create odd tessellation.

Near (X) Far (Y) Near: Distance from the camera where the specified number of subdivisions will be reached. Far: Distance from the camera where no tessellation will be applied anymore.

Surface Options

ZTest Lets you tweak depth based face culling.

Cull Lets you choose between rendering front faces, back faces or both. *For classical billboarded particles "Back" is the right choice.*

Blending Mode

Alpha Will make the shader blend the shaded pixel with the frame buffer using "traditional transparency" or alpha blending.

Premultiply Will make the shader multiply alpha on top of the albedo. Usually needed for pbr lighting - which is not supported by the shaders.

Additive Blends particles using additively blending.

Multiply Blends particles using multiplicative blending.

Color Mode

Controls how the particle color (color added to vertex colors by the particle system) and Material color blend together.

Receive Shadows If checked the material will receive real time per pixel shadows from the main directional light.

Additional Light Shadows If checked the material will receive real time per pixel shadows from the additional spot lights.

Per Vertex Shadows If checked all shadows will be fetched per vertex instead of per pixel which speeds up rendering. Soft shadows will be disabled and smooth borders come from the vertex to fragment interpolation only.

Sample Offset If set to > 0.0 the shader will sample the shadow map three times for the main directional light and offset the sample position according to the given velocity of the particle and the chosen value. Needs *Velocity* to be added to the vertex streams of the particle system. See: *Vertex Streams* above.

Surface Inputs

Base Color

Base Map

Specular Highlights

Enable Spec Gloss Map

Spec Gloss Map

Specular Color (RGB) Shininess (A)

Enable Normal Map If checked the shader will sample the assigned normal map.

Normal Map The normal map.

Enable Emission If checked the shader will sample the Emission Map

Emission Map Emission texture (RGB)

Color Color which will multiplied on top of the sampled emission

Enable Transmission If checked the shader will add fast transmission lighting which is derived from the dot product between view vector and light vector and masked by (1.0 - alpha) so less opaque parts get more transmission.

Transmission Lets you scale the transmission.

Distortion Distorts the light vector by the given normal and breaks up the uniform transmission.

Particle Options

Enable Flipbook Blending If checked frames in a flip book will be blended together smoothly. Needs a 2nd *UV* coord and *Animblend* to added to the vertex stream. See: *Vertex Streams* above.

Enable Distortion Creates a distortion effect by making particles refract objects drawn before them.

Strength Controls how much the particle distorts the background

Blend Controls how visible the distortion effect is.

Enable Soft Particles If checked particles will smoothly fade out when intersection with other geometry in the depth buffer.

Near (X) Far (Y) Near: Distance from the other surface where the particle is fully transparent. Far: Distance from the other surface where the particle is fully opaque.

Enable Camera Fading Makes particles fade out close to the camera

Near (X) Far (Y) Near: Distance from the camera surface where the particle is fully transparent. Far: Distance from the camera where the particle is fully opaque.

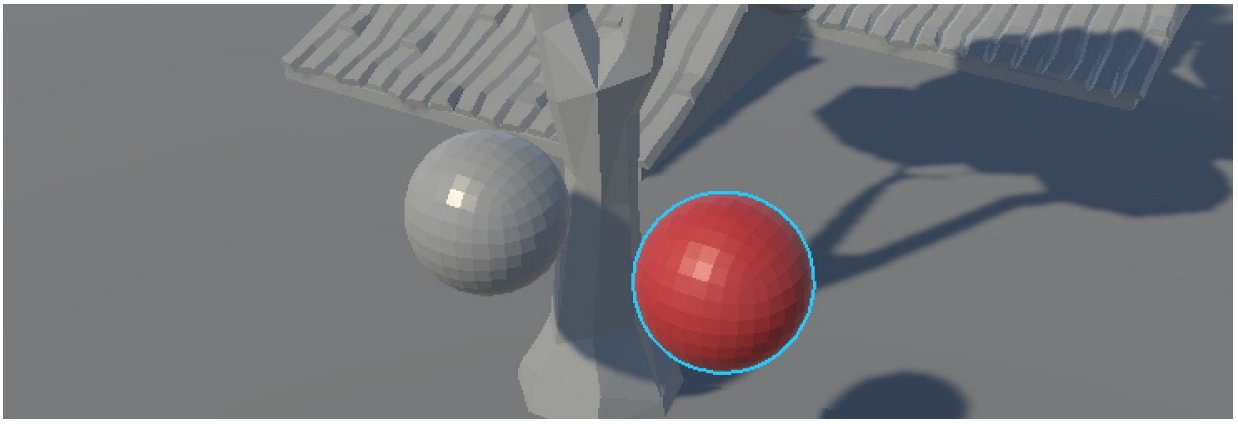
Check Vertex Streams Press this button to check your vertex streams. If all are fine the material editor will simply prompt a message into the Console. Otherwise you will get a message which renderers do not fit the material settings and the possibility to fix them.

Known Issues

- Unity's built in particles shaders render at render queue 3050 by default – for what reason ever. This will make them appear on top of any other transparent object which uses the standard transparent queue which is 3000. You may fix this by setting their *Priority* to *50* which will make them render at 3000 so they get regularly sorted from back to front respecting all other transparent materials.
- If spot lights shall casts shadows onto particle systems ALL visible spot lights have to cast shadows! I thought this was a bug in my shader… but in fact this is a bug or feature in LWRP (tested in unity 2019.1. and LWRP 5.16.1, 2019.2 and LWRP 6.9.1.) Using LWRP 6.9.1. it looks as if it was fixed in the scene view. In game view (play mode) shadows vanish as soon as there is one spot light without shadows visible on screen. On all objects – even the opaque ones using the built in shaders.

Flat Shading Shader

The flat shading shader allows your to apply flat shading to any mesh – regardless of its vertex normals: It simply skips these in the fragment shader and calculates the normals as used by the lighting function only based on the gradient of the world space position.



It is a simple HLSL equivalent to the included *Flat shading Node* for Shader Graph, so you will find in depth information here →

Unlike the node for Shade Graph this shader offers full stencil buffer support and rim lighting as well as normal mapping!? so you can use it along with the highlight shaders.

Shader Inputs

As this documentation is already pretty long i will not repeat common inputs here.

However special to flat shading is the fact that you might not assign a base map (Albedo (RGB) Alpha (A)) but fully rely on the base color and smoothness coming from a slider value.

Nevertheless you may activate *Alpha Clipping* or specify the alpha channel of the base texture to contain smoothness if *Enable Base Map* is checked.

Alpha Clipping and *Albedo Alpha contains Smoothness* are mutually exclusive which means that you can check only one. If both are checked *Albedo Alpha contains Smoothness* will disable *Alpha Clipping.* If *Enable Base Map* is unchecked both will be disabled.

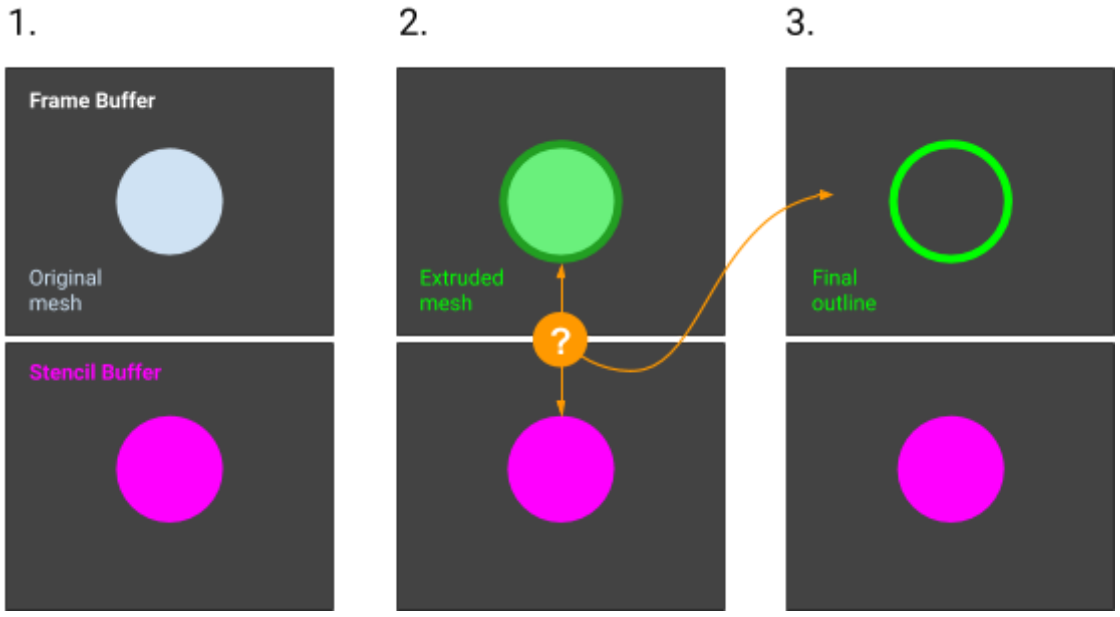
In case *Enable Base Map* is unchecked this shader should be slightly faster than the shader graph version.

Highlight Shaders

Lux URP/Fast Outline Shader

The Fast Outline shader allows you to highlight objects by adding a colored outline to them without having to use any expensive full screen image effect. The shader instead is based on the stencil buffer and needs the objects to be drawn twice:

1. First you render the object using a “regular” shader which renders your material just as is but also writes a reference value x into the stencil buffer. Do so by assigning e.g. the *Lux LWRP/Lit Extended Shader*
2. Then you render the object a second time just on top using the *Lux LWRP/Outline* shader. This will draw a slightly extruded version of the mesh taking the stencil buffer into account: It writes only to screen if the stencil buffer does not contain the reference value x.
3. All pixels covered by the regular mesh (where it wrote to the stencil buffer) will be excluded in the second pass giving you the desired outline.



Further information about the stencil buffer can be found in Unity’s documentation.

Using a method like this allows you to create various outline effects like the one shown below, just by editing the materials’ depth write property and setting the stencil options.

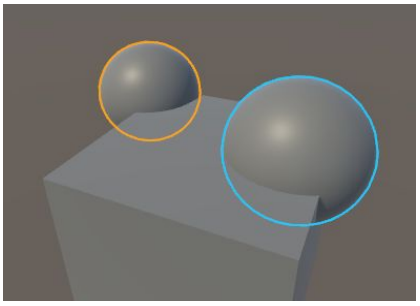
Outline Overview Demo

The *Outline Overview Demo* shows various outline effects and how to set these up.

Basic Outline

The outline will always be drawn regardless if the object is (partly) visible or not.

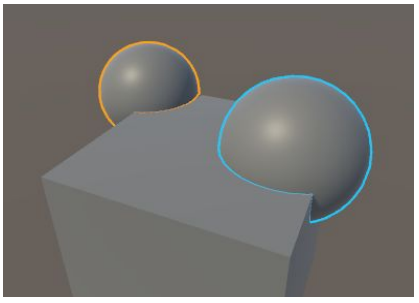
- Base Material simply writes to the stencil buffer, always.
- Outline Material uses Stencil Comparison = NotEqual and does not do depth testing: ZTest = Always



Depth Culling

Only the visible parts of the objects will be outlined.

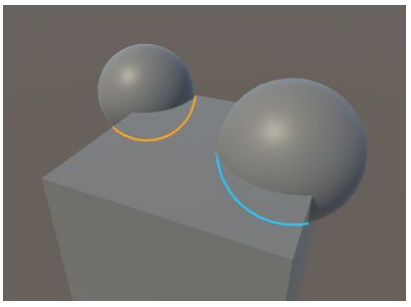
- Base Material simply writes to the stencil buffer.
- Outline Material uses *Stencil Comparison = NotEqual* and does regular depth testing: *ZTest = LessEqual*.



Depth Culling inverted

Only the invisible parts of the objects will be outlined.

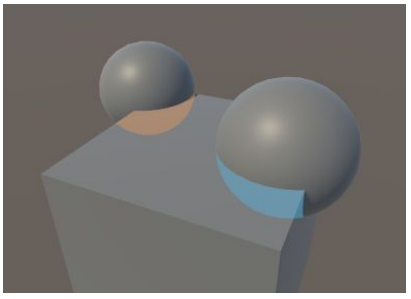
- Base Material simply writes to the stencil buffer
- Outline Material uses *Stencil Comparison = NotEqual* and does inverted depth testing: *ZTest = GreaterEqual*



Hidden Surfaces

Invisible parts of the objects will be drawn as flat, tinted surfaces.

- Actually this effect does not need any stencil buffer usage as we do not draw any outline but just the “regular” faces.
- So the Outline Material simply is set to: *Ztest = Greater*.

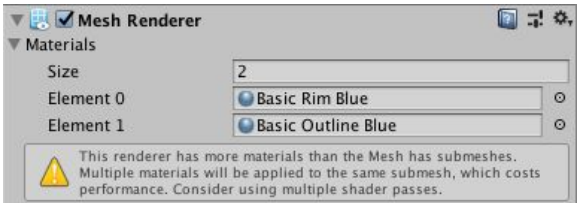


Setup

As mentioned before: The object or mesh has to be drawn twice. The easiest way to do this is just to apply two materials to the renderer component – in case your mesh does not have any submeshes/already uses multiple materials. Letter would actually need you to copy the mesh, assign proper outline materials and let unity do the rest. Or draw it from script.

In order to keep things simple, let's assume that out mesh has only one material assigned.

