Lux URP Essentials – Toon Shading

As toon shading has become more important in the package it now has its own documentation which covers the new improvements added to the toon shading and explains all its features in detail.

Unity 6 and Outlines

Since Unity 6 and the new *GPU Resident Drawer* break the old style outline rendering I had to rework toon outlines: Instead of using a custom “two pass” shader or adding two materials to a mesh which only has one submesh, outlines now are rendered using the URP Renderer Features.

Using Renderer Features to draw the outlines fixes issues caused by enabling *depth priming* automatically.

More details →

What’s new

Lux URP Essentials new toon shaders V2 come with improved lighting features such as the support for multiple steps or ramps, energy conserving or simple specular lighting and the control over light falloff and light color contribution.

The **shader graph based version** now exposes most controls like normal mapping or colorized shaded lights directly in the material inspector which makes it way easier to tweak materials.

Please note: The disadvantage of this flexibility however is the vast amount of shader variants Unity compiles at build time! So do something useful like having lunch or going to bed when first running out a build...

The **HLSL based versions** offer even more adjustments right from the inspector – like toggling *Receive Shadows* or *Environment Reflections* and give you full access to the stencil buffer. Enabling *Alpha Clipping* here is just a click.

The *Toon & Outline HLSL* shader may even draw outlines. This still results in two draws but you do not have to assign a second material here in order to get a toon outline. Using Unity 6 you have to add the “LuxURP_OutlinesRendererFeature” to make outlines show up.

Latest changes

Version 1.3: Reworked outlines for Unity 6

Version 1.2: Shaded Decal Color added (URP 12.1+)

Version 1.1: *Anisotropic specular highlights* added.

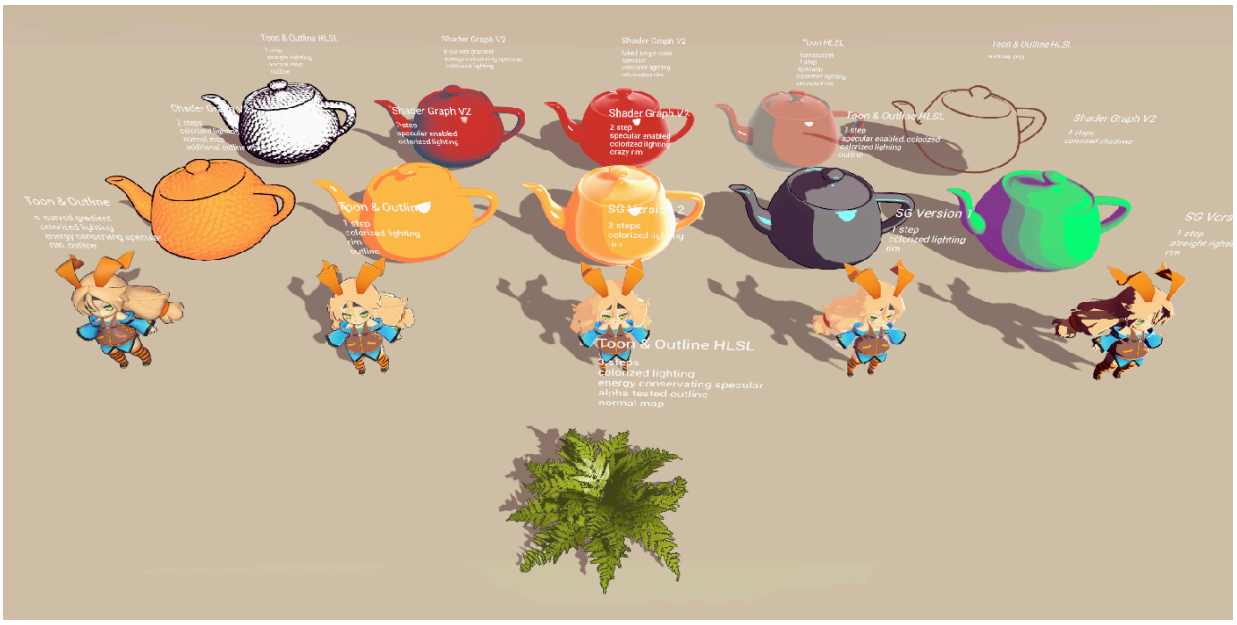


Table of Content

Shader features and inputs

This chapter describes the inputs of the HLSL shaders. Inputs for Shader Graph are likely the same.

