SHADER WORLD

A plugin by Maxime Dupart: Twitter

Documentation

Discord Link

I.Presentation

Shader World plugin is a GPU-accelerated framework for World creation, and population. It's incredibly fast, does not require a strong CPU, and works with any entry level GPU that supports compute-shaders : most GPUs and smartphones since 2013. It allows users to generate at runtime detailed, vast, rich and interactive worlds from the comfort of the well known Unreal Engine material editor.

It features: runtime layers, compute-shader powered density based asset spawning, a high performance fully triplanar landscape material, adaptive terrain rendering topology for high quality output, communication between Shader Worlds (ocean/terrain), a collection of premade high quality noise nodes (Material Functions) ready to be combined, an interactive brush system which can both edit the terrain and the runtime layers.

.

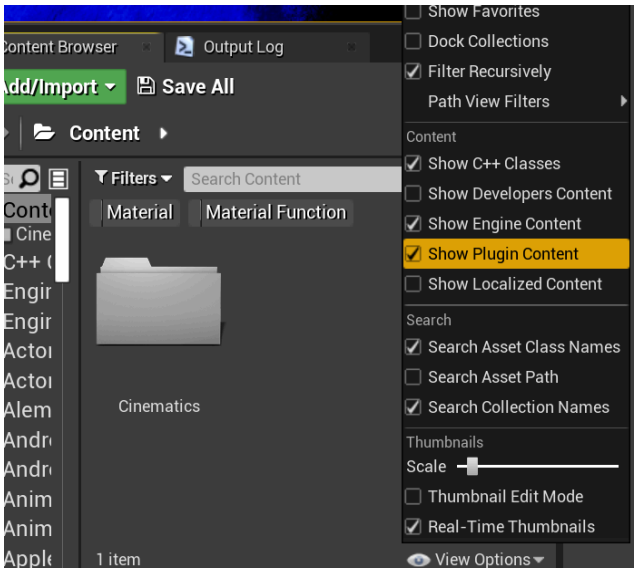
It's built for performance, scalability, and fast iteration.

[Introduction on Youtube]

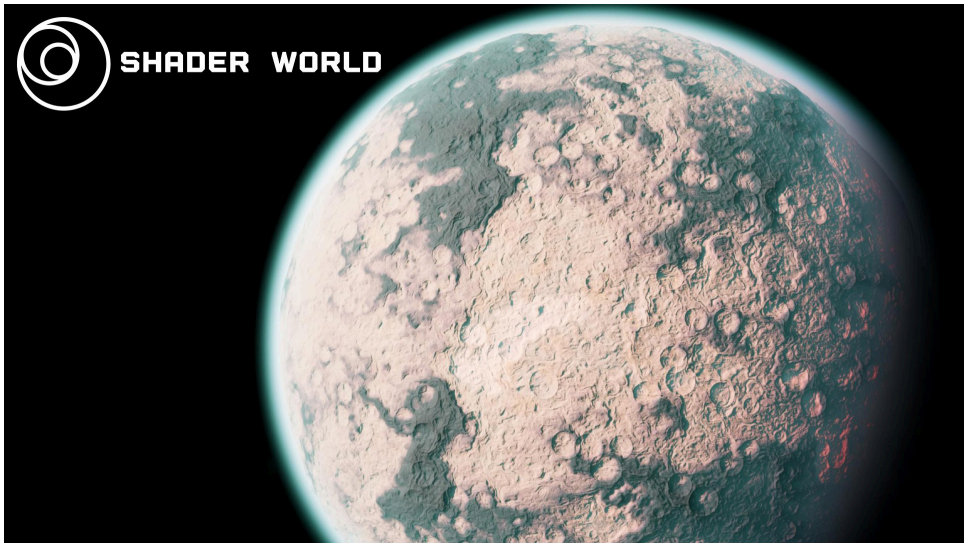
II. Getting Started

1. The world

First of all you need to activate the plugin from the editor plugins menu, then **enable visibility** of *Engine content* and *Plugin content* for your content browser.