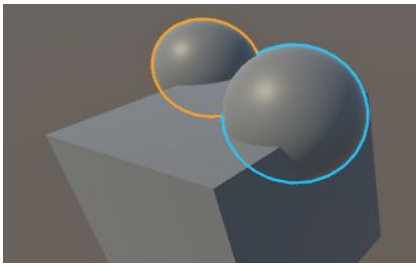
- Duplicate the material in the project tab.
- Assign the *Lux LWRP Lit Extended* shader to the new material. It is a copy of the built in Lit shader but offers some extra features such as the configuration of the stencil. And as the base material has to prepare the stencil buffer we will need such features.
- In the foldout *Surface Option* find *Advanced Options* and set *Stencil Reference* to *1*, set *Stencil Comparison* to *Always*. This will make the shader always write 1 into the stencil buffer:
- Next create a new material and assign the Lux LWRP Outline shader.
- Set its *Stencel Reference* to 1 as well.
- Now select Mesh Renderer component of the game object you want to add the outline to and set *Material → Size* to 2. Assign the new material using the *Lux LWRP Lit Extended* shader to the first slot and the outline material to the second slot.



- Done. Unity now should draw the object twice: First using the *Lux LWRP Lit Extended* shader and then using the outline shader.
- In case you mesh uses multiple materials this method will fail. Please have a look here to find out how to solve this.

Limitations

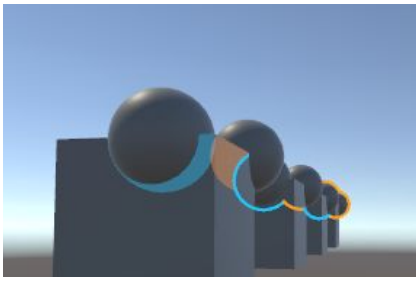
Overlapping outlines



As both spheres write 1 as stencil reference the blue outline of the sphere in the foreground will be partly suppressed by the sphere in the background.

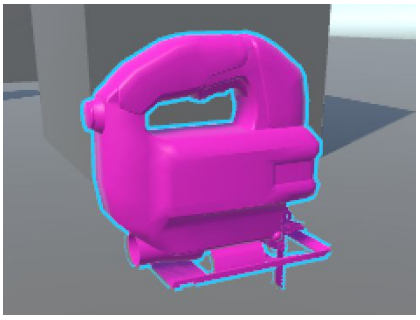
You would have to use different stencil reference values here which unfortunately breaks when using the SRP batcher right now.

Mixing outline modes



Mixing various outline modes within a single scene might lead to quite unpredictable results – especially if you mix *depth culling* and *depth culling inverted* or *Stencil Z Fail Op = Replace* and *Stencil Z Fail Op = Keep*.

Concave and complex models



The outline is generated by extruding the vertices along their normals. This works absolutely fine on convex objects – however may not always produce perfect outlines regarding complex and concave models and smoothed normals. The results are still pleasant.

Alpha testing

Alpha tested objects are not supported by design. Here e.g. using rim lighting would be a fast alternative to highlight the objects.

However you can have a look into the *Fast Outline AlphaTested shader* which uses another technique to create the outline.

In order to create an outline like in the scene view in the Unity editor you have to use a full screen image effect – expensive.

Outline Runtime Demo

Let's have a look if we can combine the outline effects (always visible and not hidden according to depth to highlight selected objects) and hidden surfaces (to always show certain objects): Here we have two concurring demands as both outlines and hidden surfaces shall always be visible regardless of depth… Do some funky stencil things?

Luckily we do not need a stencil test in order to draw hidden *surfaces*: A simple depth test done by the second material (that draws the hidden) surfaces is enough. So we will start with this material:

M Hidden Surfaces Base
Like said we do not need to write anything to the stencil buffer when drawing the regular mesh. So this material simply uses the built in Lit shader.

M Hidden Surfaces Surfaces
This material shall draw the hidden surfaces. Therefore we have to assign a material that let us choose how the depth test will be performed. We use the Lux LWRP Outline shader because of this and set *ZTest* to *Greater*. Assign this material as second material – done. Stencil tests are not needed.

We do not need a scriptable render pass or Renderer Features or anything else. We just have to make sure that the hidden surfaces are drawn after all opaque objects and most likely after all transparent objects as well. As the shader by default uses the *Queue = 3059* (which is Transparent+59) in order to get drawn on top of other transparent materials you may want to change it to *Transparent* in the material editor in case you want proper back to front sorting. In case you need front to back sorting consider using *AlphaTest*.

If you want to exclude hidden surfaces from being drawn behind selected objects you may active the stencil test and check against the reference value as written by the selected objects.

M Outline Base always visible
The selectables which shall always show an outline (when selected) however have to write to the stencil buffer due to the fact that we want to draw an outline. So this material uses the *Lux LWRP/Lit Extended* shader. The stencil buffer options are straight forward: we simply write our

reference value (1) under all conditions: So *Compare* is set to *Always* (which means no compare), all other operations use *Replace*.

M Outline Outline always visible

This material is used to draw the outline and checks the stencil buffer for the reference value (1). *Compare* is set to *NotEqual* so it only gets drawn on top of pixels if the corresponding stencil value is not 1.

Hidden surfaces

In case you want to use the outline shader to make hidden surfaces visible you can simply assign the second needed material permanently to your prefabs.

Selection outline

In this case we have to dynamically *change* the materials. Not selected objects should not use the base shader as it writes to the stencil buffer and would block outlines on other objects.

But we do not want to *change* materials at runtime as these most likely are *shared materials*. We have to *swap* materials on the desired game object/renderer as there might be several enemies or pickables using the same shared material.

The demo ships with two simple scripts (*MouseSelect.cs* and *ToggleOutlineSelection.cs*) that show how you can handle this without producing any garbage or spikes.

Usage

In order to add an outline effect select the *Renderer* component of your object, open the *Materials* foldout and set the *Size* to 2. Then assign an outline material as second material.

In case you mesh uses multiple materials this method will fail. Please have a look here to find out how to solve this.

Shader Inputs

Surface Options

ZTest Lets you tweak depth based face culling.

Culling Lets you choose between rendering front faces, back faces or both.

Stencil Reference The value to be compared against.

Read Mask Bit mask which determines which bits of the stencil buffer will be read from.

Stencil Comparison The function used to compare the reference value to the current contents of the buffer.

Outline

Color The color (RGB) and opacity (Alpha) of the outline

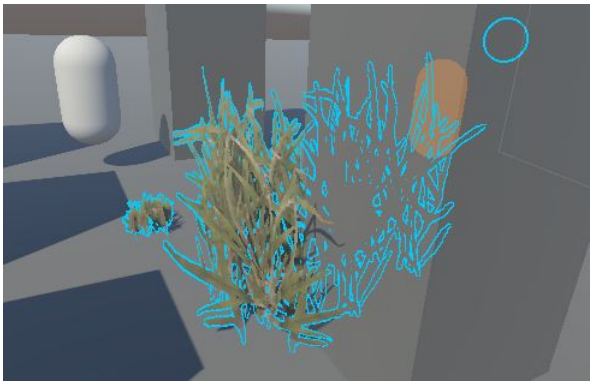
Width Width of the outline in pixels

Enable Fog If checked the outline will receive fog.

Lux URP/Fast Outline AlphaTested Shader

This shader lets you add a *rough* outline even to materials applying alpha testing.
It uses the stencil buffer as described above but instead of extruding the mesh to create the outline it samples the alpha texture 5 times and slightly shifts the uvs for each sample:

Combined, shifted alpha samples



Final result

As you can see we can create a really rough outline shifting the uvs. However the thicker the outline the more jaggy it will look. Furthermore, the thickness of the outline is dependent on view (anisotropic filtering) and distance to the camera (mip maps). Nevertheless it looks some kind of ok and is quite fast to render.

Just like the *Fast Outline shader* this shader needs you to draw the mesh twice:

1. Draw the mesh using a regular shader with alpha testing enabled. You may or may not write to the stencil buffer.
2. Then draw the mesh using the *Fast Outline AlphaTested* shader.

In order to draw the mesh twice just assign the two materials to the renderer component and ignore Unity's warning. *Actually we could draw the inner parts and the outline in a single pass but this would break some features like: only show the outline if hidden. So this shader did not make it into the package.*

Regarding the **stencil usage** please have a look at the chapter above and check out the included example in the *Outline Runtime Demo*.

Shader Inputs

Surface Options

ZTest Lets you tweak depth based face culling.

Culling Lets you choose between rendering front faces, back faces or both.

Alpha To Coverage As we use alpha testing MSAA will not smooth any edges. So you may turn on *Alpha To Coverage* to get some softer borders *(only visible in game view)*.

Stencil Reference The value to be compared against.

Read Mask Bit mask which determines which bits of the stencil buffer will be read from.

Stencil Comparison The function used to compare the reference value to the current contents of the buffer.

Outline

Color The color (RGB) and opacity (Alpha) of the outline

Width Width of the outline in pixels (roughly)

Surface Inputs

Color Color that gets multiplied on top of the albedo and alpha sample. *Here we are only interested in the alpha part of the color.*

Albedo (RGB) Alpha (A) Base texture containing allbedo in RGB and alpha in A. *Here we are only interested in alpha.*

Alpha Cutoff The cutoff value. *Please note: this should match the value used in the base material!*

Writing to the stencil buffer using the Lit Extended Shader

In case you want to draw outlines as described above, you need a special shader which prepares the stencil buffer accordingly. The *Lux Lit Extended* shader is derived from the built in one and offers the needed functionality. Details →

Adding stencil options to custom shaders

Shader graph currently does not support stencil operations. If you have a manually written shader: fine. If you use shader graph: Get the shader code (right click into the master node's header and select "Copy Shader". Create a new file in your IDE or text editor and paste the code).

Now that you have the shader code you have to a) add the needed properties and b) add the actual stencil operations.

a) In order to add the needed properties you can simply copy and paste the following lines to the property block of the shader:

```
[Header(Stencil)]
[Space(5)]
[IntRange] _Stencil ("Stencil Reference", Range (0, 255)) = 0
[IntRange] _ReadMask (" Read Mask", Range (0, 255)) = 255
[IntRange] _WriteMask (" Write Mask", Range (0, 255)) = 255
[Enum(UnityEngine.Rendering.CompareFunction)]
_StencilComp ("Stencil Comparison", Int) = 8 // always
[Enum(UnityEngine.Rendering.StencilOp)]
_StencilOp ("Stencil Operation", Int) = 0 // 0 = keep, 2 =
replace
[Enum(UnityEngine.Rendering.StencilOp)]
_StencilFail ("Stencil Fail Op", Int) = 0 // 0 = keep
```

```
[Enum(UnityEngine.Rendering.StencilOp)]
_StencilZFail ("Stencil ZFail Op", Int) = 0 // 0 = keep
```

b) At the beginning of the lighting pass you have to add the stencil operations:

```
Pass
{
	Name "ForwardLit"
	Tags{"LightMode" = "LightweightForward"}
	Stencil {
		Ref [_Stencil]
		ReadMask [_ReadMask]
		WriteMask [_WriteMask]
		Comp [_StencilComp]
		Pass [_StencilOp]
		Fail [_StencilFail]
		ZFail [_StencilZFail]
	}
```

Using Rim Lighting to highlight selected objects

In case you want to use *rim lighting* to highlight selected objects you can have a look at the *ToggleOutlineSelection.cs* and derive your script from this:

Basically you have to swap 2 materials: 1st with rim disabled, 2nd with rim enabled — as rim is compiled as "shader_feature_local" and can not be enabled at runtime.

As the 2nd material is referenced in the script the needed shader variant should be included in the build automatically. ***Please note:*** *I have not tested it. If the shader variant is missing in the build you could add an object somewhere in the scene (like below the terrain / behind the camera) and assign the material which has rim enabled.*

Another possibility is to enable rim by default and set its alpha value (Rim Color) to 0.0. Then set the proper alpha value to enable the rim lighting effect by script. This however will make the shader always do some calculations within the fragment shader even if rim is virtually disabled. *Tip: setting the frequency to 0.0 in the disabled state and setting it to the desired frequency when rim is enabled should make it a little bit faster.*

In case it is a **shared material** things get a bit more complicated. Just imagine you have 5 pickable flowers. Changing their *shared material* would tint all 5 instances although only one is selected. Changing the material will Unity make instantiate the shared material which produces a spike. So you can use *materialpropertyblocks* to only highlight the selected object.

The included *ToggleRimSelection* script uses this method. Just create 2 spheres, create a material using e.g. the *Lit Extended* shader, check *Enable Rim Lighting* and assign this material as well as the script to both of them. Hit play, then select one of the spheres and check/uncheck *Selected* in the script's inspector.

Toon outline shader

The toon outline shader is a rather simple shader which creates outlines by drawing the geometry a second time extruding the vertices along the geometry's normals.

Alpha tested materials or feathered outlines are not supported. These would require a full screen image effect.

Usage

In order to add a toon outline select the *Renderer* component of your object, open the *Materials* foldout and set the *Size* to 2. Then assign a toon outline material as second material.

In case you mesh uses multiple materials this method will fail. Please have a look here to find out how to solve this.

Shader Inputs

ZTest Lets you specify how the shader performs depth testing. Regular *LessEqual* should just be fine.

Culling Lets you specify if the shade shall cull back or front faces. I recommend to let it cull front faces.