Surface Options

Surface Type Lets you choose between *Opaque* and *Transparent*. *Not available in the Toon & Outline shader.*

> **Blending** In case the shader is set to *Transparent* you may specify the blending mode.

ZTest Lets you tweak depth based face culling.

Culling Lets you specify if the shade shall cull back faces, front faces or none. *Default is Back.*

Alpha Clipping If checked the shader will do alpha clipping or alpha testing which is the desired behavior in case your texture contains any alpha. In case you use the *Toon & Outline shader* please make sure, you have a look into the chapter Alpha Testing →

> **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.
>
> **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)*.

Color Mask Only available using URP < 12.1 Choose *All* if you want to regularly output the shader's result. Choose *Depth* if you just want to occlude the outlines which get drawn afterwards. **Please note:** This is limited to the *Toon & Outline shader* only and quite experimental. If set to *Depth* the shader will be rendered on the transparent queue + its *Queue Offset*, which should be maxed out here (+50). Latter is needed to draw other transparents and the skybox properly. Doing so currently allows us to render outlines only, nice!

Receive SSAO Only available using URP 12.1+ As SSAO will kind of corrupt the toon lighting you may just disable it..

Receive Decals Only available using URP 12.1+ 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. **Please note:** The toon shader will only sample albedo from the decals. Decals will be lit using toon lighting. Thus the shader offers the *Shaded Decal Color (multiplier)*.

Enable Normal in Depth Normal Pass Only available using URP 12.1+ 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.*

Decals (URP 12.1 +)

Shaded Decal Color (multiplier) Color that gets multiplied on top of the sampled decals' albedo in shaded areas.

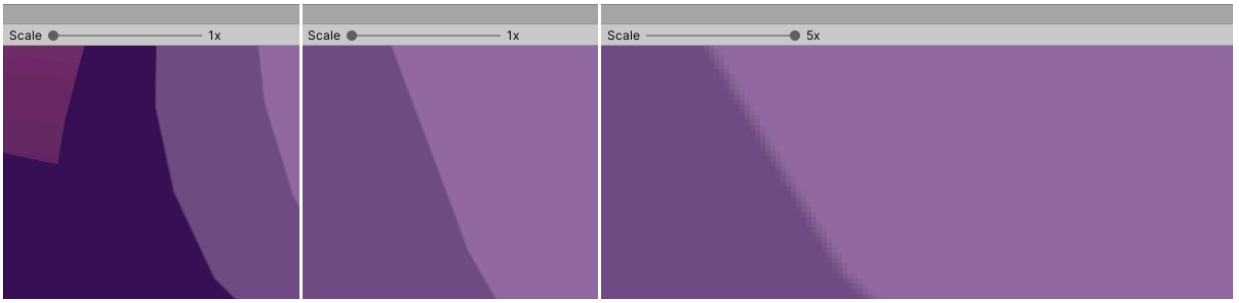
Toon Lighting

Steps

You can add between 1 and 8 steps – which will quantize the N dot L lighting accordingly giving you the typical cell shading look.
In case you're looking for a typical cel shaded style this is the way I recommend as Steps only require some rather cheap math and support great anti aliasing.

From left to right: 5 to 2 steps.

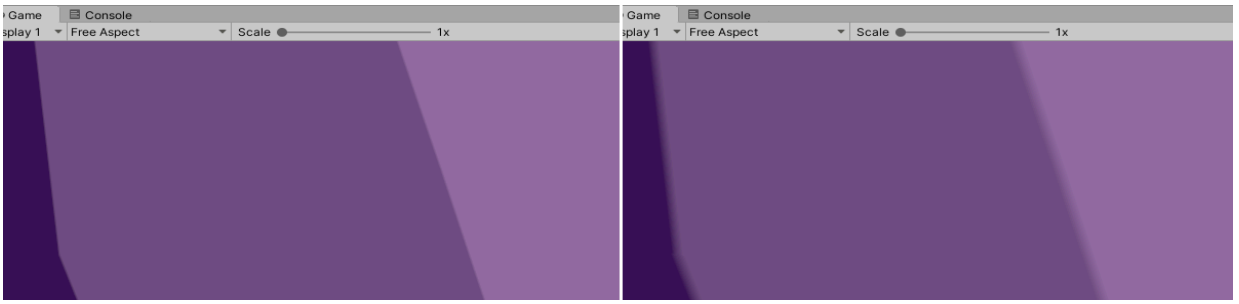
Steps are calculated analytically and thus can be smoothed within the shader in screen space using *fwidth*. This guarantees that steps will always show the same sharpness – no matter how close you get:



Analytical anti aliasing on steps

Diffuse Falloff

The area over which anti aliasing will be applied can slightly be increased using the *Diffuse Falloff* param:



Left: Diffuse Falloff = 0.. **Right:** Diffuse Falloff = 1

Diffuse Step

Diffuse Step lets you pull the highlights towards the light's position or drag it away from it in order to reveal the object's shape in an artist's controlled manner.

The teapots below are all lit by the same directional light, but *Diffuse Step* is used to position or shift the steps on the geometry:

From left to right: Using *diffuse step* = -0.5 to step = 0.25.

This feature is very useful in case real time shadows leak into your cel shading.

Ramp Mode

As stepped lighting may not always fit your needs the shaders also support ramp textures. These can be anything: from a crude 3px wide stepped ramp to a 256px wide gradient – giving you full control over the final lighting.

From left to right: S-curved gradient ramp / S-curved gradient ramp, rim / 2 steps, rim, specular lighting / 2 steps, specular

Ramps can be sampled using *smooth filtering* (aka bilinear filtering) or *point filtering*. Letter would allow you to fake some step like (cel) lighting where you do not lay out the NdotL product linearilly over the surface but have some fancy remapping – like an s-curved stepped ramp e.g..

Use *smoothSampling* in case you want smooth lighting over the entire surface. In case you use *PointSampling* instead, things get a bit more complex… as the shader does not do simple point filtering but does two texture lookups and lerps the result in order to get at least some kind of anti aliasing here.

Using a 3px wide stepped ramp texture and SmoothSampling will result in a fully smoothed lighting. No steps at all.

Using a 128px wide stepped ramp texture and smooth filtering instead will reveal the steps and add some blending or "anti aliasing" between the steps. The larger the texture, the smaller the blend area.

The package contains three different 3 Steps Ramps which let you quickly check the final results.

PointSampling is good for nonlinear stepped **small** ramp textures. If you have a linear ramp texture consider using *Steps* instead,

SmoothSampling is good for A) smooth ramps or B) **large**, in our case wide, individually stepped ramp textures.

Off If set to *Off* the shader will use the step function to calculate lighting as described above.

SmoothSampling Use this option whenever your ramp texture actually contains a smooth ramp. Checking *SmoothSampling* will activate bilinear sampling.

PointSampling If *PointSampling* is chosen the shader will use point sampling but actually it will sample the ramp texture twice in order to at least add a tiny amount of anti aliasing to the final result.

The ramp or gradient is sampled from the **red color channel** of the provided texture. Not the best choice when it comes to e.g. DXT1 texture compression. So if you are targeting PC, Mac or consoles you may consider using *R compressed BC4* instead. Tiny ramps like a 8x1px ramp will be stored as *RGB8 UNorm* anyway.

Occlusion Drives the amount of ambient occlusion (and may tweak the value coming from the mask map if this is enabled). A value of 0.0 will fully suppress diffuse ambient lighting and specular reflections.

Advanced Toon Lighting

Colorize Main Light If set to 1.0 the directional light will colorize all pixels of the material — even the unlit ones. NdotL based light attenuation (using *Step* or *Ramp*) and shadows will be ignored.

Colorize Add Light If set to 1.0 additional lights will colorize all pixels of the material — even the unlit ones – based on distance attenuation only. NdotL based light attenuation (using *Step* or *Ramp*) and shadows will be ignored.

Additional lights colorize the shaded parts. **Additional lights do not colorize the shaded parts.**

Light Color Contribution If set to 0.0 the shader will fully desaturate all lights and only take the resulting luminance.

Light Falloff This parameter only affects spot and point lights and lets you control their fall off. If set to 0.001 the shader will more or less ignore the light's smooth falloff and may give you a more toony style.

Use "Light Falloff" in conjunction with the built in light features such as "Inner/Outer Spot" angle.

Specular Toon Lighting

Specular Toon Lighting is based on *BlinnPhong* specular lighting (no PBR).

Enable Specular Highlights If checked the shader will add specular highlights.

