Lux URP Essentials – URP 12.1.+

Since URP 12 introduced a huge amount of new features I had to update Lux URP Essentials in several steps. Since version 1.85 however you now get a fully ported version of the package ready to get you started.

This document gives you a brief overview of what has changed and what has been added for URP 12.1. and above.

Latest Changes

Please read the **ChangeLog.txt** file to find out more. Or have a look into What's new.

Custom Lighting

Custom lighting functions such as Cloth, Skin Transmission or Toon can be used in forward and deferred rendering – however when it comes to deferred, materials using these functions will be rendered in forward rendering.

Due to decals and SSAO, they need an additional but rather cheap GBuffer pass as well :(

> *This however should only be true in case you have checked "Accurate G-buffer normals" in your deferred renderer. Otherwise you may comment out the GBuffer pass.*

Deferred Rendering

Just like in the built in render pipeline URP's deferred renderer does not allow us to use custom stencils – they are simply ignored. This means that stencil based fast outline features are not supported in deferred. *Instead you can use the new Fast Outline Double Pass shader.*

Deferred also does not offer any support for custom lighting functions which therefore will fallback to forward. See above.

Forward Rendering

Forward rendering now supports *Depth Priming* which is just what most people know as *depth prepass*. In case it is enabled or in case decals or SSAO are enabled (which also cause a depth prepass) *Alpha To Coverage* most likely will produce artifacts: We simply can not write more than a single value into the z-buffer.

> *So simply deactivate Alpha To Coverage on all materials in case you use Depth Priming, SSAO or decals.*

Shaders

Additional Surface Options

Most shaders contain two new custom surface options:

Enable Normal in Depth Normal Pass The built in Lit shader will output per pixel normals in the lit depth normal pass which makes it sample the normal map twice. In case you only have subtle normals or just need better performance you may uncheck this and the depth normal pass will skip the per pixel normals. *This affects SSAO and decals.*

Receive Decals HDRP offers this per material and so do Lux URP Essentials. So in case you do not want or need a material to receive decals just uncheck this option.

Uber

Uber's **parallax mapping** has been optimized for the new *Depth Priming* and will produce stable results even if alpha testing is enabled (unlike the built in lit shader. I filed a bug report about this…).

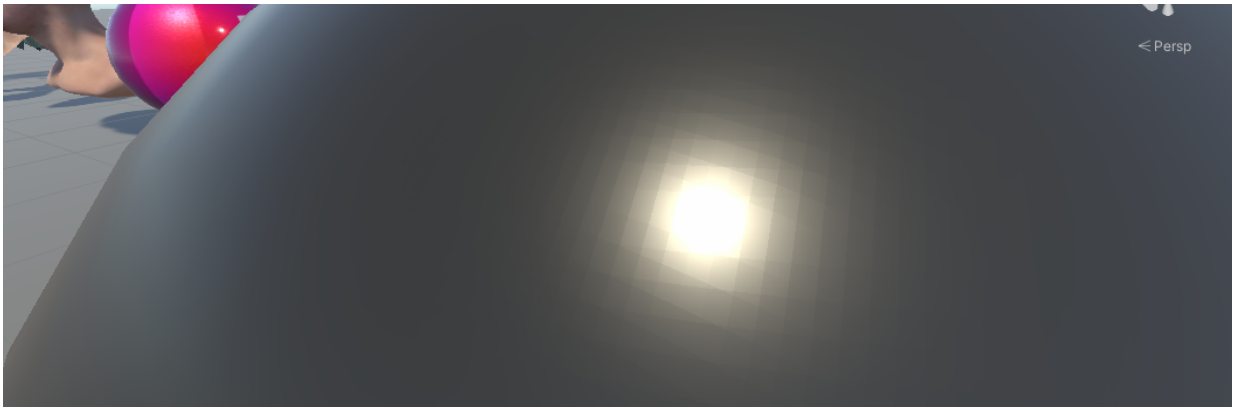
Uber now supports **best-fit normals** as invented by Anton Kaplanyan for Cryengine 3. These are useful in deferred rendering: The default quality of the GBuffer normals is far from being optimal (8bit per channel only) and will create a faceted look on sharp specular highlights.