Outline

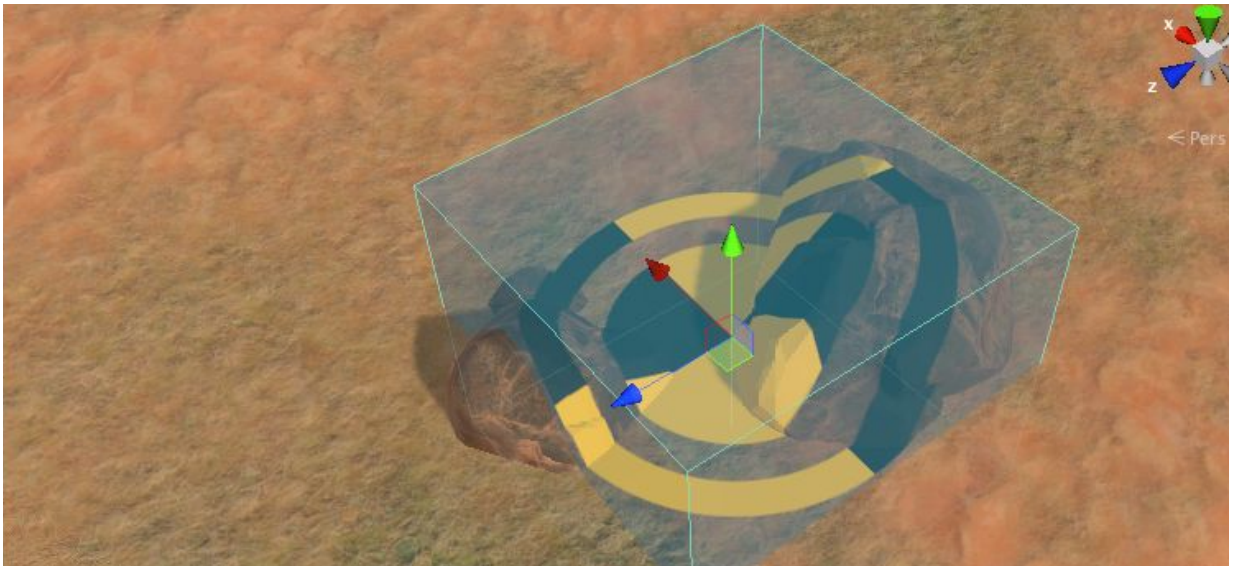
> **Color** Color of the outline. *You may set the alpha here as well as the shader uses alpha blending.*
>
> **Width** Width of the outline. *Depends on the scale of the object unless you check "Compensate Scale" or "Calculate width in screen space".*
>
> > **Compensate Scale** If checked the outline will have the same width regardless of the scale of the object. Otherwise the width will scale with the scale of the object. *Useful if you have several instances of the same mesh at different scales. In case you have complex hierarchies of skinned mesh renderers this may not produce the desired result tho as scale in this case is handled in a special way.*
> >
> > **Calculate width in screen space** If checked the shader will calculate the width of the outline in screen space and keep it stable over distance - just like the regular outline shader does. *However – this does not really look nice on a toon outline shader...*

Decals

Please note: Make sure your lightweight scriptable render pipeline settings have *Depth Texture* checked.

In case your camera uses **orthographic projection** you have to check *Enable Orthographic Support* in the material inspector.

Lux LWRP Essentials ship with two rather simple decal shaders which allow you to project decals on arbitrary geometry – just like you might know from deferred decals: If you draw a cube onto screen using a proper decal material then everything within the cube's volume will receive the decal:



Decal projected onto a mesh terrain and two rocks. The blueish volume marks the volume of the decal. The local up or y axis (green arrow) shows the decal's projection axis.

The decal shaders reconstruct the world position of the underlying geometry from the depth buffer and then calculate a corresponding decal UVs – according to the decal's projection axis.

Limitations

In case you use the *lit decal shader* the shader also reconstructs the underlying normals from the depth buffer. This leads to quantized normals and a common "flat shading" look:

Usage

- Make sure that you have checked *Depth Texture* in the *Scriptable Render Pipeline Settings* assigned under *Project Settings → Graphics*.
- Drag the *PF Decal Manager* into your Scene. This manager and the assigned script is only needed in the editor and lets you draw the decal gizmos which are needed to be able to select decals. Check/uncheck *Gizmos* to toggle the decal gizmos.
- Drag the *PF Decal* on top of any geometry in scene. This prefab contains a simple mesh renderer and uses the built in cube as decal mesh. It furthermore contains the Decal.cs which is only needed in editor and identifies the game object as decal so it will be found by the *DecalGizmos.cs* script (editor only as well). The script also offers the possibility to simply align the decal to the underlying normal by checking "Align" once.
- Move, scale and rotate the *PF Decal* to your needs.
- **Please note:**
    - The decal gets projected along its main axis which equals the local y-axis (green arrow). The decal texture will be stretched if the normal of the underlying surface is not aligned with the decal main axis.
    - Shading costs are determined by the size of the decal's volume. So try to make it as small as possible.
- Do **not** use decals on flat geometry as they would be too expensive. If you have flat geometry like a wall of floor use a simple quad or any other geometry and a transparent material instead.

Performance

Unlit decals of course are faster than lit decals.

In order to improve performance use layer based culling → so decals will be skipped at a distance of e.g. 50 meters (lightweight!). The decal shaders support distance based fading so you can hide any popping.
Decals in the Decal Demo are set to layer "Water" which gets culled at 50m according to the settings in "00 Wind and Settings" → LuxLWRP_LayerBasedCulling.cs

Do not check *Enable Orthographic Support* unless you need it.

Exclude objects from receiving decals

Using the Stencil Buffer

The decal shaders support stencil buffer operations and may be masked out on certain objects if these use a shader that writes to the stencil buffer (like provided by the *Lit Extended Shader*). So if the decal shader compares against the stencil reference value of 0 you may

simply create a material using the *Lit Extended Shader* which writes 1 to the stencil buffer and assign it e.g. to the player or your npcs.

Please have a look at the *Decals Demo* which uses two materials which are set up properly:

M Exclude Decals This material is used by the red capsules which shall not receive any decals. It uses the *Lit Extended Shader* and writes *Stencil Reference = 1* into the stencil buffer. *Stencil Comparison* is set to *Always* as we do not want the stencil buffer drive the rendering of the objects but rely on depth testing solely.

M Default Decal The default decal material. *Stencil Reference* is set to *0* and *Stencil Comparison* is set to *Equal* – so the decal will only be drawn if the stencil buffer contains 0.

Mesh Terrain and Top Down Projection Shader Excluding decals on materials using these shaders does not make much sense in my opinion. But in case you need variants of these shaders which allow you to write to the stencil buffer just let me know.

Using the Render Queue

Rendering certain objects after the decals have been drawn will automatically exclude these from receiving decals.

This method is used in the *Decals Demo* to prevent decals on grass and water:

Grass Shader The grass shader does not support stencil operations – so grass would always receive decals. In order to exclude grass you may however make the decal shader run before the grass shader by editing its render queue: Grass renders at *Render Queue = Alpha Test* (2450). So if you set the decal shader to e.g. *Render Queue = 2448* grass will not be affected by the decals.

Water Shader Like the grass shader the water shader does not support stencil operations. So in order to exclude water from receiving decals make sure the decals use a lower *Render Queue*. If you set the decal shader to *Render Queue < 3000* water will not be affected by the decals.

Decals on top of decals

Decals on top of decals are not automatically supported. And if you place decals on top of decals the sorting order might suddenly change according to the camera position.

In order receive predictable results stacked decals will have to use different *Render Queues* – like Render Queue = 2448 for the lower decal and Render Queue = 2449 for the upper one.

Decals and Outlines

Decals and outlines (objects highlighted using the outline shader) are somewhat tricky as both usually depend on the stencil buffer – especially when the outline always shall be visible.

In order to mix both i exposed *Write* and *Read Mask* in the stencil options. This lets us use the stencil buffer together with a bit mask. So you may think of stencil values like flags: If a certain bit is set render or do not render special features.

Outline material receiving decals

Stencil Shader – using *M Basic Lit Stencilwrite Receive Decals*

Stencil Reference	2	00000010	
Write Mask	2	00000010	
Stencil Buffer	2	00000010	

Outline Shader

Stencil Buffer	2	00000010	
Read Mask	2	00000010	
Final Ref Value	2	00000010	*The masked value equals the used Stencil Reference value in the Outline material. So the outline gets properly drawn.*

Decal Shader

Stencil Buffer	2	00000010	
Read Mask	1	00000001	*Only the lowest bit will be taken into account.*
Final Ref Value	0	00000000	*The masked value from the stencil buffer equals the used Stencil Reference value. So decals will appear on top.*

Outline material not receiving decals

Stencil Shader – using *M Basic Lit Stencilwrite Dont Receive Decals*

Stencil Reference	3	00000011	
Write Mask	3	00000011	
Stencil Buffer	3	00000011	

Outline Shader

Stencil Buffer	3	00000011	
Read Mask	2	00000010	
Final Ref Value	2	00000010	→ The masked value equals the used Stencil

Decal Shader

Stencil Buffer	3	00000011	
Read Mask	1	00000001	
Final Ref Value	0	00000001	*The masked value from the stencil buffer is greater than the used Stencil Reference value. So no decals will be drawn on top.*

Shader Inputs

Surface Options

Receive Shadows If unchecked the material will not receive shadows *(faster)*. *Lit only.*

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

Surface Inputs

Color Tints the color sample from the texture input (HDR).

Albedo (RGB) Alpha (A) Albedo in RGB and Alpha in A.

Smoothness Overall smoothness. Will be multiplied with Smoothness sampled from the *Mask Map* if it is enabled. *Lit only.*

Specular Specular color. *Lit only.*

Enable Normal Map Check this to sample the normal map. *Lit only.*

Normal Map Normal map. *Lit only.*

Normal Scale Scale of the sampled normal. *Lit only.*

Mask Map *Lit only.*

Enable Mask Map Check to make the shader sample the mask map.

Metallness (R) Occlusion (G) Emission (B) Smoothness (A) This combined texture contains masks/maps for all other features supported on the lit decals.
Please note: The texture should be imported as linear texture, so uncheck "sRGB (Color Texture)" in the import settings.

Emission Color Tint color for the emissive lighting.

Occlusion Lets you dampen the sampled occlusion.

Stencil

Stencil Reference The value to be compared against.

Read Mask Bit mask which determines which bit will be read from.

Write Mask Bit mask which determines which bit will be written to.

Stencil Comparison The function used to compare the reference value to the current contents of the buffer.

Advanced

Enable Fog If checked the decals will receive fog. *Unlit only.*

Enable Specular Highlights If unchecked the shader does not calculate direct specular highlights which makes it cheaper. *Lit only.*

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper. *Lit only.*

Billboard Shader

The billboard shader lets you create all kinds of camera facing billboards. It expects the default Unity quad as mesh input but should work with other geometry as well.

It collapse all vertices towards the pivot in the vertex shader and expands them in view space according to the uv coordinates.

Due to this dynamically or statically batched meshes will not work: Both methods will create one big mesh in world space with just a single pivot. GPU instancing works.

Shader Inputs

Surface Options

ZTest Lets you tweak depth based face culling.

Alpha Lest you choose between alpha testing and alpha blending. Alpha testing is needed to cast and receive real time shadows and writes to depth. Alpha blending creates way softer edges and is suitable for all kind of unlit billboards.

Threshold aka *Alpha Cutoff*: If *Alpha* is set to *Testing* this value determines where the alpha clipping starts.

Blending If *Alpha* is set to *Blending* you may choose between *Transparent* (traditional transparency based on the alpha channel in the *Albedo (RGB) Alpha (A)* texture), *Additive* (additive blending) and *SoftAdditive* (soft additive blending).
If *Alpha* is set to *Tested* the shader will always use traditional transparency and *Blending* will be ignored.

Receive Shadows If unchecked the material will not receive shadows (faster).

Billboard Shadow Offset Lets you offset the shadows to prevent odd self shadowing. *Please have a look at the "PF Billboard Sphere" which uses this feature.*

Billboard Options

Enable upright oriented Billboards If checked the shader will extrude the mesh along world y axis resulting in billboards such as from SpeedTree. By default the shader will create camera aligned billboards. ***Please note:*** *When using upright oriented billboards you may NOT rotate its transform as otherwise you will get odd result as the billboarding breaks. Moving and scaling is fine, of course.*

Set Pivot to Bottom Especially if *Enable upright oriented Billboards* is checked it might be helpful to expand to the billboard only upwards.

Expand X Lets you scale down the expansion of the billboard along the x axis. This feature is helpful to reduce overdraw and safe fill rate in case the billboard texture does not cover the entire geometry.
Please have a look at the "PF Billboard Spruce" which uses this feature.

Surface Inputs

Base Color Color which tints the sample from the albedo texture.

Albedo (RGB) Alpha (A) The Albedo (RGB) and the Opacity mask (Alpha)

Lighting

Enable Lighting If checked the shader will perform full pbr lighting. Please note: The *Normal Map* will always be sampled – as lighting with just the billboard normal does not make much sense.

Normal Map The normal map.

Normal Scale Scale of the sampled normal.

Smoothness Overall smoothness.

Specular The specular color.

Shadow Offset Lets you offset the shadows to prevent odd self shadowing.
Please have a look at the "PF Billboard Sphere" which uses this feature.

Fog

Enable Fog If checked the shader will apply fog.

Render Queue

Queue Offset Equals *Priority* in the standard Lit shader. As the material inspector sets the render queue based on chosen *Alpha* property you are not able to edit the *Render Queue* property at the bottom of the inspector manually. Use *Queue Offset* to adjust the final *Render Queue*.