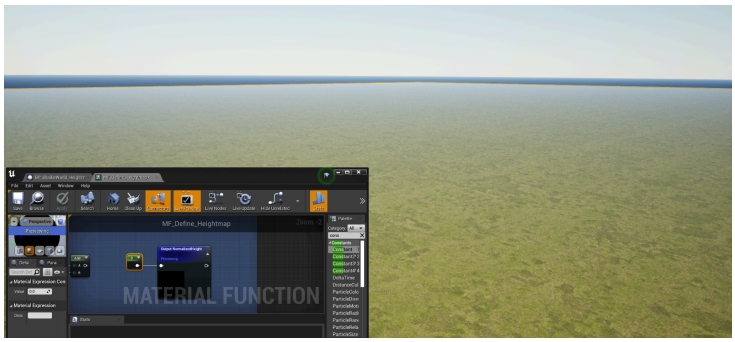
From the **Shader World** plugin *Content* folder, navigate to the *Maps* subfolder and open one of the demo maps. In the world outline select the blueprint named **BP_*NameOfMap*Demo**, within its detail panel you can find its **generator** Material which defines the foundation of how your world looks like, is generated.



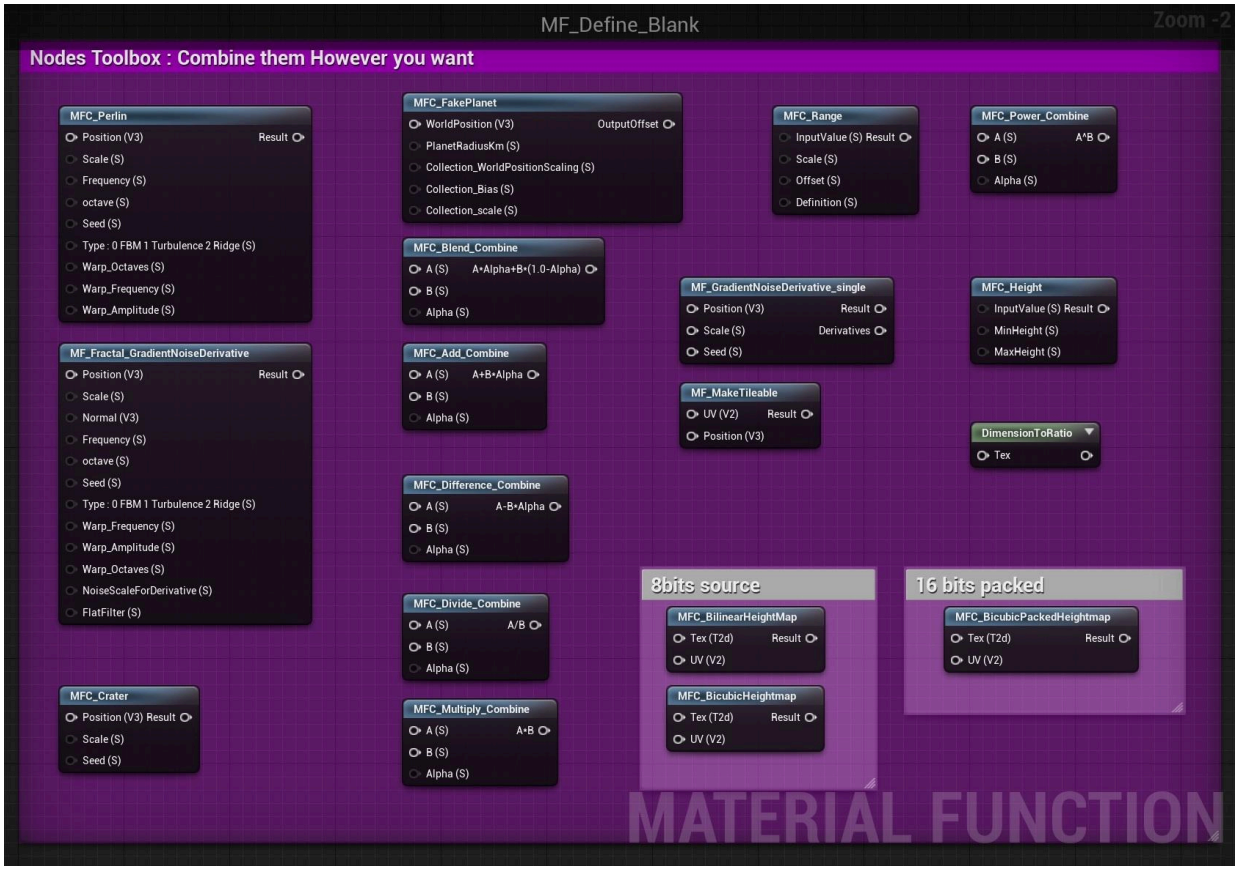
See annex for how this world is generated