Anisotropic Specular If checked the shader supports anisotropic specular highlights which can be rotated to follow the the bitangent or tangent. ***Please note:*** *Anisotropic specular is based upon GGX aniso which has a wider tail than the blinn phong specular highlights usually used. So when turning Anisotropic Specular on you will have to adjust the smoothness to get a similar result regarding the size of the highlights.*
Anisotropic specular highlights are slightly more expensive.

> **Anisotropy** Lets you define the anisotropy direction along bitangent or tangent. A value of 0.0 results in isotropic reflections.

Energy Conservation If checked *Smoothness* will not only affect the size but also the brightness of the highlights (a bit PBR here…). If unchecked *Smoothness* effectively will only drive the size of the highlights.

Specular Specular color which is only used to colorize the highlight. It does not affect anything else (no PBR).

Secondary Specular Secondary specular color. In order to use this feature you have to assign a *MaskMap*.

Smoothness Drives the size and maybe brightness of the specular highlights. It also drives specular reflections in case these are enabled.

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.*

Toon Rim

Rim Color Rim color.

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 Attenuation Drives the influence of diffuse lighting (angle attenuation and shadows) on rim lighting.

Rim Attenuation = 0 Here you will get full rim lighting even in the shaded areas

Rim Attenuation = 1 Rim lighting gets canceled by NdotL and shadows.

Toon Shadows

Receive Shadows If unchecked the material will not receive any shadows.

Shadow Offset Lets you offset the casted shadows in order to prevent or minimize self shadowing artifacts. *Default is 1.0.*

Shadow Falloff Lets you define the edge or feather of the real time shadows. *Smaller values will result in a sharper edge. A good value is 0.5.*

Shadow Bias Directional Defines the shadow strength of the directional light. Values > 0.0 will reduce the shadow strength.

Shadow Bias Additional Defines the shadow strength of the additional lights. Values > 0.0 will reduce the shadow strength.

Toon Outline

Only available in the Toon & Outline shader

Color (RGB) Alpha (A) Color (RGB) and opacity (A) of the outline.

Width Width of the outline. *Depends on the scale of the object unless you check "Compensate Scale" or "Calculate width in screen space".*

Compensate Object 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…*

ZWrite Outline Lets you define if the outline writes to depth or not.

ZTest Outline ZTest method used by the outline pass. *Default is LessEqual. When using alpha testing set this to Less.*

Culling Outline Culling used by the outline pass. *Default is Front.*

Surface Inputs

Color (RGB) Alpha (A) Diffuse color in RGB and Alpha in A.

Shaded Color (RGB) Darkened diffuse color which is used on unlit pixels.

TextureMode

> **Off** Neither the *Albedo (RGB) Alpha (A)* nor the *Shaded Albedo* texture will be sampled.
>
> **One** Only the first diffuse texture will be sampled.
>
> **Two** Both diffuse textures will be sampled.

Albedo (RGB) Alpha (A) The diffuse texture that will be applied to lit pixels. Opcity might be stored in the alpha channel in case you want to use alpha testing or alpha blending.

> **Tiling Offset** Tiling and Offset applied to all texture lookups.

Shaded Albedo (RGB) The diffuse texture that will be applied to unlit pixels.

Enable Mask Map Check this in case you want to use a Mask Map

> **Emission (R) Specular (G) Occlusion (B) Smoothness (A)** This *Mask Map* stores the emission mask in the red, specular in the green, occlusion in the blue and smoothness in the alpha channel.
> **Specular** here is not a value or multiplier but lets you fade between the *Specular* and *Secondary Specular* color. White pixels will use Specular, black pixels will use the *Secondary Specular* color.
>
> **Emission Color** Gets only applied if the Mask Map is enabled