So URP's deferred renderer offers *Accurate G-buffer Normals*. Activating this feature however means that all normals on all materials will be first encoded to OctQuadEncoded ones during the GBuffer pass and then decoded for each deferred lighting pass – which produces some overhead (and breaks the terrain add pass…).

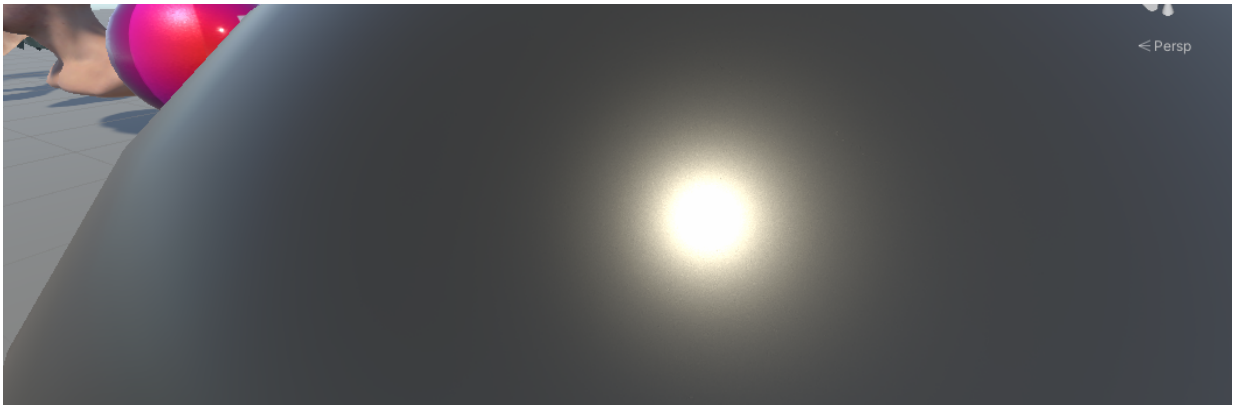
You might have a lot of materials where low quality normals would just be fine as you have a strong normal map or low smoothness so artifacts are barely visible.

Here *best-fit normals* come into play as you dedicatedly specify which material needs high quality normals and only use them on these. *Best-fit normals* only need the encoding step, decoding is just the same as for regular written normals and virtually free.

So *best-fit normals* let you save a lot of computational power and bandwidth by adding the possibility to render high res normals only on selected objects/materials.



URP low quality GBuffer normals. Note the pixelated or faceted look of the highlight.



Uber using best-fit normals. Normals here still are not perfect and will show up some noise. But in most cases they look way better than the standard low quality normals. Using *Accurate Gbuffer normals* all across the board will look better than *best-fit normals* for sure but at higher costs on the GPU.

In case you opt for *Accurate G-buffer Normals best-fit normals* will automatically be optimized away.

Best-fit normals always only are applied in deferred rendering because in forward rendering we simply have nearly perfect normals anyway.

Toon Shader

Due to the new *Depth Priming* we can no longer use a single shader to draw the toon shaded surfaces plus the outline: As we only have one *Depth* or *DepthNormal* prepass the outline would simply be ignored. Thus we have to assign two materials to the renderer: First one for the toon shading and the second one to add the outline.

The "Lux URP/Toon & Outline HLSL" is still included in case you do not use Depth Priming.

In case you use **alpha testing** the "Lux URP/Fast Outline AlphaTested" shader now offers the option "Do not calculate width in Screen Space" which if checked will calculate the outline just like in the "Lux URP/Toon & Outline HLSL" shader. *Look for the fern "SM Fern DepthPriming" in the toon demo.*

You will find an opaque sample looking for "UnityChan Toon DepthPriming".

Grass

URP 12 allows us to use shaders supporting instancing within the terrain engine. So now you can place grass using the grass shader right within the terrain engine.

Grass supports *improved specular lighting* as I added the *Specular Mask* feature from ATG. Using this feature you should be able to add some nice reflections even if your grass model has heavily tweaked/smoothed normals.