If you open the generator material you will find a material function named **MF_Define_*** which is actually responsible for creating your landscape/ocean: for each XY coordinate as input, it outputs a height.

Creating a new world in shaderworld, consists in creating such a material function, using the provided nodes toolbox and the well known Unreal Engine material editor. To visualize its effect, first select the landscape in your currently opened demo map (*better start with **Blank** demo map*), and check **Update Height Data Over time** in its **detail panel** '*Config*' tab: this option is usually used to create oceans, but allows instant feedback when iterating on landscapes. Open the **MF_Define_*** Material Function associated with your current world, now updating every frame, and replace the current computation by a simple constant to the output pin, *compile the material function and observe the result*.



What this material function is doing is: given a world position input, outputs the height of the terrain.



Node Based Workflow, connect those as you wish to create worlds

The intended design of Shader World plugin is to create a **node based workflow** to generate worlds within the material editor, so you don't have to be a professional technical artist to create interesting worlds, similar to other terrain generator software available on the market. The screenshot above displays the current state of the toolbox as of version 1.3, those nodes are *premade* and *calibrated* (inputs being in 0..1 0..10 0…1000 ranges) and can be combined as you want.

In Shader World plugin the nodes are actually material functions, you can develop your own nodes, or even directly write your own shader logic without using them at all.

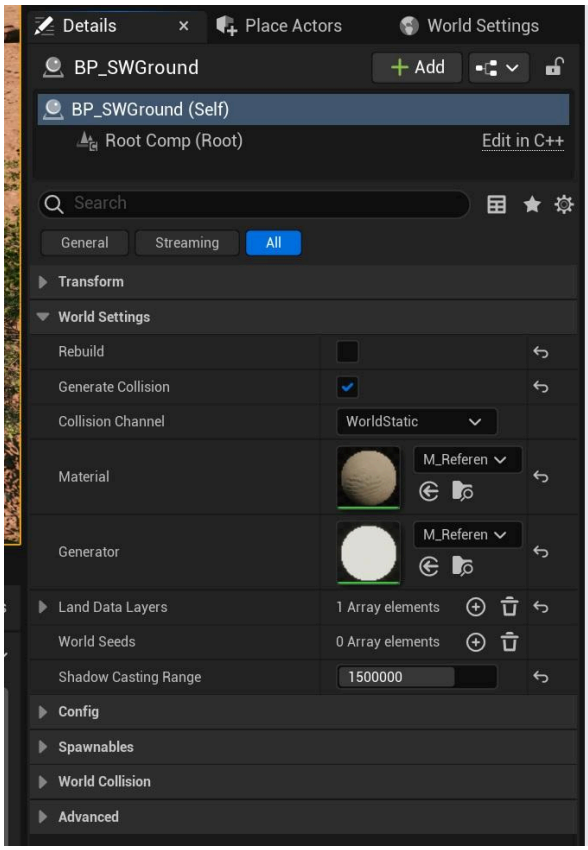
If you check the various demo map **MF_Define_*** material functions you can find some reference on how to use world coordinates to read a heightmap to define your terrain, as well as adding some lower scale noise variations to it, or relying exclusively on noise functions.