Enable Normal Map Check this in case you want to use a Normal Map

> **Normal Map** The normal map
>
> **Normal Scale** Lets you scale the normal

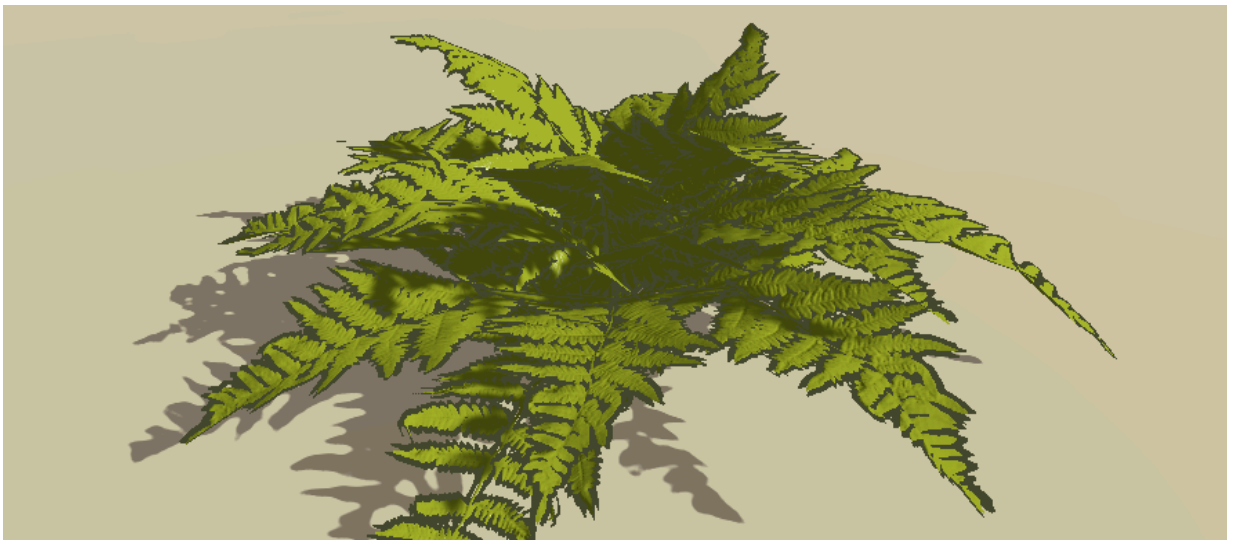
Please note: Further inputs like *Rim Lighting*, *Stencil* or *Advanced* are just common Lux URP Essentials features and are described in the original documentation.

Alpha Testing and Outlines

When using the *Toon & Outline shader* alpha testing is really tricky as the way we usually draw the outline (by scaling the geometry, then rendering the back faces) just fails. Instead the shader here will read the alpha map several times and shuffle it around. Details are described here →

In order to avoid gaps between the shaded parts and the outline we do not rely on clipping inner parts of the outline based on alpha (which would need an additional texture lookup as well) but use the depth buffer instead:

ZTest Outline should be set to *Less*. As the outline is drawn after the shaded parts this will guarantee that the outline does not stomp on the shaded parts.



Alpha tested outline: Not perfect but better than nothing.

Toon Lighting V2 Sub Graph

The inputs of this subgraph are “grouped” more or less to fit the grouping in the inspector of the HLSL shaders which is also used to structure the chapter *Shader features and Inputs*.

If you edit the provided Shader Graph “Lux Toon Graph V2” you will see these groups.

So I will cover only the **missing** inputs here.

Please note: Something that should be mentioned here explicitly is the fact that the provided shader graph example uses a custom ShaderGUI. The reason for this is the fact that Shader Graph does not let you mix regular shader parameters and keywords: Keywords are always appended at the bottom of the inspector which in most cases make them appear very far away from the feature (settings) they drive. It also does not support advanced or even custom material property drawers — like the int slider for the *steps* parameter. So the custom shader GUI just makes the GUI a bit nicer.

Surface Options

Surface Options in general have to be defined in the master node of the shader graph.

Alpha To Coverage or **Color Mask** like other advanced fixed function inputs are not available when using Shader Graph.

Specular Toon Lighting

Secondary Specular should be calculated up front. See *Mask Map* below.

Toon Shadows

Shadow Offset We can not tweak this parameter using Shader Graph.

Toon Outline

We have no possibility to define a custom pass using Shader Graph. So this option is fully missing. You have to assign a 2nd material instead using the provided *Toon Outline* shader.

Surface Inputs

Surface Inputs usually are what you do in your shader graph. So I only added a few simple nodes here as some kind of blueprint.

Mask Map Everything the mask map does actually happens before it comes to toon lighting. Tweaking occlusion, smoothness or the specular color all can happen before plugging the final results into the subgraph. This is what the HLSL shaders do as well.

Toon Outlines in Unity 6

Since Unity 6 and the new *GPU Resident Drawer* break the old style outline rendering I had to rework toon outlines: Instead of using a custom "two pass" shader or adding two materials to a mesh which only has one submesh, outlines now are rendered using the URP Renderer Features.

