Lux SRP Grass Displacement

The Lux SRP Grass Displacement package contains an early preview of a fast yet versatile grass displacement solution for the Universal Render Pipeline and was designed to work in conjunction with the Lux LWRP/URP Essential grass shader and URP version of the Advanced Terrain Grass package.

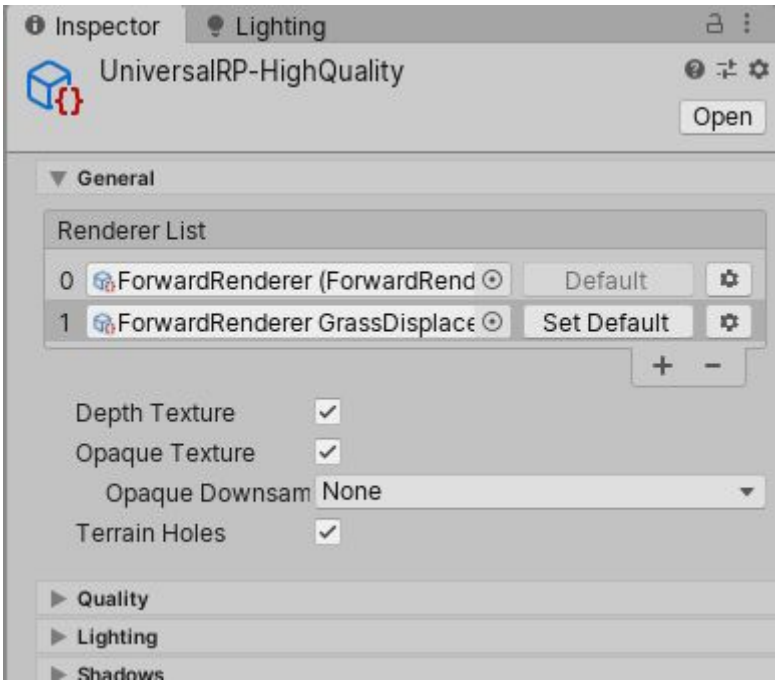
Compatibility

Lux SRP Grass Displacement needs URP. LWRP is not supported. It has been successfully tested on DX11, GLES 3.2 and Metal (both desktop) and Android using a Google Pixel (1). VR only supports *single pass* rendering at the moment.