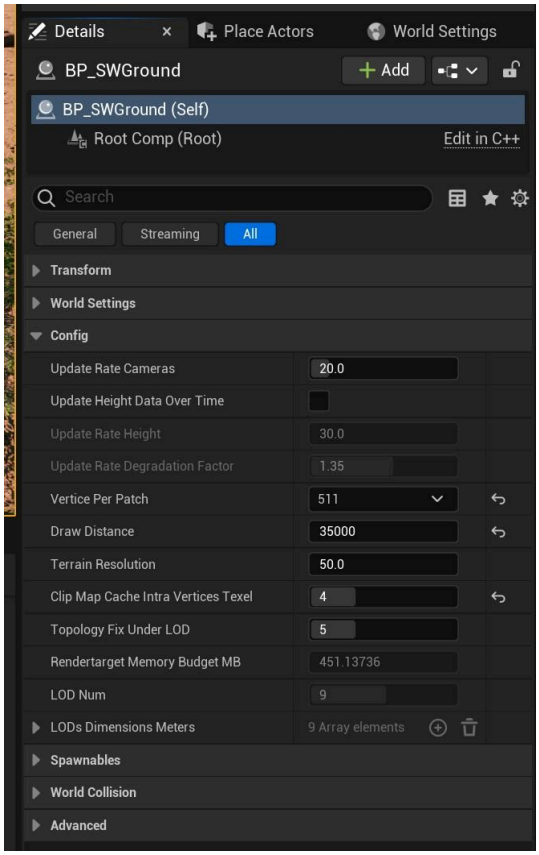
Alongside the **provided toolbox** you can use all the preexisting material functions within the Unreal Engine material editor, notably the **Noise** node (prefer computational noises to avoid precision issues). But we would recommend using Shader World's own noises which have been extensively tested. The toolbox is set to expand over time.

As a general idea you could think of generating a node combination for mountains, another for valleys, another for underwater, and blend between each other given a biome, mapped using a texture or another set of nodes, possibilities are endless.

2. BP_ShaderWorld tabs:

1. World Settings and Config





For the rendering, we rely on *GPU based geometry clipmaps*, if you want to learn more about it, understand what the **vertices per patch**/ring means check the link below: https://developer.nvidia.com/gpugems/gpugems2/part-i-geometric-complexity/chapter-2-terrain-rendering-using-gpu-based-geometry

By default ShaderWorld ‘cache’ the evaluation of the generator at a given location in a texture. The resolution of this cache is independent from the **terrain resolution** which corresponds to the real world distance between two vertices of your terrain ground at its highest quality LOD. **Clipmap cache intra vertice texel** allows to define how many extra height evaluations are gonna be generated in-between your terrain vertices, allowing a much more precise computation of your terrain normal, a better looking shading, than if we were just evaluating at each vertice.

With **Clipmap cache intra vertice texel** to 1, you will only precompute the height/normal of the terrain at each vertice generating your world terrain triangles. The more you increase ‘Clipmap cache intra vertice texel’ the more you add details in between those vertices.

You have the option to add landscape layers in **Land Data Layers**. For each layer you have to specify a **unique name** and a **material to generate the layer**. Each layer can access the data computed before itself:

- Layer 1 can access: Heightmap, Normalmap
- Layer 2 can access: Heightmap, Normalmap, Layer 1
- …

Topology fix Under LOD : Allows to adjust the terrain patch rendering to better fit the topology of the terrain and avoid ‘spikes’ of the terrain landscape. As this computation is expensive, it’s recommended to only use it on the higher quality LODs around the players.