Advanced

Enable Specular Highlights If unchecked the shader does not calculate direct specular highlights which makes it cheaper.

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

Volumetric Shaders

The volumetric shaders let you create fast volumetric light beams or simple box or sphere volumes.

Please note: All shaders need the *_CameraDepthTexture* to be available.

Light Beams

Light beams are based upon the old UDK implementation which still looks quite convincing and renders pretty fast.

Apart from the shader light beams need a proper mesh and two fall off textures:

- The mesh should be a double sided cone with uvs covering the entire 0-1 uv space. Proper uvs are important because the shader relies on calculations done in tangent space.
- The first fall off texture describes the light fall off along the light's forward direction
- The 2nd one will add the spot light fall off when looking from underneath.

Make sure the textures are set to Clamp in the import settings. And always use best quality.

Further information can be found in the UDK documentation →.

Shader inputs

Surface Options

ZTest Lets you tweak depth based face culling.

Culling Lets you specify if the shade shall cull back or front faces. Default is front faces whereas back faces may allow you to actually enter the light beam.

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

Surface Inputs

Color Color (RGB) and Opacity (A) of the light beam.

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 mask perpendicular to the beam in the green color channel. Red and blue should be set to 0 (black) to improve the texture quality.

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

Detail Noise

Enable detail noise If checked the shader will 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 control the scroll speed for the first sample whereas ZW control the scroll speed for the second sample.

Scene Fade

Near The distance between the intersection of the beam and the scene's geometry at which the beam shall start to fade in. Usually 0.0.

Soft Edge Factor Describes the feather between beam and scene geometry.

Camera Fade

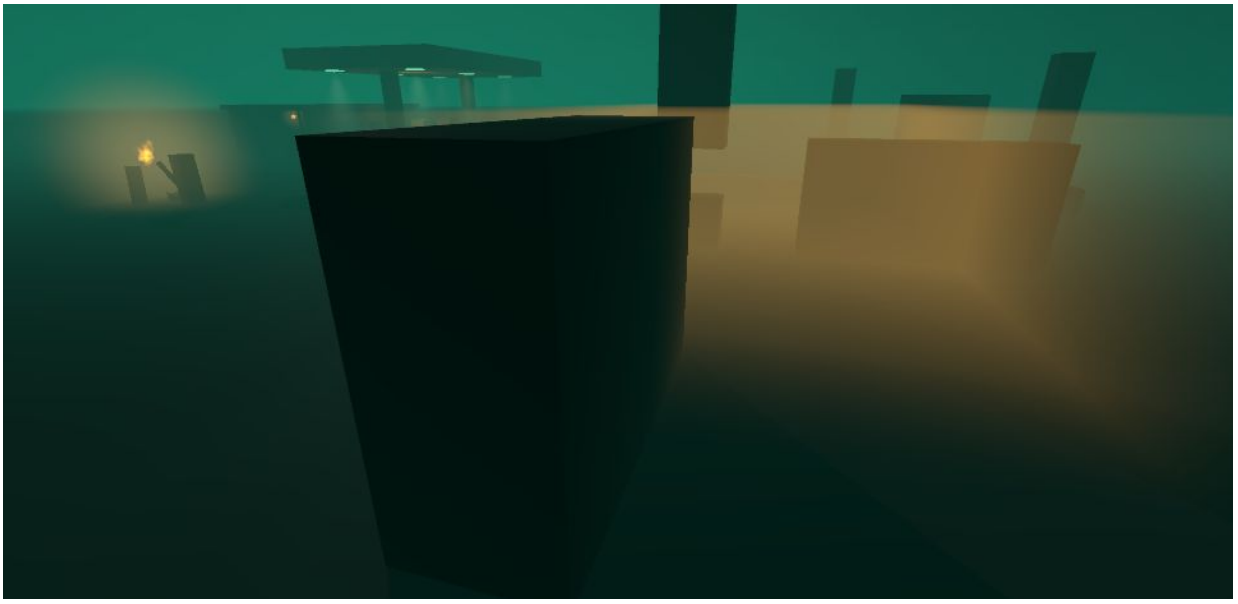
Near Dist Distance to camera at which the beam is fully transparent.

Far Dist Distance to camera at which the beam has reached its full opacity..

Limit Length Length in object space the beam will be drawn. *Although the package does not contain any script you could write your own one doing some simple raycasts to find out if the ray should be blocked by any geometry in the scene. If so you may lower the limit length to shorten the ray while keeping its shape.*

Render Queue The Render Queue is set to 3050 by default which matches the queue any transparent object using the built in Lit shader is rendered at. This ensures that transparent objects are properly rendered back to front. Nevertheless you may get sorting issues and may have to tweak the queue.

Box and Sphere Volumes



The box and sphere volume shaders both are rather simple and do not support any textures apart from simple gradients, which however may break the immersion as the shaders do not do any ray marching.

They allow you to create rather huge volumes which can be entered by the camera.

Please note: *If you are looking for some super cheap light halos please have a look at the billboard shaders.*

Default Settings

In order to support entering the volume the shaders render the back faces and skip the front faces *by default*. They furthermore ignore standard depth testing so that even if we are rendering the back faces the shader may create the desired effect on geometry which lies in front of the back faces.

Proper depth based occlusion is solely achieved by sampling the *_CamerDepthTexture* and tweaking the opacity accordingly.

Please note: For these reasons the shaders using the default settings do not benefit from early depth testing and all pixels will always be drawn.

This makes it very important to have your volumes properly adjusted and placed. It does not make any sense to stick them into the ground and create a huge overlap because this overlap will cause overdraw and cost fill rate. Box volumes may simply be scaled accordingly. When it comes the sphere volumes consider using a half sphere if you just need a common dome. The package ships with some proper sample meshes.

Optimize Rendering

In case your camera never enters a certain volume you can speed up rendering by setting *ZTest* to “LessEqual” and *Culling* to “Back”. This will activate early depth testing and the GPU will skip the fragment shader for all hidden pixels.

You may even think about swapping two materials based on the distance to the camera.

Box and Sphere Meshes

Box and sphere intersection calculations are done in object space and rely on a **unit box** and a **unit sphere**. Unity's built in box is just fine as mesh for the box volume shader. When it comes to the sphere volume shader consider using a simple sphere or half sphere mesh like provided with the package.

Shader Inputs

Surface Options

ZTest Lets you tweak depth based face culling. *Default is Always.*

Culling Lets you specify if the shade shall cull back or front faces. *Default is Back Faces which allows you to actually enter the volume.*

In case you never enter the volume you may set ZTest to "LessEqual" and Culling to "Back". Doing so will speed up rendering as the shader benefits from early depth testing.

Enable Orthographic Support In case your camera uses an orthographic perspective you have to enable this. *Experimental...*

Surface Inputs

Color Color (RGB) and Opacity (A) of the volume.

Enable Gradient If checked the shader will sample the assigned gradient (RGB).

Vertical Gradient (Box shader) Gradient texture applied vertically from bottom (left) to top (right).

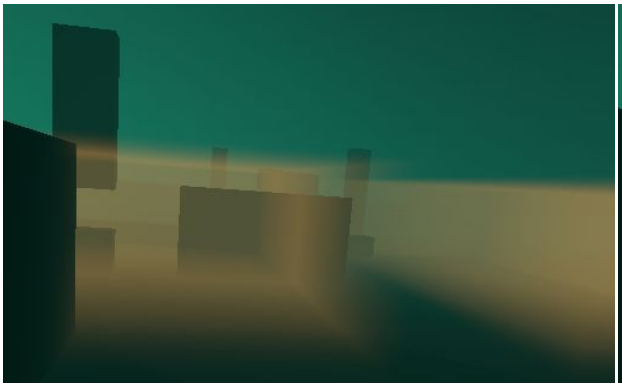
Thickness Gradient (Sphere shader) Gradient texture applied according to the thickness (way the view ray travels through the sphere volume).

Thickness Remap

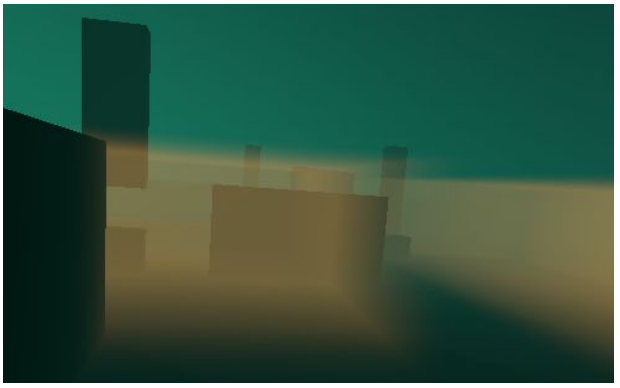
Lower and **Upper** act as lower and upper bound of a smoothstep function within the shader and let you remap the fall off.

Enable Fog In case you want the volumes to fade nicely with the rest of your scene you should check this.

HQ Fog If enabled the shader will recalculate the fog factor based on the result from the depth buffer instead of the clip space z position of the volume as provided by the vertex shader. This will give you the most accurate results but is *more expensive* and not really needed if your volumes a rather small.
The big box volume in the demo uses this feature while all other volumes do not.



HQ Fog disabled The cube in the middle of the volume receives fog as calculated at the backface of the volume which breaks the volumetric effect as parts of it receive too much fog.



HQ Fog enabled The cube in the middle of the volume blocks the fog and receives fog right at its surface which keeps the immersion intact.

Render Queue The Render Queue is set to 3050 by default which matches the queue any transparent object using the *built in Lit shader* is rendered at. This ensures that transparent objects are properly rendered back to front. Nevertheless you may get sorting issues and may have to tweak the queue.

Transparent objects within the volumes most likely will look odd anyway. Depending on their pivot and position within the volume they will get drawn:

- ***after** the volume (if they are closer to the camera) → they won't be affected at all.*
- ***before** the volume (if their pivot is further away than the volume's one) → they will be affected by the volumes depending on the closest opaque geometry behind them as the volume calculates opacity based on the depth buffer.*

However additively rendered particles like flames still may look correct - like shown with the torch light in the demo scene.

Custom Nodes for Shader Graph

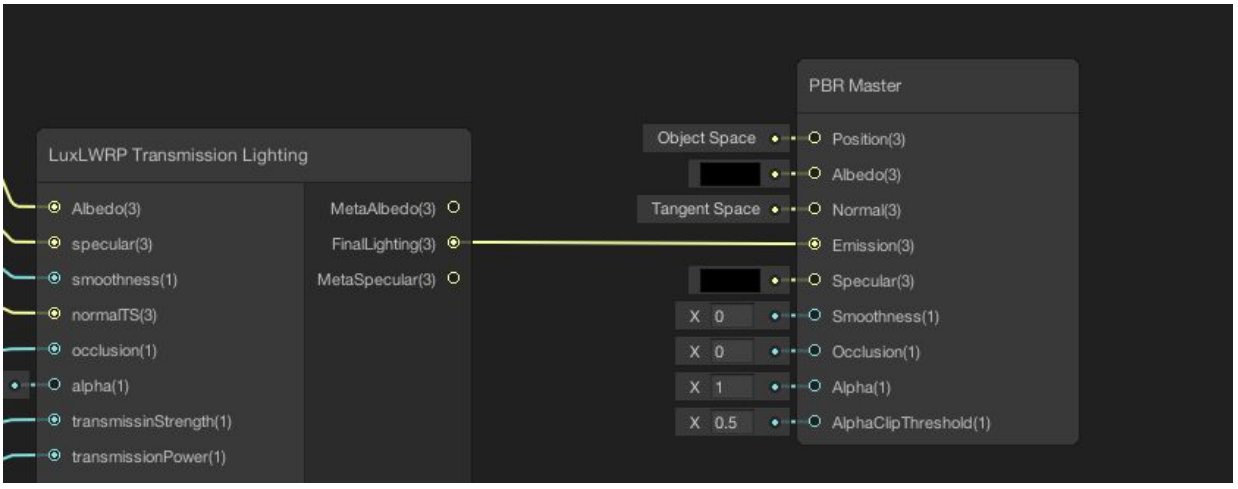
Starting with version 1.11 Lux URP Essentials ships with a collection of custom nodes for Shader Graph. Most nodes are custom lighting functions which hijack the *PBR master node* to get access to all variables and shader keywords.

You can read an in depth explanation of how and why it works on Medium →.

Custom lighting nodes — Introduction

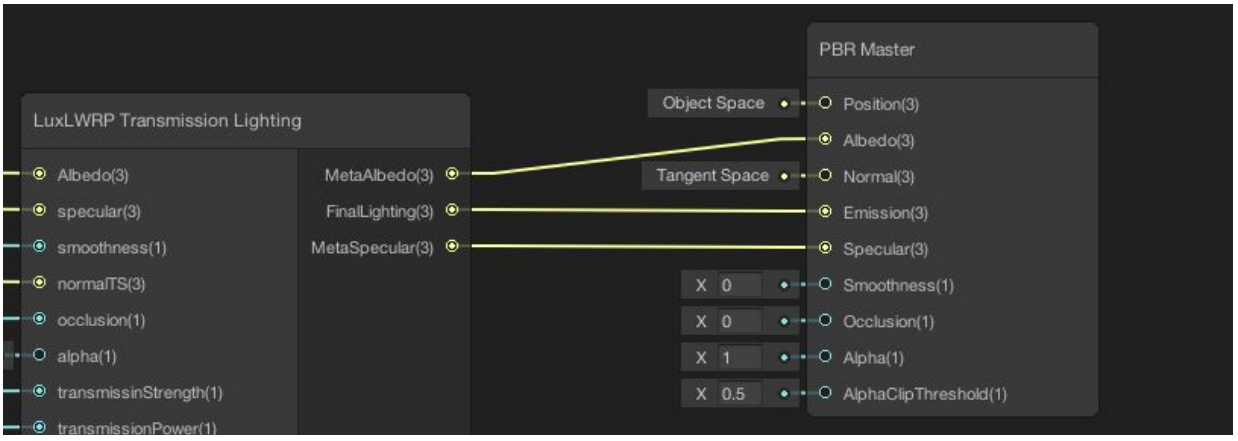
The basic idea is to "mute" the *PBR Master node* by nulling all its inputs so its result will always be half3(0,0,0) and the shader compiler will strip it. The custom lighting then is plugged into the *Emission* node.