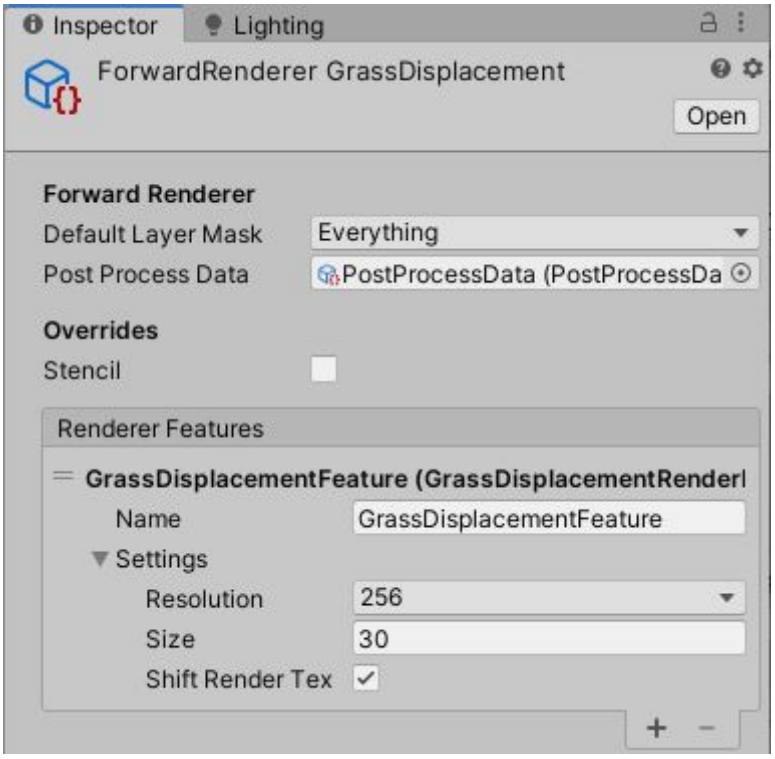
Table of Content

Getting Started

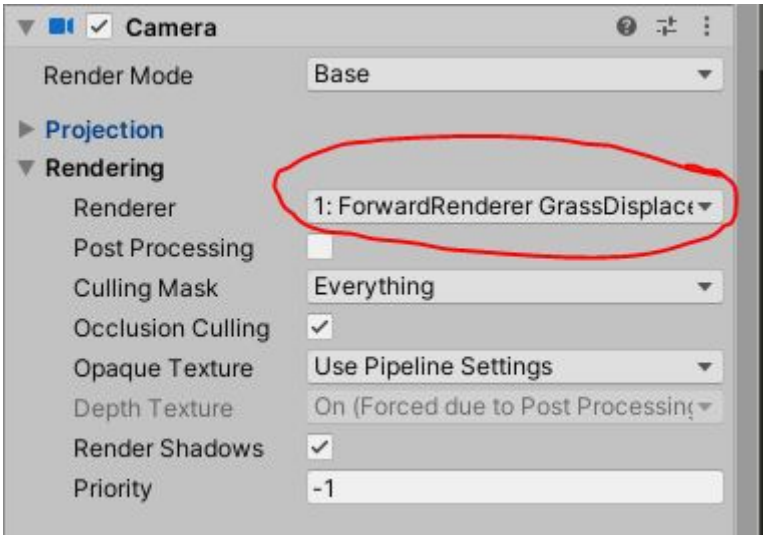
1. Import the provided package then edit your URP Settings (like: *UniversalRP-HighQuality*), open the *General* foldout and add another renderer by hitting the “+” button. Then assign the *ForwardRenderer GrassDisplacement* which is provided with the package.



2. Select the *ForwardRenderer GrassDisplacement* renderer asset in the project tab so that is shows up in the inspector and make sure that the *GrassDisplacementFeature* is assigned as *Renderer Feature:*



3. Now open the included demo (Grass Displacement RenderFeature Demo), find the camera, open its *Rendering* foldout and assign the *ForwardRenderer GrassDisplacement* renderer. This will activate our custom scriptable render pass.



4. If everything worked out fine you now should see a preview of the generated render texture (*Gizmos* have to be activated in the scene and game view) in the upper left corner of the scene and game view which is displayed by the *DisplayGrassDisplacementTexture* script also attached to the camera.
5. Enter play mode and watch how the spheres and the turbines displace the grass.
6. Different packages may contain additional demos (like the package that ships with ATG). When testing the additional demos, please check that the proper *Forward Renderer* is assigned to the main camera as described in 3.

How it works

Lux SRP Grass Displacement uses a common displacement technique based upon an offscreen rendered displacement texture which gets sampled by the grass and vegetation shaders to actually apply the displacement.

But unlike other solutions it does not use a second camera to render the displacement texture but utilizes Unity's scriptable renderpasses instead – which saves us a lot of overhead and minimizes the work the CPU has to do: no extra culling, no extra light culling. It also does not reclaim an extra layer which is fine as we only have 32.

The scriptable render pass sets up a *virtual orthographic camera* looking top down from more or less the current camera's position, which only renders objects (displacers: simple meshes like quads or particle systems) using a shader whose only pass is named: *LuxGrassDisplacementFX*. This name guarantees that the displacers are not rendered by the regular camera but the virtual only.

The result is stored in a render texture and send to the grass shader – together with the size and position of the displacement texture in world space. The shader then checks if the processed vertex is within the bounds of the displacement texture and – if so – samples and applies the displacement.

No matter how many displacers are active and affecting a single grass mesh the shader will always only do a single texture lookup per vertex at most.

Putting it all together we can efficiently render the displacement texture (thanks to the scriptable render pass) as well as render the displacement with just one texture lookup in the vertex shader (if needed).

Please note: As we are using a scriptable render pass displacers must be “visible” to the actual camera to be rendered. So they might not appear in the top down projected displacement texture although they would be visible to the top down camera. This is absolutely fine in 99% of all use cases and speeds up rendering.

The demo

Coming back to the demo we have various objects displacing the grass:

- The **blue sphere** is driven by Physix. Its displacement is driven by a particle system (PF Particle System (Trail Follower blue)) that smoothly follows the sphere and emits particles according to velocity.
- The **green sphere** also is driven by Physix. Its displacement is driven by a simple quad (Quad(Follower green)) that smoothly follows the sphere.
- The **red sphere** is driven by an animation. Its displacement is driven by a particle system (PF Particle System (Trail Follower red)) that smoothly follows the sphere and emits particles according to velocity.
- The **yellow spheres** show how you can dynamically adjust the strength of a simple displacer according to its distance to ground using the *ControlDisplacer* script.
- The **orange cube** uses a simple quad based displacer which is parented under the cube. As its displacement material has *Rotate Normal* checked you may rotate it around the y axis. The displacement source texture actually has some information in the B color channel which pushes down the vertices in the inner of the rectangle.

The reason why most displacers use the *Smooth Follow* script is the fact that (most) they roll and rotate and i did not want Unity to rotate particle systems… Other objects with a different movement may just use different scripts to move their displacers or even simply contain them as child object. **Please note:** Rotating the displacer and its texture will break the directionality of the displacement unless you check *Rotate Normal* in the *Simple Displacement* shader. See the orange cube.