Experimental World Representation

2. Spawnables (Update Height Data Over time **OFF**)

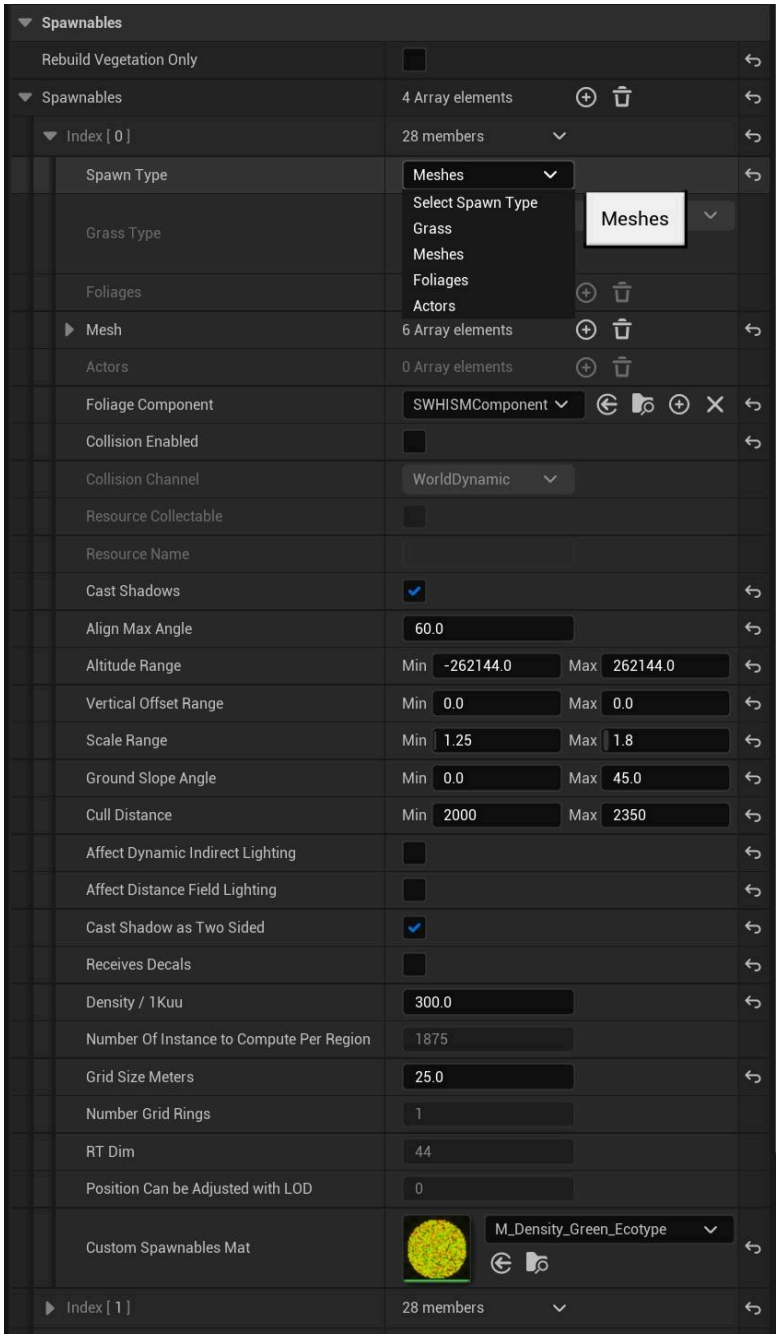
Spawnables are of four different types: *Meshes*, *Grass*, *Foliages*, or *Actors*.

- *Meshes :* you provide one or multiple *Static Meshes*
- *Grass :* you provide a *Landscape Grass Type*
- *Foliages:* you provide one or multiple *Static Mesh Foliage*
- *Actors:* you provide one or multiple *Actors*

Each Spawnable will be computed on a set of grids dynamically placed around the players: **Number of Grid Rings** let you know how many rings of grid will be computed around the player, the less number of grids there is, the less draw calls will be generated, everything being computed on GPU.

To lower the number of grid rings, increase the size of the grids, but be careful to not have too many entities computed per grid (over 10 000 ?).

Example : **Cull Distance Min** indicates at which distance your instanced Meshes will start fading away, and **Cull Distance Max** indicates at which distance your instances won't be rendered anymore (2350 cm | 23.5 meters): You don't need to use 500 meters wide grids if you render instances only at 23.5 meters! Here we use 25 meters wide grids.