It also comes with *Screen Space Normals* which are normals on single sided geometry not being tweaked based on *VFACE* but their orientation in screen space. So if enabled the shader will not flip or mirror the normals but just ensure that normals will always point towards the camera. This may soften lighting on some models using the grass shader (models whose normals are not fully softened).

Grass now supports displacement or **touch bending** by default. So there is no need to use the "Lux URP Grass TextureDisplace" shader anymore which will be deprecated. *You have to enable this feature in the shader tho :)*

Foliage

The foliage shader (which can also be used within the terrain engine, just like grass) offers various techniques to add some kind of transmission even to *deferred rendering*, which are: Standard, Simple, NormalVS, Transmission.

> **Standard** Well, this mode does not support any kind of transmission… Normals of back faces will be corrected using *VFACE* *Cheap*
>
> **Simple** Just like *SpeedTree* the shader will not flip the normals based on *VFACE*. So some faces will show up proper front face lighting, some will show up false back face lighting. This will create the *illusion* of transmission but also may add a lot of false specular highlights. *Cheapest.*
>
> **NormalVS** This will not add transmission lighting but may correct and smooth lighting compared to *Standard* and *Simple*. Normals here are not flipped or mirrored based on *VFACE* but will be tweaked in screen space to always point towards the camera. *A bit more expensive.*
>
> **Transmission** This mode will add proper transmission lighting for the most dominant directional light only. Additional spot or point lights are not supported. This mode will sampe shadows and cookies in the deferred GBuffer pass (which usually does not do this at all) and thus makes it: *Most expensive.*

Wrapped diffuse lighting is not supported in deferred.

Foliage now comes with an early preview of **touch bending** and includes additional, advanced **turbulence** which may replace edge fluttering in the future. The foliage shader also includes a **vertex color debug** more so no additional shader is required.

Terrain

Here Unity seems to have to add some more love:

- The Lit terrain shader shipping with URP 12 does not support bakes shadow maps. Lux Essentials' one does.
- When it comes to *Real Time Global Illumination* the original shader does not support this as well. Lux Essentials' one does somehow.

  Please note: It is all a bit unstable. So in case the additional pass does not render properly leaving black spots on the terrain, the distance terrain using the basemap shader does not show up or anything else strange happens, try to toggle the terrain on and off.

Skin

Skin now supports **detail normals** which however only gets applied to the normals used for specular lighting. Diffuse lighting is not affected by these.

Hair

The transparent **Hair Blend** shader does not define _SURFACE_TYPE_TRANSPARENT although it is transparent in order to receive decals and screen space shadows. In case you run into issues consider defining the keyword by editing the shader (you will find commented lines).

Water

Water is always rendered using forward.

New inputs:

> **Diffuse Normal Up** Lets you tweak the normal that is used to calculate diffuse lighting on the underwater fog. 0.0: Only the geometry normal will be taken into account. 1.0: Only a fully upwards pointing normal will be taken into account. Anything in between will lerp both normals.
>
> **Add Foam from Normal** Lets you strengthen the foam distribution based on the water normals to break up foam. Default was 4.0.

Billboard

The billboard shader will use deferred lighting if deferred is enabled and *Alpha* is set to *Tested*. If *Alpha* is set to *Blended* the shader will always be rendered using the forward pass.

The **LuxURP_BillboardBounds.cs** script lets you tweak the billboards' mesh bounds to prevent them from being culled too early.

Fast Outline Double Pass

In order to make fast outline and stencil work with deferred we have to change the way it works.

Deferred clears the stencil buffer. So we can't prepare the stencil buffer while rendering the mesh using its deferred material shader.

Instead the Fast Outline Deferred shader prepares the stencil buffer itself by using 2 passes: The first one only writes into the stencil buffer while the second one then draws the outline.

> *As this shader does not rely on the original material being able to prepare the stencil buffer you can use it on materials using shader graph shaders as well!*

As the shader uses 2 passes you have 2 settings for ZTesting and Culling.

The stencil settings are shared among the passes, altho the stencil pass of course ignores *Stencil Comparison* and will always write to the stencil buffer if its *ZTest* and *Culling* params let it.