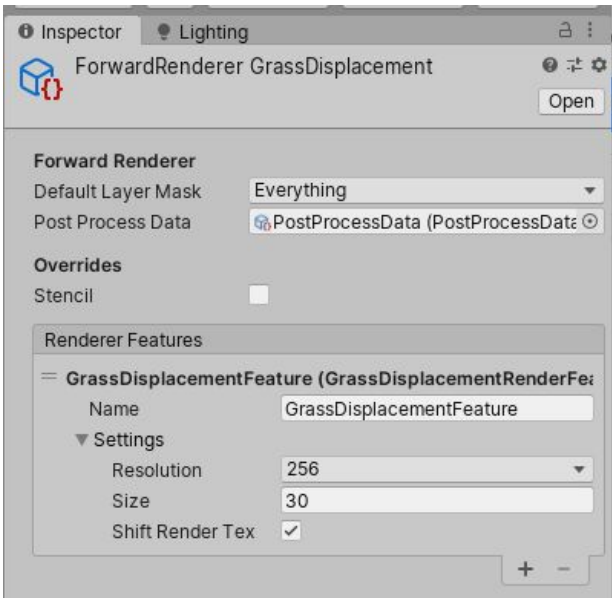
- The **turbine** uses a different particle system (and some supporting geometry) which gets rotated by script.
- The **moving turbine** uses the same particle system as the turbine and gets animated up and down by script. It uses the *ControlDisplacerParticleSys* script which checks its distance to ground and adjusts the alpha of the emitted particles accordingly.

Please note: Grass in the demo scene use a tweaked shader: *Lux LWRP Grass TextureDisplace*. The regular *Lux LWRP Grass* shader will just ignore any displacement.

Tweaking the displacement

Size and Resolution of the final Displacement Texture

Size and Resolution of the final displacement texture which gets sampled by the grass shader can be set in the assigned Renderer → *Render Features* → *GrassDispalcementFeature* → *Settings*:



Resolution Resolution of the rendertexture in pixel. *Keep it as small as possible to speed up rendering.*

Size The size or coverage of the rendertexture in meters and world space. As the render texture is centered on the camera by default a size of 16 would give you 8 meters in front of your camera covered by the displacement texture.

Shift Render Tex If checked the virtual camera will be pushed a little bit forward so more of the rendertexture is in front of the actual camera and you can reduce the *Size* of the render texture. This however will make the rendertexture a bit more unstable in space so if you just rotate the camera grass might suddenly change between being displaced and not being displaced.

Tweaking the displacement per displacer

The most important ingredients are the displacement source texture (usually some kind of normal) assigned to the *Simple Displacement FX* shader and its alpha value.

Displacement source texture

While the displacement source texture RGB channels drive the *direction* in which the vertices will be displaced the A (alpha) value drives the final *strength*.

The provided *T Displacement Normal* displacement source texture contains a radial normal in RG (B is set to white) and a smooth falloff in A. So all vertices will be pushed outwards from the center according to the fall off.

- **RG** define the displacement along the world xz axes.
- **B** lets you add some boost when it comes to pushing the vertices downwards. If you do not want this extra boost just leave it completely white. Setting B to 0.0 (black) will push vertices down at full strength. *The orange cube in the demo shows an example of using B to push vertices downwards.*
- **A** may contain a mask.

This texture has to be imported *using sRGB (Color Texture)* to be unchecked. Do not import it as *Normal Map* and make sure that you have chosen a high compression quality. You may even consider importing it uncompressed as RGBA 32 bit to avoid all compression artifacts in case it is a small texture (like 128x128 pixel).

Simple Displacement FX shader

All renderes that shall contribute to displacement have to use the *Simple Displacement FX* shader which is listed under: "Lux SRP Displacement/Simple Displacement" in the shader dropdown. This shader is quite basic and simply does some alpha blending according to the texture's alpha and vertex color.

The shader uses *Tags{"LightMode" = "LuxGrassDisplacementFX"}* which makes it visible only for the virtual orthographic camera. If you need a visual feedback of the diplacer in scene and game view choose the *Simple Displacement FX Debug* shader which has two passes and gets rendered by the virtual orthographic camera as well as by the scene and game view cameras.

Shader Inputs

Displacement Source Texture The displacement source texture which drives directionality and strength. RGB contains a normal while A contains the fall off. This texture has to be imported *using sRGB (Color Texture)* to be unchecked.

Dynamic Alpha If you want to change the final alpha by script using a *MaterialPropertyBlock* (in order to take the distance to ground into account) you have to check this.
Checking Dynamic Alpha will make the shader not being compatible with the SRP Batcher. So only check it if you need it or consider enabling GPU Instancing.