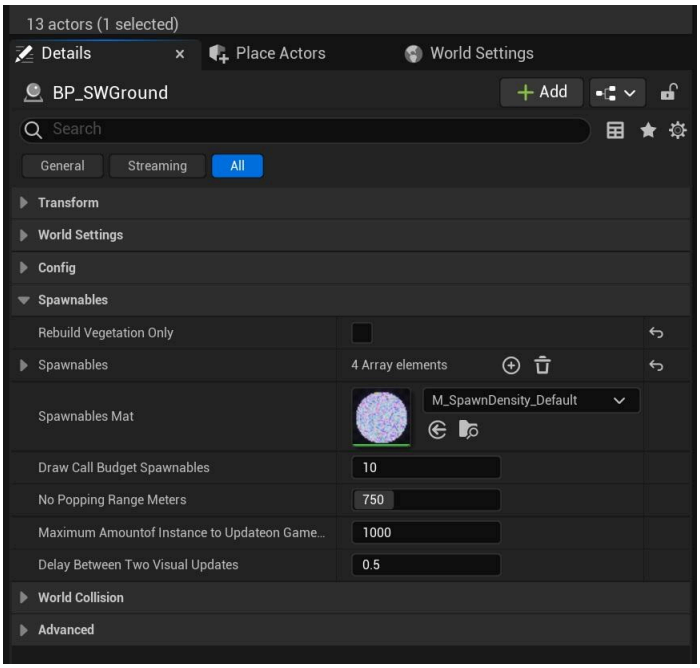
As counter-intuitive as it may appear the **number or rings** is often a **more relevant** information than the **number of instances computed per grid** regarding performances.

The important parameter is whether or not a spawnable has collision enabled: collisions disabled allows 10000 instances per grid or more, as long as the meshes are simple enough for your targeted hardware, but if the spawnable requires collisions it's advised to stay below 400-1000 instances per grids or lower, depending on the complexity of the collision representation.

By default spawnables will use the material **M_SpawnDensity_Default** to generate a density map defining the probability for an asset to spawn.

If you want to add custom logic to a given spawnable, you can specify a **Custom Spawnables Mat**, allowing to define advanced spawning conditions using all the informations at your disposal within the Landscape Data layers, as well as Heightmap and Normal map (see demo_heightmap for its custom grass spawning density function).

As spawnables are computed by drawing materials, you can define a specific **draw call budget** for computing spawnables, which represent how many draw calls will be allowed per frame to compute spawnables, each grid represents one draw call. *Priority order for spawnables computation depends on* view distance (Objects close to camera first)*, collision status (collision enabled first), and being within view frustum or not.*