Please note: *Workflow* **must** be set to *Specular*.
Using *Metallic* instead would make lighting be calculated twice. You may use the *LuxLWRP Metallic Albedo to Specular Albedo* node to convert from metallic to specular.



Manually nulled PBR Master node Albedo and specular are set to black, smoothness and occlusion to 0.0.

In case you use *GI* just setting *Albedo* and *Specular* to black won't let the material contribute to GI. So the lighting functions provide "Meta" outputs that will set *Albedo* and *Specular* to black in the regular *lighting pass* but provide proper values in the *meta pass*.



Nulled PBR Master node with proper inputs for albedo and specular in the meta pass Albedo and specular are fed by the "Meta" outputs of the lighting function.

Unfortunately we do not have access to per *vertex lighting*. So the lighting function will force Unity to render all additional lights per pixel — even if the pipeline asset defines them per vertex.

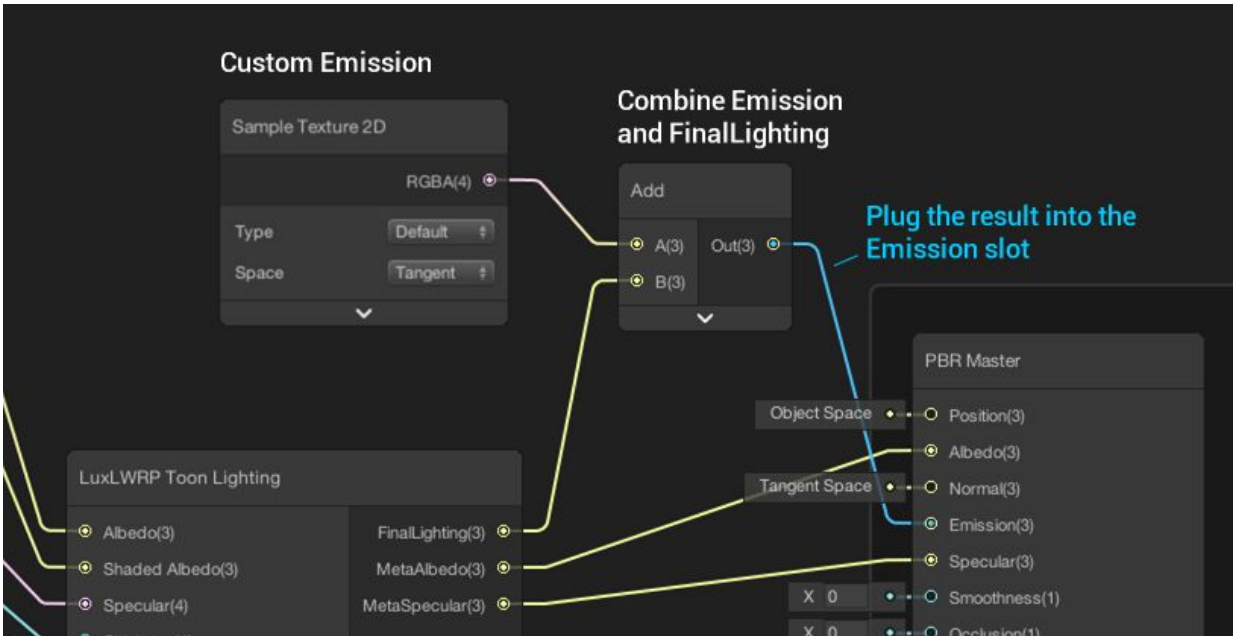
The lighting functions are written in HLSL and wrapped in a Sub Graph. So in order to add them to your Shader Graph just add the desired Sub Graph.

Next to the already mentioned shortcomings we have further restrictions when it comes to Shader Graph at the time of writing: No alpha to coverage, no stencil support. So Shader Graph and custom nodes will let you achieve a lot of things not possible without custom nodes but they do not fully replace the need for manually written HSLS shaders.

Adding Emisission

In case it is not obvious :) — adding a “real” emission feature is straight forward: Just combine your custom emission with the *FinalLighting* most likely using “add” and plug the result to the *Emission* slot of the *PBR Master*. *FinalLighting* will be set to black = 0,0,0 in the meta pass, so the meta pass actually only sees the custom emission.

Unfortunately Shader Graph’s material inspector does not set the *Lightmap Flags* for emission. This one can be fixed by editing the material: Select the material, change the inspector to *Debug* mode, then set the *Lightmap Flags* from 4 to 2.



Feature variants

As Shader Graph does not support custom keywords, optional features such as normal mapping or rim lighting are controlled by booleans exposed by the lighting functions. You have to feed these inputs with constant(!) values (which are known at compile time), so that the shader compiler can wipe out all related “if” constructions and will produce performant code. Do not expose these booleans in the inspector. At least not in the final build :)

So in order to create feature variants for different materials you actually have to duplicate the shader graph, then edit the constants within the graph.

Please have a look at the provided sample graphs.

Toon Lighting

I never ever looked into toon lighting before. So this is my first toon lighting function inspired by Unity's Chan project. It supports duo tone lighting by letting you define lit and unlit albedo values, supports rim lighting as well as eyeballed *BlinnPhong* specular highlights (no PBR).

As simple Toon Outline shader is included in the package as well.

Toon lighting using duo tone shading and rim lighting. Specular lighting is muted as *Specular* is set to black in the example materials. Deactivating Specular lighting in the shader however would be more performant.

Sub Graph Inputs

Albedo Diffuse color

Shaded Albedo Darkened diffuse color which is used on unlit pixels.

Specular Color of the specular highlights.

Shininess Drives the brightness and size of the specular highlights.

NormalTS Normal in tangent space (needs *Enable Normal Mapping* to be checked at compile time).

Occlusion Ambient occlusion.

Diffuse Step Lets you define where the shader switches from unlit to lit.

Diffuse Falloff Lety you define the edge or feather between unlit and lit. *Smaller values will result in a sharper edge.*

Specular Step Lets you define where the shader switches from no highlight to highlight. *Should be around 0.5*

Specular Falloff Lets you define the edge or feather of the highlight. *Smaller values will result in a sharper edge.*

Shadow Falloff Lets you define the edge or feather of the real time shadows. *Smaller values will result in a sharper edge.*

Shadow Bias Directional Defines the shadow strength of the directional light. Values > 0.0 will reduce the shadow strength and let you create a Zelda like lighting.

Shadow Bias Additional Defines the shadow strength of the additional lights. Values > 0.0 will reduce the shadow strength and let you create a Zelda like lighting.

Rim Power Defines the width of the rim lighting

Rim Falloff Lets you define the edge or feather of the rim lighting. *Smaller values will result in a sharper edge.*

Rim Color Rim color.

Rim Attenuation Drives the influence of diffuse lighting (angle attenuation and shadows) on rim lighting.

LightMapUV In case you use lightmaps feed in the proper lightmap UV channel (UV1)

Enable Specular (Feature variant to be defined at compile time) If checked the shader will calculate specular BlinnPhong based highlights.

Enable Normal Mapping (Feature variant to be defined at compile time) If checked the shader will transfer the provided normal in tangent space to world space. Otherwise the vertex normal will be used for shading.

Enable Rim Lighting (Feature variant to be defined at compile time) If checked the shader will calculate rim lighting.

Main light colorizes shadows (Feature variant to be defined at compile time) If checked the directional light will colorize all pixels of the material — even the unlit ones. NdotL and shadows will be ignored.

Add lights colorize shadows (Feature variant to be defined at compile time) If checked additional lights will colorize all pixels of the material — even the unlit ones – based on distance attenuation only. NdotL and shadows will be ignored.

Additional Lights colorize the shaded parts.

Additional Lights do not colorize the shaded parts.

Sub Graph Outputs

FinalLighting The final lighting to be plugged into the Emission slot of the PBR node.

MetaAlbedo Outputs black for the regular lighting pass, contains albedo for the meta pass.

MetaSpecular Outputs black for the regular lighting pass, contains a standard dielectric specular value for the meta pass (the specular color as used by the shader is not physically based).

Transmission Lighting

The transmission lighting function is a slightly simplified version of the lighting used by the HLSL shaders as it does not support wrapped around diffuse lighting.

The package provides two shader examples with different complexity.

Sub Graph Inputs

Albedo Diffuse color

Specular Specular color.

Smoothness Smoothness.

NormalTS Normal in tangent space (needs *Enable Normal Mapping* to be checked at compile time).

Occlusion Ambient occlusion.

Alpha In case you switch your shader graph to transparent you have to feed in an alpha value here.

Transmission Strength Lets you scale transmission. Usually fed with thickness and a multiplier.

Transmission Power Determines view dependency. Higher values will make transmission kick in only when the view ray more or less directly points towards the light source.

Transmission Distortion When calculating transmission the shader distorts the inverted light direction vector slightly by the given normal to simulate the scattering. Default value is 0.01. Higher values will give you more scattering and break up the uniform look.

Transmission Shadow Strength As transmission might be totally eliminated by self shadowing this parameter lets you suppress shadows when it comes to transmission. Other lighting features (diffuse, specular) are not affected.

Mask by incoming shadow strength Lets you suppress transmission according to the shadow strength as set on the given light source - or from point lights which do not cast any shadows.

LightMapUV In case you use lightmaps feed in the proper lightmap UV channel (UV1)

Enable Normal Mapping (Feature variant to be defined at compile time) If checked the shader will transfer the provided normal in tangent space to world space. Otherwise the vertex normal will be used for shading.

Sub Graph Outputs

FinalLighting The final lighting to be plugged into the Emission slot of the PBR node.

MetaAlbedo Outputs black for the regular lighting pass, contains albedo for the meta pass.

MetaSpecular Outputs black for the regular lighting pass, contains the specular value for the meta pass.

Charlie Sheen Lighting

Charlie Sheen lighting is suitable for most cloth materials as it adds some sheen or fuzz lighting at grazing angles — caused by little fibres stinging out of the material.

The provided Shader Graph is set up to support single sided materials as well using the *Double sided flipped normalTS* node handle normals on back faces properly.

Sub Graph Inputs

Albedo Diffuse color

Specular Specular color.

Smoothness Smoothness.

NormalTS Normal in tangent space (needs *Enable Normal Mapping* to be checked at compile time).

Occlusion Ambient occlusion.

Alpha In case you switch your shader graph to transparent you have to feed in an alpha value here.

Sheen Color Color which tints the specular highlights.

Transmission Strength Lets you scale transmission. Usually fed with thickness and a multiplier.

Transmission Power Determines view dependency. Higher values will make transmission kick in only when the view ray more or less directly points towards the light source.

Transmission Distortion When calculating transmission the shader distorts the inverted light direction vector slightly by the given normal to simulate the scattering. Default value is 0.01. Higher values will give you more scattering and break up the uniform look.

Transmission Shadow Strength As transmission might be totally eliminated by self shadowing this parameter lets you suppress shadows when it comes to transmission. Other lighting features (diffuse, specular) are not affected.

LightMapUV In case you use lightmaps feed in the proper lightmap UV channel (UV1)

Enable Normal Mapping (Feature variant to be defined at compile time) If checked the shader will transfer the provided normal in tangent space to world space. Otherwise the vertex normal will be used for shading.

Enable Transmission (Feature variant to be defined at compile time) If checked the shader will add transmission lighting.

Sub Graph Outputs

FinalLighting The final lighting to be plugged into the Emission slot of the PBR node.

MetaAlbedo Outputs black for the regular lighting pass, contains albedo for the meta pass.

MetaSpecular Outputs black for the regular lighting pass, contains the specular value for the meta pass.

GGX Anisotropic Lighting

GGX Anisotropic lighting is suitable for cloth materials, which have a rather closed surface and more or less no fibres stinging out. It can be used for any other anisotropic surface as well. It is a quite expensive lighting model tho.

The provided Shader Graph is set up to support single sided materials as well. It uses the Double sided flipped normalTS node to handle normals on back faces properly.

Sub Graph Inputs

Albedo Diffuse color

Specular Specular color.

Smoothness Smoothness.

NormalTS Normal in tangent space (needs *Enable Normal Mapping* to be checked at compile time).

Occlusion Ambient occlusion.

Alpha In case you switch your shader graph to transparent you have to feed in an alpha value here.

Anisotropy Controls the scale factor for anisotropy. 0 would be standard isotropic lighting, values smaller or greater 0 will shift the specular highlights according to the tangent or bitangent.

Transmission Strength Lets you scale transmission. Usually fed with thickness and a multiplier.

Transmission Power Determines view dependency. Higher values will make transmission kick in only when the view ray more or less directly points towards the light source.

Transmission Distortion When calculating transmission the shader distorts the inverted light direction vector slightly by the given normal to simulate the scattering. Default value is 0.01. Higher values will give you more scattering and break up the uniform look.