> **Alpha** Final alpha multiplier if *Dynamic Alpha* is checked. This usually gets set by script using a MaterialPropertyBlock.

Rotate Normal In case you rotate a displacer/displacement source texture the normals have to be rotated within the shader to fit. So check this if your rotate the displacer.
Please note: *The mesh used by the displacer (quad or particle systems) needs normals and tangents. This might need you to adjust the vertex stream of your particle system.*

Scr Blend Mode and **Dst Blend Mode** define the blending. Regular alpha blending needs *ScrAlpha* and *OneMinusSrcAlpha* which is the default.

This shader works for particle systems as well as for single quads but it does not support texture sheet animation.

Using simple geometry

When using simple geometry like a quad just make sure you use the *Smooth Follow* script so the quad will always face the virtual top down camera and does not rotate around the y axis which would break the directionality of the displacement.

Alternatively you may try to check *Rotate Normal* in the material used by the displacer. This would allow at least rotations around the y axis. Other rotations are not supported by the shader.

Using particle systems

Using particle systems of course is the most powerful way as it allows you to create trails and complex effects like explosions.

Animating the *Alpha* over lifetime is pretty important to get a smooth displacement.

The *Render Mode* has to be set to *Horizontal Billboards* so the virtual camera renders the particles properly.

Everything else is up to your creativity :)

Please have a look into the provide particle systems and prefabs.

Distance to ground

Imagine you attach a particle system to an enemy to make it create trails. Then this enemy may jump or climb up a wall. In this case you do not want it to displace any grass anymore. In order to achieve this you can either stop the particle system from emitting further particles – or you can tweak the *start color* alpha value and make the particles less visible and even fully transparent. Latter is the way i have chosen for the example scripts.

ControlDisplacerParticleSys.cs

This script does a raycast to determine the distance to ground and adjusts the start color's alpha of the currently emitted particles accordingly. This means that the distance to ground of the particle system at the moment a new particle is emitted drives the opacity of the new particle. The distance of the particle to ground over lifetime is not taken into account.

Max Distance Max distance to ground before the emitted particle will be fully transparent.

Fall Off Lets you adjust how the distance influences the opacity (uses power).

Layer Mask Here you have to assign the layer your ground/terrain is assigned to as the raycast only will take a single layer into account.

Debug Ray If checked the script will draw the ray in green (if there is a hit inside the max distance) or red (otherwise).

ControlDisplacer.cs

This script does a raycast to determine the distance to ground and adjusts the *Alpha* of assigned material using a MaterialPropertyBlock. Please note that you have to check *Dynamic Alpha* in the material inspector.

Max Distance Max distance to ground before the displacer will be fully transparent.

Fall Off Lets you adjust how the distance influences the opacity (uses power).

Layer Mask Here you have to assign the layer your ground/terrain is assigned to as the raycast only will take a single layer into account.

Debug Ray If checked the script will draw the ray in green (if there is a hit inside the max distance) or red (otherwise).

Please note that both scripts are only blueprints. In most cases you already know the distance to ground of your enemy e.g. so there is no need to fire a 2nd ray. Or you may use a unity terrain. In this case *Terrain.SampleHeight* might be more efficient than doing a raycast.

Overlapping Displacers

Currently the diplacers are rendered simply using alpha blending so they may just overwrite each other. Summing up the displacement would be nice but required an RGBAhalf rendertexture which made things more expensive…

Using alpha blending makes the final render texture be driven by the order Unity draws the displacers which should be back to front as they are transparent. You may however make sure that a strong displacer never gets hidden by a weaker one by editing its material and raising the *Render Queue*.

Adjusting the grass material

The tweaked *Lux LWRP Grass TextureDisplace* shader comes with a new parameter → *Displacement* → *Strength* which allows you to control the impact of the displacement texture.

In case you use **ATG URP** you will notice that the grass shader has a section *Touch Bending*.

Enable Touch Bending Check *Enable Touch Bending* to make the shader sample and apply the displacement texture.

Sample Size Lets your tweak the position at which the displacement map gets sampled. 0.0 → at pivot, 1.0 → at vertex position.
Use 0.0 e.g. for single plants such as flowers.

Displacement XZ Lets you control the displacement strength along the XZ axis.

Displacement Y Lets you control how much the vertices will pushed downwards.

Depending on the baked bending you may have to use rather high values her like 3 or 4.

Normal Displacement Lets you control the strength and direction in which the normal gets bent.

TODOS

- Add support for texture sheet animations.
- ~~Add support for rotation.~~ *Done*
- Add support for trail renderes.
- Add support for single pass instanced.

After all it is far from being finished. But a first step. Feedback is welcome.