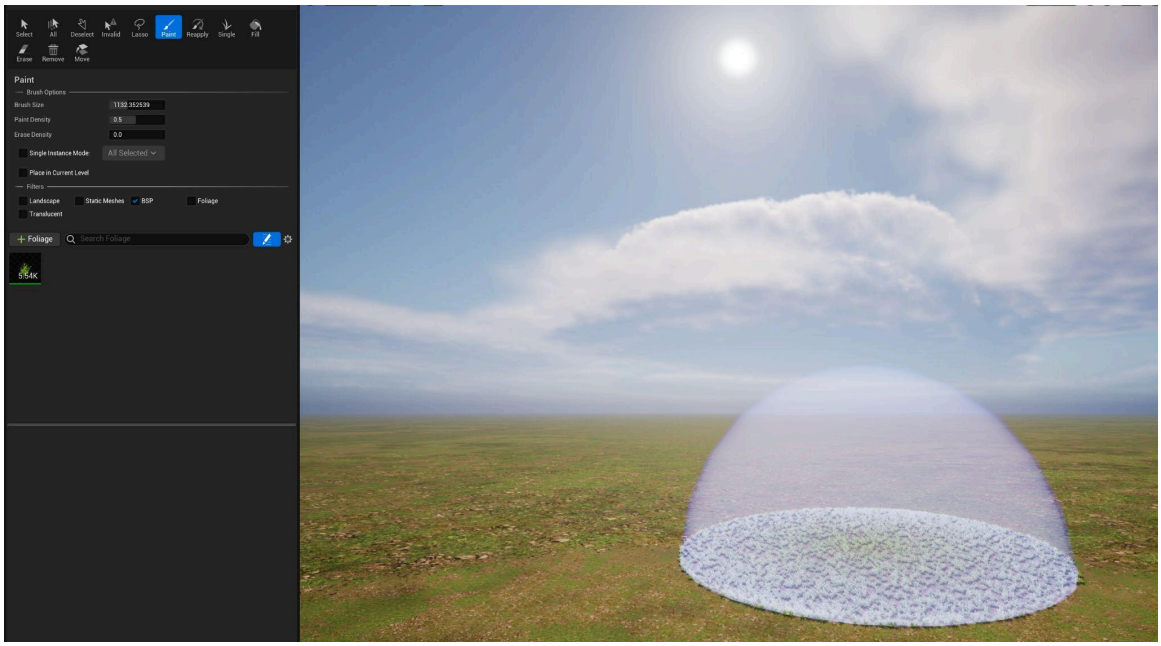


No Popping Range Meters, Spawnables improve their spawning location as you get closer to them: As the landscape ground quality improves as you get closer, spawnables locations are re-evaluated to leverage the higher quality terrain information computed.

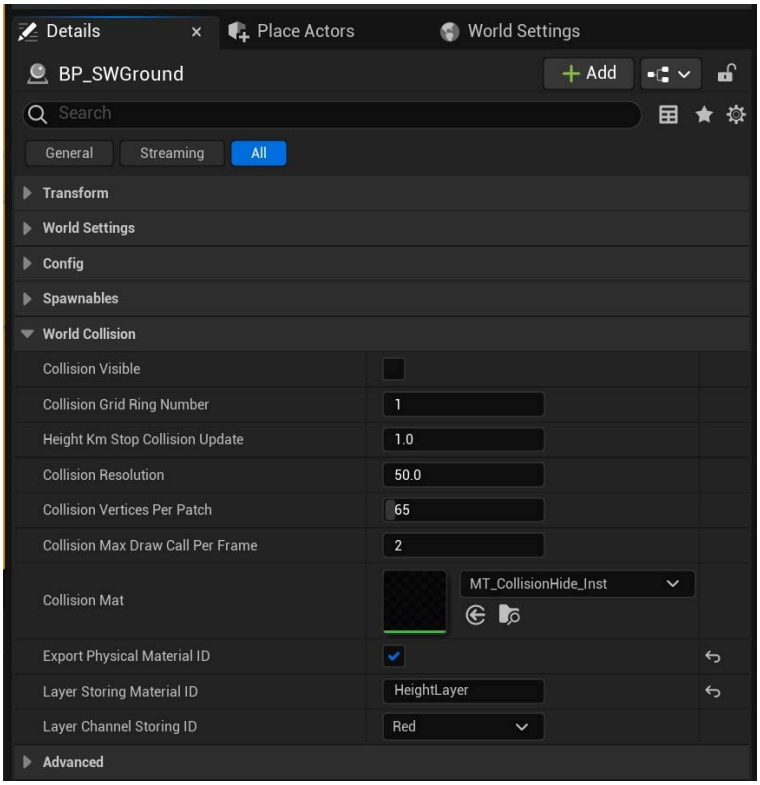
One possible issue is that the higher quality terrain information might indicate that the terrain which seemed suitable once evaluated at lower quality is actually not suitable for spawning an asset (the ground slope is beyond the tolerated range for instance) the instance should then be removed (or the terrain is suitable and it should now spawn). As it looks jarring seeing asset spawning/despawning next to the camera, you have the option to prevent it from happening within a specific range. Default value is 750meters.

Collecting resources: By default spawnables with **collision enabled** also have the possibility to be collected if **Resource Collectable** is checked and a non-empty **Resource Name** is provided, one of the demo maps showcases how to collect large rocks. Note that there is no book-keeping of those spawnables, nor server authoritative management of resources, if the player goes beyond the generation range and comes back, the resource will reappear. A bit like *No Man's Sky* procedural collectable vegetation.

Painting foliage (UE5 Only) : If your world generates collision, you can use the Foliage Mode of Unreal Engine 5 to paint and place foliage on your Shader World terrain, as long as BSP is selected as target, and within the range you're computing collision for.