Transmission Shadow Strength As transmission might be totally eliminated by self shadowing this parameter lets you suppress shadows when it comes to transmission. Other lighting features (diffuse, specular) are not affected.

LightMapUV In case you use lightmaps feed in the proper lightmap UV channel (UV1)

Enable Normal Mapping (Feature variant to be defined at compile time) If checked the shader will transfer the provided normal in tangent space to world space. Otherwise the vertex normal will be used for shading.

Enable Transmission (Feature variant to be defined at compile time) If checked the shader will add transmission lighting.

Sub Graph Outputs

FinalLighting The final lighting to be plugged into the Emission slot of the PBR node.

MetaAlbedo Outputs black for the regular lighting pass, contains albedo for the meta pass.

MetaSpecular Outputs black for the regular lighting pass, contains the specular value for the meta pass.

Clear Coat Lighting

Clear coat lighting for materials such as car paint.
Please note: The clear coat lighting effectively doubles the cost of specular computations.

The provided Shader Graph uses simple slider inputs for most properties. You may however change it to texture lookups of course.

Sub Graph Inputs

Base Color Diffuse color (albedo) of the base layer.

Secondary Color secondary diffuse color of the base layer which will be applied at grazing angles (needs *Enable Secondary Color* to be checked at compile time).

Base Layer Specular Specular color of the base layer.

Base Layer Smoothness Smoothness of the base layer.

Base Layer NormalTS Normal of the base layer in tangent space (needs *Enable Normal Mapping* to be checked at compile time).

Base Layer Occlusion Ambient occlusion for the base layer.

Alpha Would be needed if you created a transparent material. Default is 1.

Clear Coat Drives the thickness of the coat which influences the darkening of the albedo of the base layer towards grazing angles.

Clear Coat Smoothness Smoothness of the clear coat.

Clear Coat Specular Specular of the clear coat. Default is RGB(51,51,51). *Actually you should not really change it as some calculations are based upon the default value.*

LightMapUV In case you use lightmaps feed in the proper lightmap UV channel (UV1)

Enable Normal Mapping (Feature variant to be defined at compile time) If checked the shader will transfer the provided normal in tangent space to world space. Otherwise the vertex normal will be used for shading.

Enable Secondary Color (Feature variant to be defined at compile time) If checked the shader will mix *Base Color* and *Secondary Color* according to the viewDirection.

Enable Secondary Reflection Sample (Feature variant to be defined at compile time) If checked the shader will sample the given reflection probe twice: once for the coat layer and

once for the base layer. By default reflections get only applied to the coat layer. *Expensive but worth it on certain materials such as coated carbon fibres.*

Sub Graph Outputs

FinalLighting The final lighting to be plugged into the Emission slot of the PBR node.

MetaAlbedo Outputs black for the regular lighting pass, contains albedo for the meta pass.

MetaSpecular Outputs black for the regular lighting pass, contains the specular value for the meta pass.

Skin Lighting

This lighting node offers the pre integrated skin lighting features also used by the Lux HLSL skin shader. It uses up to two different normal samples for diffuse and specular lighting and supports small scale scattering as well as large scale transmission.

Sub Graph Inputs

Base Color Diffuse color (albedo) sample as RGB.

Smoothness The smoothness value as a single half.

Specular Specular color as RGB.

Alpha Currently not used.

Emission Emission color (RGB) which will be added on top of the final lighting.

Normal Map The normal texture.

UV The UVs to sample the normal texture. Usually UV0.

Normal Scale Scale of the sampled normal.

Diffuse Normal Bias Amount of blur added to the sampled diffuse normal (by sampling a lower mip level)

Subsurface Color Color used to tint transmission.

Ambient Reflection Strength Lets you adjust the intensity of the ambient reflections.

Skin Mask Must be 1.0 on parts which shall use skin lighting and 0.0 on parts which shall use standard lighting.

Thickness Values around 1.0 define thin parts where there will be a lot of subsurface scattering and transmission like on the nose or the ears.

Occlusion Ambient occlusion value.

Transmission Power Drives the view dependency of the transmission effect. Larger values will make it more view dependent.

Transmission Strength Lets you scale the transmission effect.

Shadow Strength As transmission might be totally eliminated by self shadowing this parameter lets you suppress shadows when it comes to transmission. Other lighting features (diffuse, specular) are not affected.

Mask by incoming shadow strength Lets you suppress transmission according to the shadow strength as set on the given light source - or from point lights which do not cast any shadows (here shadow strength is 0.0).

Transmission Distortion When calculating transmission the shader distorts the inverted light direction vector slightly by the given normal to simulate the scattering. Default value is 0.01. Higher values will give you more scattering and break up the uniform look.

Curvature Drives the pre-integrated diffuse lighting and lets you control small scale light scattering. *The example shader supports sampling a curvature texture as well as deriving it from the thickness value.*

Light Map UV Usually not need on skin...

Enable Normal Mapping *If checked at compile time* the final shader will look up the provided normal map.

Enable Diffuse Normal Sampling *If checked at compile time* the final shader will perform a 2nd normal map texture lookup to get the blurred diffuse normal. *Needs Enable Normal Mapping to be checked as well.*

Please note: Shader inputs have to provide a skin LUT texture (referenced as _SkinLUT) to give you proper lighting.

Sub Graph Outputs

FinalLighting The final lighting to be plugged into the Emission slot of the PBR node.

MetaAlbedo Outputs black for the regular lighting pass, contains albedo for the meta pass.

MetaSpecular Outputs black for the regular lighting pass, contains the specular value for the meta pass.

Transparent Lighting

Deprecated as URP >= 7.2. now supports proper shadows on transparent materials.

This lighting node offers advanced lighting for transparent surfaces and makes them receive proper directional light shadows.

You may combine a material using this shader with a material using the *Lux simple multiply* Shader Graph to actually tint transparent objects or tweak the albedo/alpha input so that it looks as if it would somehow tint the background. Please have a look at the *Custom Shader Graphs Demo* scene to find out more.

Important: In order to get proper PBR lighting you have to set *PBR Master → Blend* to *Premultiply*. You also have to feed *Alpha* in the *PBR Master* node as otherwise Shader Graph will not set the needed keyword *_ALPHAPREMULTIPLY_ON* will not be set.

Sub Graph Inputs

Albedo Diffuse color

Specular Specular color.

Smoothness Smoothness.

NormalTS Normal in tangent space (needs *Enable Normal Mapping* to be checked at compile time).

Occlusion Ambient occlusion.

Alpha The alpha value here. **Please note:** You also have to feed *Alpha* in the *PBR Master* node as otherwise Shader Graph will not set the needed keyword *_ALPHAPREMULTIPLY_ON*.

LightMapUV In case you use lightmaps feed in the proper lightmap UV channel (UV1)

Enable Normal Mapping (Feature variant to be defined at compile time) If checked the shader will transfer the provided normal in tangent space to world space. Otherwise the vertex normal will be used for shading.

Sub Graph Outputs

FinalLighting The final lighting to be plugged into the Emission slot of the PBR node.

MetaAlbedo Outputs black for the regular lighting pass, contains albedo for the meta pass.

MetaSpecular Outputs black for the regular lighting pass, contains the specular value for the meta pass.

Simple Multiply

This Shader Graph lets you create a material which will just multiply on top of the given screen buffer and therefore tint it.

Important: In order to get proper results you have to set *Unlit Master* node → *Blend* to *Multiply*.

Graph Inputs

Tint Color The tint color.

Power The strength of the tint color is calculated according to the NdotV product to add more tint to areas where we might assume a longer way of the light to travel through the medium..

Tint min Lets you specify the minimum amount of tint at flat viewing angles.

Using Multiply we can not handle fog properly. The shader currently simply fades out according to the fog density which makes it vanish when looking towards the skybox.

Flat Shading

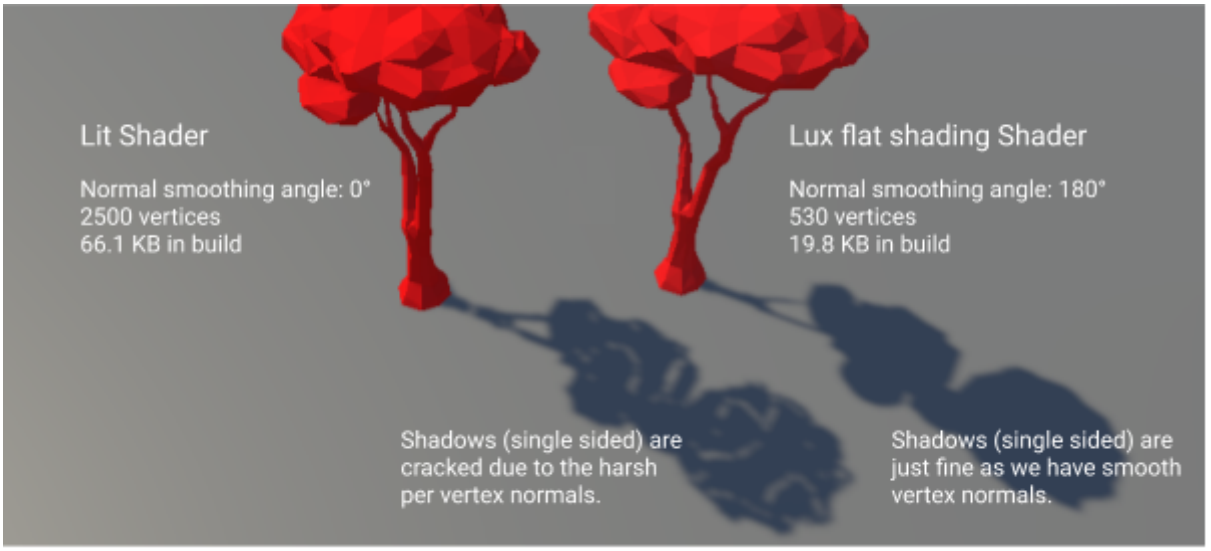
The flat shading lighting node allows you to apply flat shading to any mesh – regardless of its vertex normals: It simply skips these in the fragment shader and calculates the normals as used by the lighting function only based on the gradient of the world space position.

Actually we could even drop the vertex normals entirely if they were not needed in the vertex shader to apply an offset and reduce shadow acne...

So instead of importing the mesh with a normal smoothing angle of 0° you should import it with fully smoothed normals using a normal smoothing angle of 180° (or any other reasonable value).

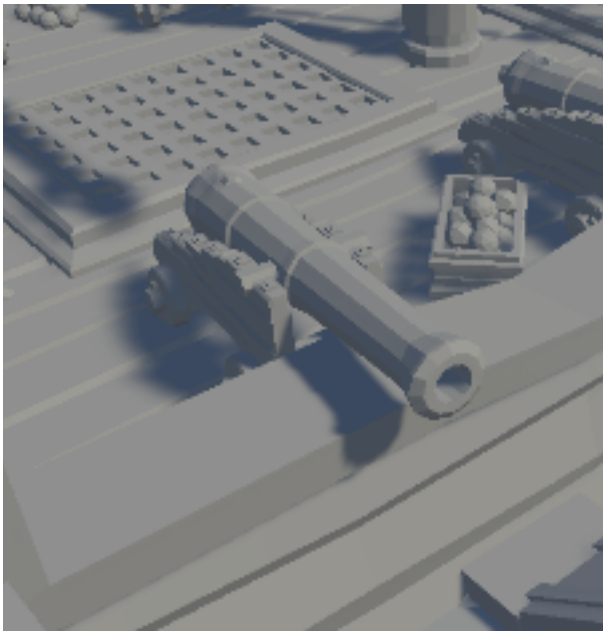
This will fix all issues when it comes to cracks in the shadows (like shown below) and saves a lot of vertices, memory and bandwidth. The fragment shader benefits from less work as it does not have to interpolate the vertex normal which should more or less compensate for the work that has to be done to calculate the derivatives.

> *Cracks in the shadows could mostly be fixed using: Cast Shadows = Two Sided – but this would increase overdraw.*

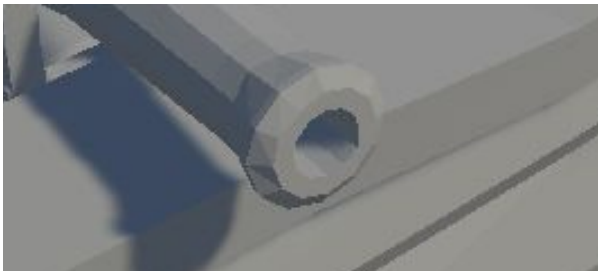


However normals will be calculated based on the actual geometry which in our case always consists of triangles – whereas using quads in your DCC app and setting the normal smoothing angle to 0° in the import settings will make Unity calculate quad based normals.

The following images compare imported hard vertex normals and Lux flat shading:



Standard Lit shader using hard vertex normals.



Lux flat shading shader
It works fine on most surfaces. The opening of the cannon however clearly reveals some triangles.

In order to keep quad based shading you have to make *planar quads* – e.g. using *Blender* and *Mesh → Clean Up → Make Planar Faces*. This will slightly change your geometry and won't work for all faces but should fix most problems.

If this does not work or is no option – well, then you have to go with hard vertex normals.

Sub Graph Inputs

Albedo Diffuse color

Specular Specular color.

Smoothness Smoothness.

Occlusion Ambient occlusion.

Alpha The alpha value here. **Please note:** You also have to feed *Alpha* in the *PBR Master* node as otherwise Shader Graph will not set the needed keyword *_ALPHAPREMULTIPLY_ON* will not be set.