Using the Renderer Features to draw the outlines fixes issues caused by enabling *depth priming* automatically.

HLSL Toon Shaders

The provided "LuxURP_OutlinesRendererFeature" automatically handles outlines defined in the "Lux URP Toon & Outline HLSL" shader. The rendered outlines are driven by the outline settings in the base material (color, width etc.).

Shader Graph Toon Shaders

In case you want to add outlines to objects using a shader graph based toon material things are a bit more complicated. You can either:

A. Use the old style **2 materials approach** – so adding an outline material on top of the base toon material. This needs you to add the "DisallowGPUDrivenRendering" script in case the *GPU Resident Drawer* is enabled.
This allows you to use different outline materials.

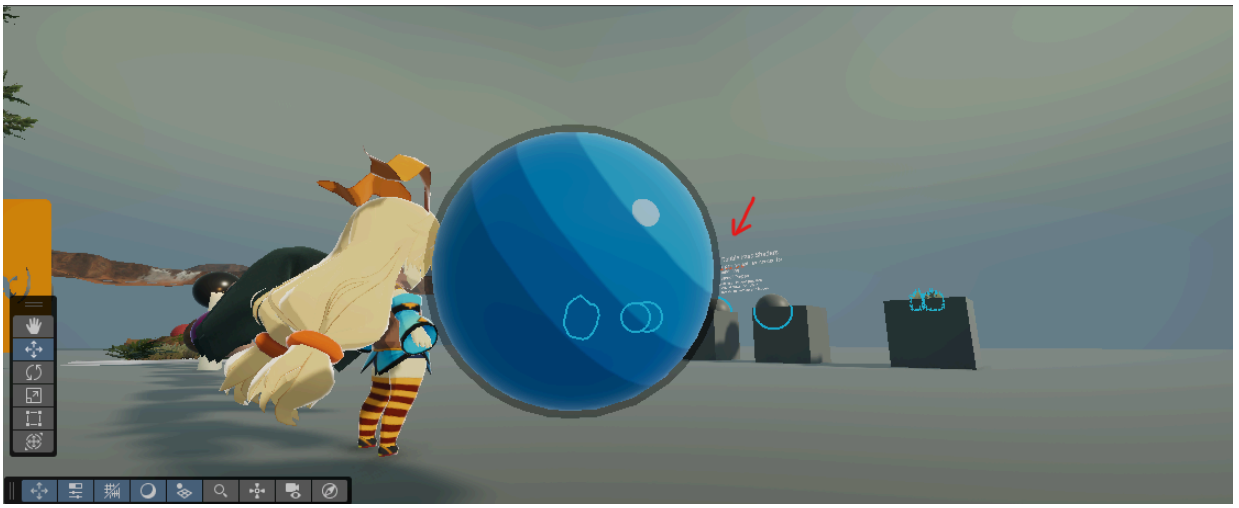
B. Use the built in **"Render Objects" Renderer Feature**, select the objects you want to add outlines to by limiting the effect to certain layers and define an outline material as material override. Please refer to the URP documentation to find out more.
This allows you to use only one outline material.

LuxURP_OutlinesRendererFeature

In order to make toon outlines from the "Lux URP Toon & Outline HLSL" shader being rendered you have to add this renderer feature to your *Universal Renderer Data* – just where the built in Renderer Features such as *Screen Space Ambient Occlusion* or *Decals* are added.

Layer Mask Lets you specify a layer. Only the outlines of objects set to the selected layers will be rendered.

Injection Point Lets you specify when the outlines are drawn. If your outlines are fully **opaque** choose *"After Rendering Opaques"*. But if your outlines are **semi transparent** there is no right injection point unfortunately…



If you use “Before Rendering Transparents” and “ZWrite” in the outline settings of the toon material is set to “On” no transparents will be rendered behind the outline but just the skybox. As shown above the transparent white text does not shine through the transparent outline.

If you use “After Rendering Transparents” the “Fast Outline Double Pass” shader may punch holes into the depth buffer which will make the toon outline filling these holes. While the transparent text shines properly through the transparent outline, the latter gets rendered on parts of the toon sphere where the “Fast Outline Double Pass” shader invalidated the depth buffer to do its job.

You may use the Layer option and add the renderer feature multiple times to split up the rendering of your outlines using different injection points.