> *Find more information about the fast outline features and stencils in the original documentation.*

Fast Outline AlphaTested Double Pass

Like the *Fast Outline Double Pass* shader described above this shader allows us to draw outlines on alpha tested materials when using deferred shading.

Particles

Simple lit particles are always forward but now support light layers, point light shadows and cookies.

Glass Shader

The Glass shader is always forward but now supports light layers, point light shadows and cookies.

Depth Only

For URP 12 i added a new shader: *Lux URP Depth Only Lit*.

The old ones use regular Depth and DepthNormals passes which may not play nicely with *Depth Priming* or the *Depth Normal* pass.

The *Lux URP Depth Only Lit* just uses a regular Lit pass but only writes to depth. Doing so we can manually set when it gets rendered into the depth buffer by adjusting the *Render Queue* param. Using *Render Queue = Transparent-x* (values between 2750 - 3000 should be fine) here lets you at least occlude transparent materials like water.

Deprecated Shaders

Terrain Blend

The Terrain Blend shader does not play well with URP's new rendering features – first and foremost: Depth Priming.

Depth Priming overrides `ZTest` to `Equal` while the shader would need `LEqual`. We could work around this by setting the shader to `RendeQueue = Transparent` but this may introduce clipping artifacts on objects close to the blended geometry…

Nevertheless version 14.0.4. added some changes which improves its overall compatibility but it still breaks if depth priming is active.

Versatile Blend Shader

The Versatile Blend Shader reveals yet another problem: SSAO which breaks the smooth blending.

Both shaders should still compile using URP 12.1. but have not really been ported.

Shader Graph and custom lighting

The old Shader Graph nodes for custom lighting still work when it comes to forward lighting but do not support any new features such as cookies or light layers.

The deferred problem

They also can not really be used with deferred lighting enabled: Actually they may produce a somehow proper output for just a single directional light – which they may sample and output to emissive. Any additional light however will be fully ignored: Their data is simply not bound.

Here Unity will have to add some functionality like declaring a pass as “Forward only”. I filed a request and we will have to see what they will come up with.

The deferred solution

Meanwhile I looked into a workaround and actually found that **enabling “Clear Coat” in the master node** declares the final forward pass as “UniversalForwardOnly”. So we can use this trick to make the always forward rendered shader graph shaders picking up proper lighting.

Implementing custom lighting

The basics of implementing a custom lighting with Shader Graph are still the same:

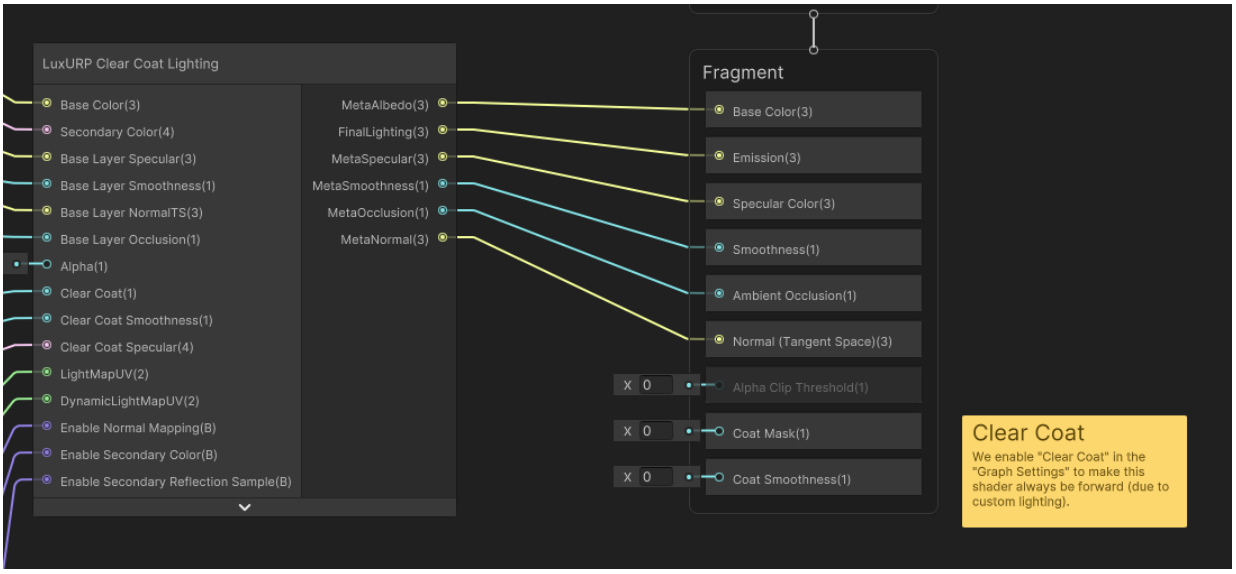
We have to **“mute” the *PBR Master node* by nulling all its inputs** so its result will always be half3(0,0,0) and the shader compiler will strip it. The custom lighting then is plugged into the *Emission* node. ***Workflow* must be set to *Specular*.** Using *Metallic* instead would make lighting be calculated twice. You may use the *LuxURP Metallic Albedo to Specular Albedo* node to convert from metallic to specular.