LightMapUV In case you use lightmaps feed in the proper lightmap UV channel (UV1)

Sub Graph Outputs

FinalLighting The final lighting to be plugged into the Emission slot of the PBR node.

MetaAlbedo Outputs black for the regular lighting pass, contains albedo for the meta pass.

MetaSpecular Outputs black for the regular lighting pass, contains the specular value for the meta pass.

Double sided flipped normalTS

Helper node for double sided materials which flipps the tangent space normal according to VFACE.

Sub Graph Inputs

NormalTS Normal in tangent space.

Sub Graph Outputs

NormalTS Flipped or unflipped normal in tangent space based on VFACE.

Metallic Albedo to Specular Albedo

Helper node to convert Metallic/Albedo to Specular/Albedo as needed by the custom lighting nodes.

Sub Graph Inputs

Albedo Albedo input.

Metallic Metallic input.

Sub Graph Outputs

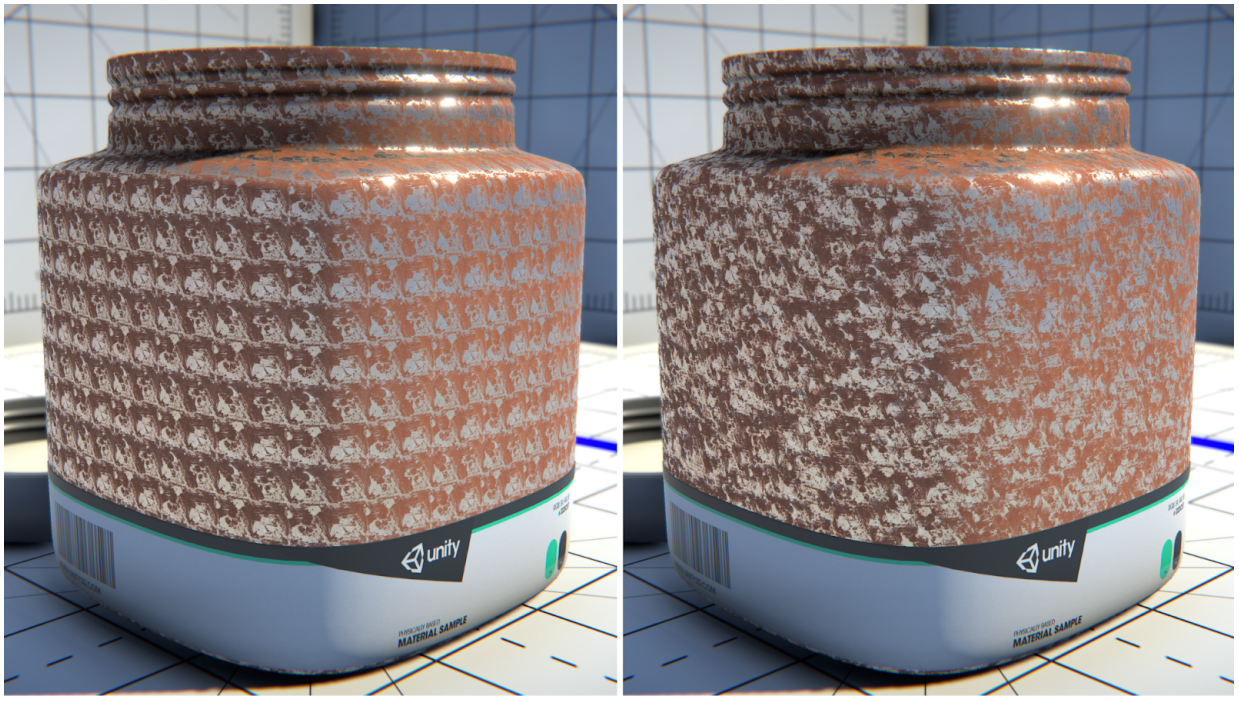
Albedo Calculated albedo output.

Metallic Calculated specular color.

Procedural Stochastic Texturing

Node which implements *procedural stochastic texturing* like introduced by Eric Heitz and Thomas Deliot and removes tiling artifacts. Read more about this on the Unity Blog.

Please have a look at Procedural Stochastic Texturing Simple as well.



Regular texturing on the left, procedural stochastic texturing on the right (Image taken from Unity Blog).

This node is really a beast as it needs a lot of inputs, which even have to be exposed in the editor in order to make Unity serialize the properties. So making a sub graph out of it did not make any sense to me.

It is also a beast when it comes to performance because a regular texture sample will be transformed into up to 7 samples (3 regular samples and 4 into the LUT which actually is rather small).

Usage

1. Create a new material using the *Lux Procedural Texturing* shader graph and assign it to your object.
2. Create a *ProceduralTexture2D* asset using right click/Create or Assets/Create. Assign the desired texture input, select its type (*Color* for strictly color information such as albedo or emission, *Normal* for normal maps, *Other* for other data such as occlusion, roughness, height, etc.) and hit *Apply*. *You have to create one ProceduralTexture2D asset for each albedo, normal and mask map.*
   Do not forget the check "Include Alpha" in case you want to Smoothness or Alpha from the albedo texture.
3. The *ProceduralTexture2D* asset contains two textures next to a bunch of parameters which are not exposed in its inspector. As there are many of these it would be a pain to copy these manually. So simply assign the *ApplyProceduralTextureProperties* script to your object, assign your material and the *ProceduralTexture2D* assets for albedo, normal and mask, then hit *Apply*.
4. Adjust *Blend* if needed in the material inspector. This value should be between 0 and 1. A value of 0 works well for most cases, but higher values can produce less messy results for textures featuring strong lines and shapes. A value too high will however start showing a hexadecimal pattern due to the blending scheme.
5. Set *Tiling* in the material inspector.

Tweaking the Shader Graph

Like written before: It is a monster… But actually you can leave most of the inputs untouched and focus on the outputs of the *custom function* node.

Inputs

The procedural stochastic custom function may convert three textures at once: *Albedo* (RGB) including *Alpha* or *Smoothness* (A), *Normal* and a *Mask Map* (RGBA).

The *Mask Map* can be anything like a Specular/Smoothness or e.g. a combined Metallic/Occlusion/Smoothness texture. Important however is to set the bool *Mask sRGB* properly: In case you use the *Mask Map* as Specular/Smoothness it should be set to *true* as this texture is an sRGB texture. In case the input texture is in linear color space just leave it at *false*. *Set Mask sRGB at compile time so the shader compiler can stripp out any branching.*

Albedo will always be treated as sRGB texture while the *Normal* will be treated as normal.

Outputs

More interesting than the inputs are the outputs.

Albedo(3) returns sRGB color sample.

Alpha(1) returns the sample from the alpha channel. It may contain Smoothness or Alpha – depending on what you added to the alpha channel in your input texture.

Normal(3) returns the unpacked tangent normal.

Mask(4) returns the sample from the Mask Map – either in sRGB or linear color space.

The included shader graph and demo material use the Mask Map to store Ambient Occlusion.

Optimizing

Optimize the shader be simply disconnecting outputs of the custom function from the master node: The shader compiler will automatically stripp all unused code.

Procedural Texturing

This node is a simplified version of the node described above. It skips all *ProceduralTexture2D* asset inputs and works on the original textures instead. Although it is way less accurate and crisp or vivid especially at low *Blend* values, it still produces nice results, is faster to render (3 lookups per texture only) and easier to handle, which allows us to include it as a sub graph :)

The sub graph is able to handle up to 4 textures at once (albedo, normal and two arbitrary textures).

Blend weights as calculated by the original formula are modulated by the luminance of the albedo sample.

Sub Graph Inputs

Blend Lets you adjust the sharpness of the blending between the 3 samples.

UV Base UV coordinates used for all texture samples.

Procedural Scale Scales the hexadecimal pattern used by the blending. *Keep this around 1.0 to avoid artifacts.*

Albedo Texture (RGBA) Albedo input (A may contain Alpha or Smoothness)

Normal Map Normal Map.

Normal Scale Scale applied to the normal.

Metallic Specular (RGBA) Arbitrary – e.g. a metallic or specular texture. Actually just any kind of input.

Mask Map (RGBA) Arbitrary – anything like a combined Thickness/Occlusion/Smoothness texture.

Sub Graph Outputs

Albedo Sampled albedo output.

Alpha Sampled alpha output.

Normal Scaled normal in tangent space.

Metallic Specular Sampled metallic specular output.

Mask Sampled mask output.

Optimizing

Optimize the shader be simply disconnecting outputs of the sub graph from the master node: The shader compiler will automatically stripp all unused code.

Improved Sampling

This node eliminates the common diamond “crosses” caused by bilinearly sampling a low res texture at virtually no costs as only a few ALUs are added - adopting the work of Inigo Quliez.

Inputs

Base Texture Assign your base texture here.

Base UVs Assign the proper UV channel here - most likely UV0.

Outputs

Improved UVs Use these UVs to finally sample your texture(s)

Advanced Parallax

This subgraph allows you to use advanced parallax mapping as used by the Uber shader. It combines 2 samples of the provided heightmap in order to reduce pancaking artifacts you usually get from parallax mapping. This being said it is still rather cheap compared to other methods like *parallax occlusion mapping* which do actual ray marching using 8-32 heightmap samples if not even more.

Inputs

HeightMap The heightmap.

Parallax Describes the amount parallax extrusion.

UV the original UV coordinates.

Outputs

ParallaxUV The final parallax UVs you should use to sample all textures.

Height Final height which may be used in some way…

You will find an example of how to use it in the *Lux Advanced Parallax* shader graph.

Camera Fade

This subgraph allows you to fade out objects according to the distance to the camera.
It will enable *alpha testing* so early depth testing will be disabled.

Please note: You may want to use LODs in order to benefit from early depth testing on lower LODs (using another shader). Alternatively you may swap materials by script if the camera approaches.

Inputs

CameraFadeDistance Distance to camera where the fade should start.

CameraInversFadeRange Lets you control the range over which the fade will take place. Higher values will sharpen the transition.

IN Alpha The original alpha value which gets combined with the alpha from the fade.

Outputs

OUT Alpha The final alpha value which must be plugged into the alpha value of the master node..

You will find an example of how to use it in the *Lux Advanced Parallax* shader graph.

Additional Inputs

Fade Shadows In case you declare a keyword -> boolean -> “FADESHADOWS_ON” in the graph inputs you may control if shadows will fade out or not.

You will find an example of how to use it in the *Lux Advanced Parallax* shader graph.

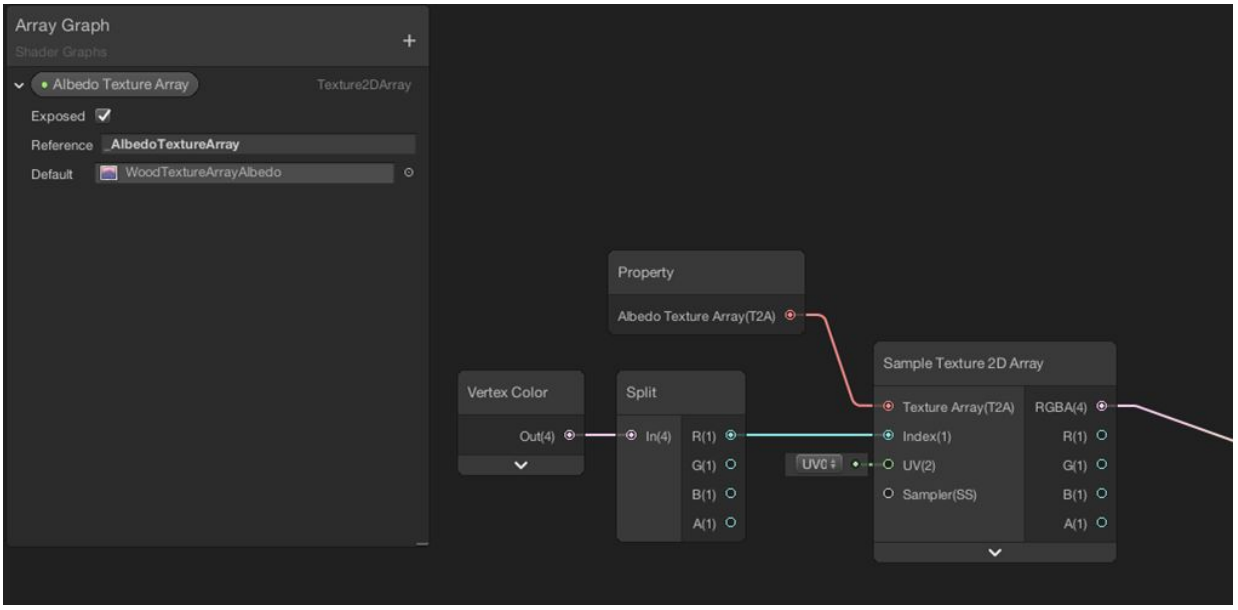
FAQ

Fast or Toon Outlines and multiple Materials

As we can't use multiple passes in LWRP/URP we add the outline effect by adding an additional material to the renderer component. This works fine if your mesh just uses one single material but does not work with multiple materials on the same mesh as Unity will always only draw the last submesh.

So you can either:

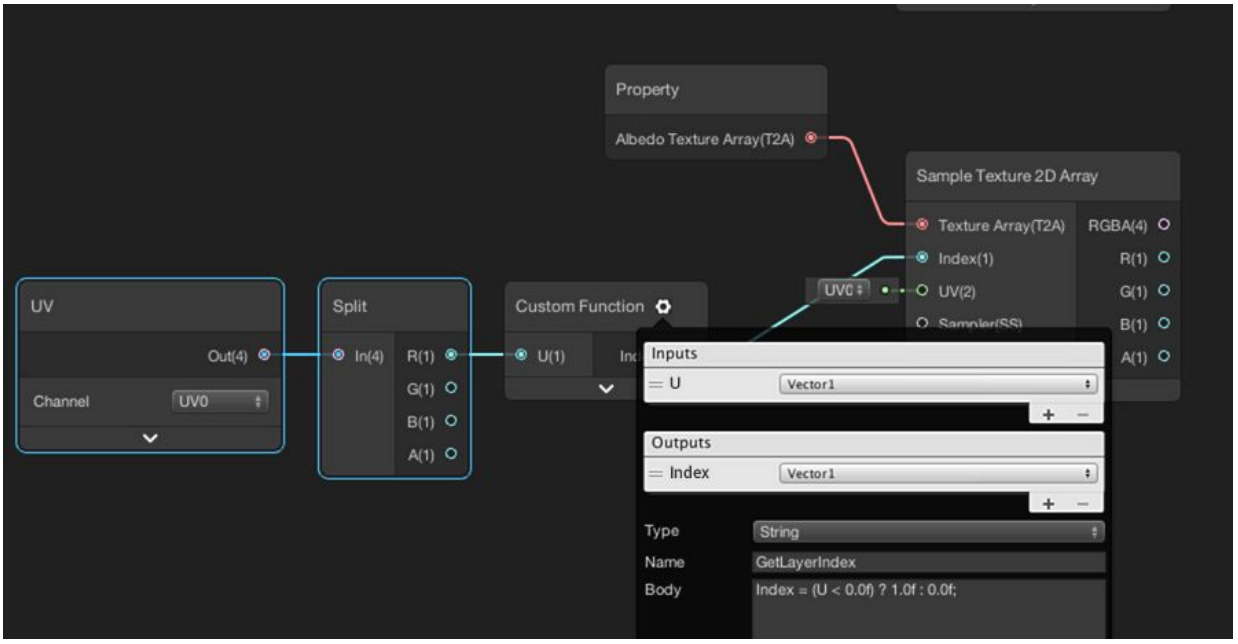
1. Use an additional proxy mesh which only consists of a single material used to only render the outline.
   Might be expensive in case you use skinned meshes.
2. Split up the mesh into 2 meshes = 2 renderers = 2 game objects.
   This far from being ideal but the only solution right now in case the materials use different shaders, different fixed functions or alpha testing and non alpha testing (well, alpha testing is not supported by the outline shaders anyway).
   Please note that some Lux HSLS shaders support advanced and standard lighting like skin and clear coat.
3. Merge the submeshes into one. Then you "simulate" two materials by assigning vertex colors. So material 0 uses e.g. vertex color 0,0,0,0 while material 1 is set to vertex color = 1,0,0,0. In the pixel shader you do not sample a texture2D but a texture array with 2 layers. Which layer will be sampled is then determined by the vertex color. In shader graph this might look like this:



If you do not want to add vertex colors to the mesh (as they will make the mesh data a bit fatter) you could also use different *uv shell positions* to determine material 0 and material 1: Just put the uv shell for material 1 into the negative uv (u) space and keep the shell of material 0 in the positive space.

Then get the index by using: `float index = (uv.x < 0) ? 1.0f : 0.0f;`
Please note that this needs the uvs of mat 1 always to be smaller than 0.0.

In Shader Graph this might look like this:



You will have to do this for all the used textures like albedo, shaded albedo and normal. But of course you have to get the *Index* only once.

In order to **create texture arrays** you may use this free tool from the asset store:

https://assetstore.unity.com/packages/tools/utilities/texture-array-inspector-109547

Creating a tinting glass material

No matter which blending mode you use — you won't be able to create a tinting glass material: Either tinting will look fine or your specular highlights will do. But there is no blending mode which will cover both: Tinting needs multiply, transparent glass however needs premultiply...

The "in shader" solution would be to grab the background, tint it in shader code and render the reflections on top. Then write the result back to the frame buffer. This however would need a "grab pass", read the "grab texture" in the shader, combine both and write the result back to the frame buffer. This is how the included glass HLSL shader does it.

So the question is: Can we get away cheaper?

The answer is yes: If we can afford to render the mesh twice using two different materials.

The first material just tints the background using blendmode = multiply. The second material will add all the lighting using premultiplied alpha blending.

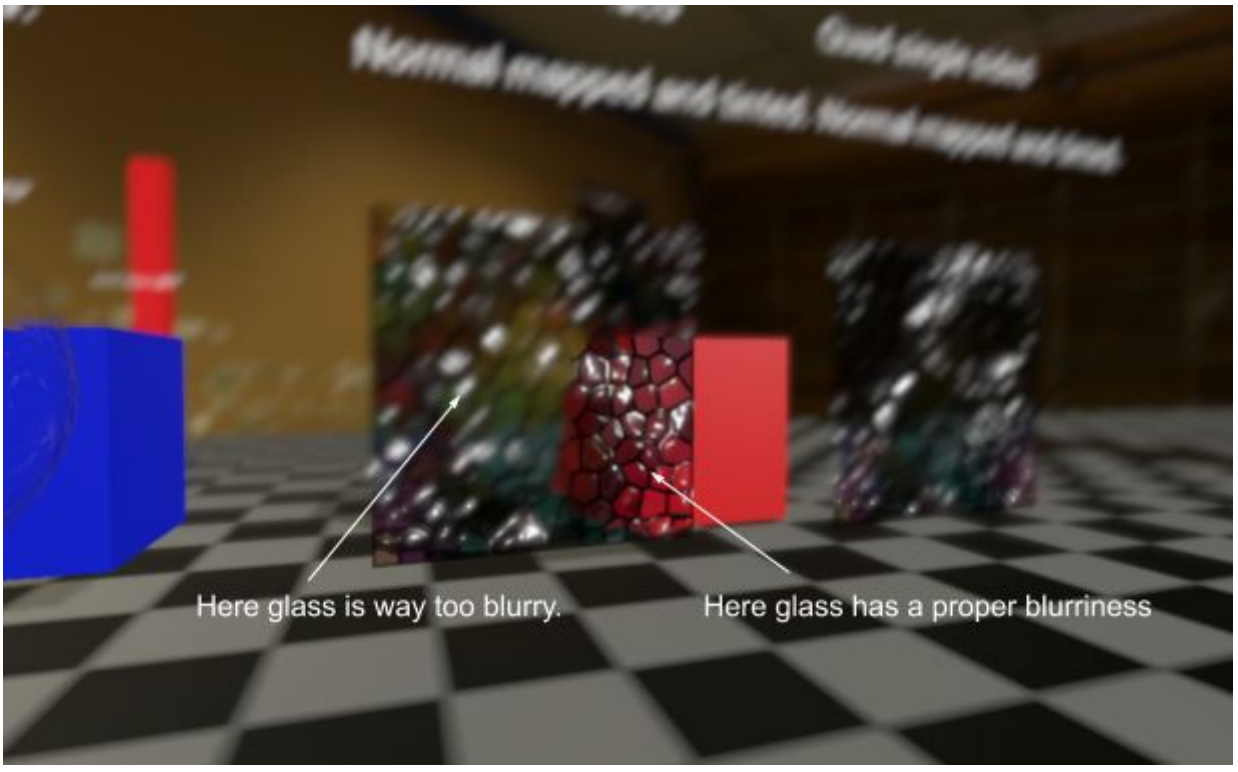
Please have a look at the Custom Shader Graphs Demo → _Transparent Lighting.

Refractive Glass and DOF

Like any other material using the transparent render queue, refractive glass may write into the depth buffer but will not be included in the *_CameraDepthTexture* which is used by the DOF image effect as this snapshot of the depth buffer is *always* taken right after the opaque passes.

Making the glass material writing to depth or not does not make any difference as far as DOF is concerned. It makes sense for glass and all other transparent passes – but it renders simply too late to be included in the snapshot.

This will lead to glass being less blurry where it is close to opaque objects in the background (red cube) and glass being way more blurry where there is nothing in the background at all as shown below:



Refractive glass and DOF Notice the different blur on the glass where the geometry behind it is rather far away (pretty blurry glass) and where is rather close (red cube) – which results in less blurriness.

The solution

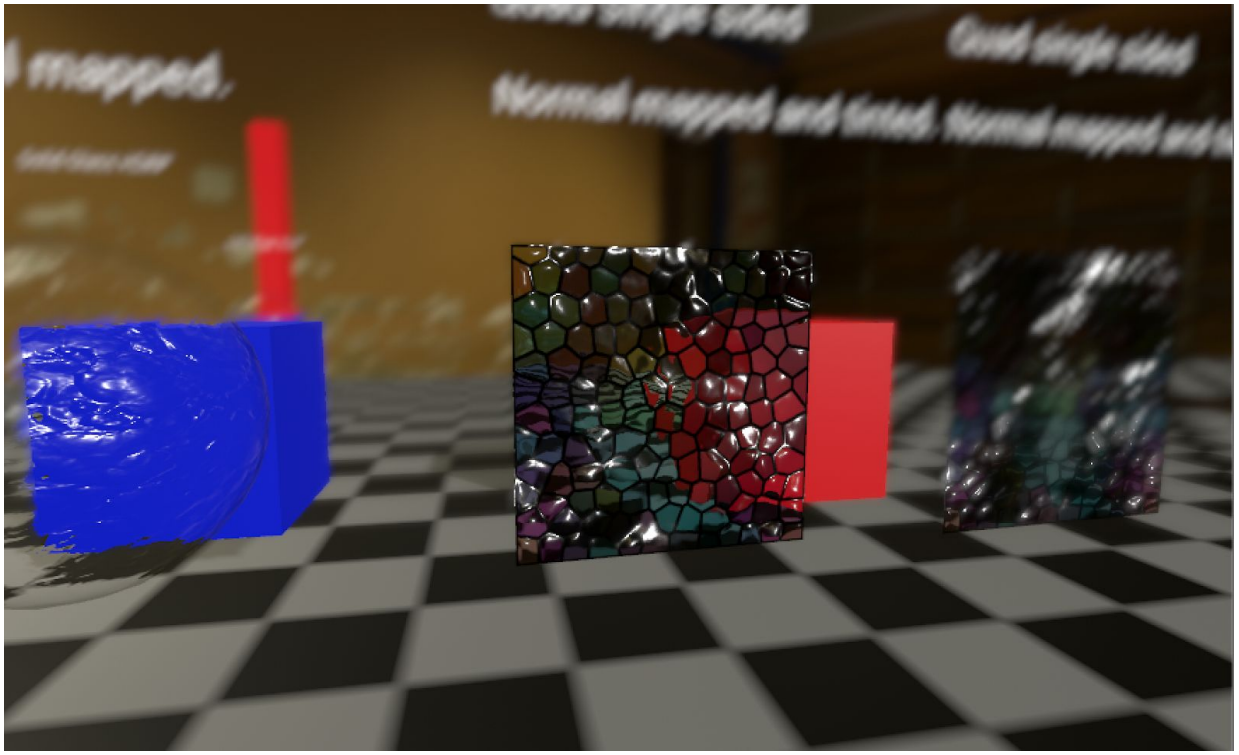
In order to make the entire glass show up proper blurriness according to its distance to the camera or depth we have to make sure that it writes to depth during the opaque pass and before Unity grabs the *_CameraDepth Texture*. But as we want glass to refract the background we need the *_CameraOpaqueTexture* which means that glass has to be rendered on the transparent queue.

Luckily Lux LWRP comes with the *"Depth Only"* shader.

So simply create a new material using the *Depth Only* shader and apply it as 2nd material on the glass geometry.

This will make sure that the glass geometry actually writes to the depth buffer already during the opaque passes and will be seen by the DOF effect.

The transparent glass material, which renders later on the transparent queue, uses *ZTest LEqual* (less or equal) – and as it should have equal depth it will render even if there is something in the depth buffer already at exact the same position.



Refractive glass and DOF – using 2 materials and the *Depth Only* shader Now the entire glass surface shows up a proper blurriness.

You may play with the render queue settings of the *Depth Only* material: Highest allowed value should be 2999 which is Transparent-1. Another breakpoint would be 2450 where cutout materials usually render.

Unfortunately, this technique does not work with transparent materials which rely on the *_CameraDepthTexture* in order to do some distance based calculations like *Water* and *Soft Particles*: Making these write to depth using a *Depth Only* material will corrupt rendering.

Mixing Cloth and standard metallic lighting

Other shaders support mixing different lighting features (like skin or clear coat) – cloth does not as it based on totally different lighting functions compared to standard. So you would have to apply cloth and standard lighting to each pixel and then finally lerp based on a mask…

This is true in the case of *Charlie Sheen* lighting.

GGX anisotropic lighting however lets you more or less mimic standard lighting by setting *anisotropy* to 0.0 on certain parts.

So you would need to add an rgba texture, where (rgb) defines specular color and (a) works as a mask to lerp between the anisotropy from the slider input and 0.0. actually you can mask the metal part with alpha = 0 and simply use:

anisotropy = inputFromSlider * specularTextureSample.a

if you take the shader graph version this should be straight forward as specular color and anisotropy are exposed in the lighting node.

if you want to add it to the hlsl shader, well then you have to dig through its code…

Please note that custom lighting functions always use the specular setup. So albedo should be black and specular should contain the spec color of your metal.