Please note: Shaders using custom lighting will always be rendered in forward even if your camera uses deferred.

Nulling the PBR Master Node

In order to add basic support for the **Rendering Debugger** we should not simply null Albedo, Specular, Smoothness and Ambient Occlusion but provide proper values for the debug view. We also have to provide a normal in tangent space. Latter is used by the debug view but also by the depth normal pass.

The new custom lighting functions therefore provide new outputs: MetaSmoothness, MetaOcclusion and MetaNormal:



Manually nulled PBR Master node using the Meta Outputs.

Supported features and shortcomings

Rendering Debugger

Custom Shader graphs support **Material Overrides** and let you debug Albedo, Specular, Smoothness, Ambient Occlusion and Normals. They do NOT support **Lighting Debug Modes** as here the shaders fall back using the built in lighting functions.

Real Time Global Illumination Preview

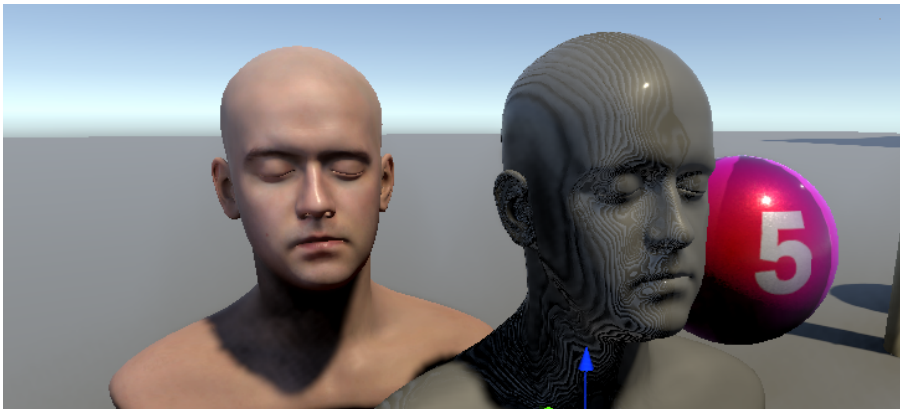
Indirect works. Albedo is broken.

Baked Global Illumination Preview

Here Albedo and Emission are broken – unfortunately. You can however preview the baked lightmap.

Deferred and Accurate G-buffer normals

If this feature is enabled SSAO will break as is not based upon the depth normal buffer but only takes the octa encoded GBuffer normals into account. The result on a simple shader graph based shader using clear coat (gray head):



I do not recommend to use *Accurate G-buffer normals* as they slow down rendering and also break e.g. terrain rendering. If you need accurate highlights on some of your shiny materials:

- use the Lux URP Essentials Uber shader and activate best-fit normals.
- or use a forward only material – which would be most precise.

URP 12+ custom lighting functions

Currently available custom lighting functions:

- GGX Aniso
- Charlie Sheen
- Transmission
- Toon V2
- Skin
- Clearcoat
- Flat Shading: Here i added a simple node (LuxURP Flat Shading Normal) that converts the normal properly. I recommend using this node and going with standard lighting instead of using the custom flat lighting function. See: Lux Flat Shading Deferred shader graph