3. World Collision

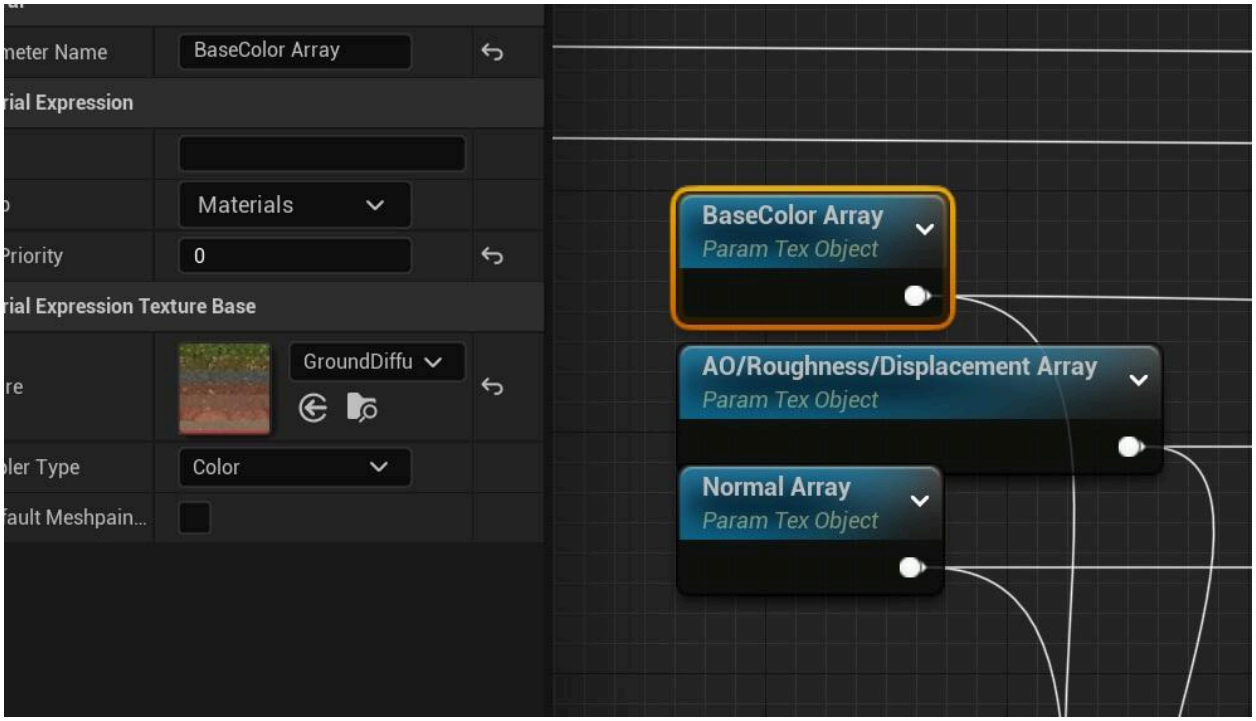


Similar to spawnables computation, collisions are computed in grids around the players. **Collision Resolution** defines the distance between two adjacent vertices for your collision meshes. **Collision vertices per patch** defines the size of patches you will be computing collisions for. When not simulating the game or playing, you can visualize the collision mesh in Wireframe mode in blue, or by checking “Collision Visible”.

Export Physical Material ID, allows to specify that the ground material IDs are available within one of your terrain runtime layers, and to specify in which channel (Red, Green or Blue). By exporting those IDs you’ll be able to identify which material you are hitting during collision events, line traces etc.. Allowing you to generate appropriate sounds and VFX for a given ground material.

By default we provide a highly optimized fully triplanar landscape material using those generated Material IDs as indexes to read three textures arrays storing respectively: *Base Color*, *Normal*, and packed *AO/Roughness/Displacement* of a couple of free Megascan ground materials.

You can of course use any landscape material that you want, given that you convert the landscape specific nodes (layer blend) accordingly.



The provided landscape material and its ground materials arrays

4. Advanced | Clipmap Hack Culling

Our world relies on height being evaluated within a material **on GPU**, as a consequence, the CPU has no information regarding the actual height of the terrain and will cull (hide by not asking the GPU to render) pieces of the ground that it assumes we can't see.

As far as the CPU is concerned we're drawing an infinite flat plane, the higher/lower we are from this original plane, the more it will start skipping pieces of the ground that we can actually see once height is evaluated.

To prevent this issue, instead of sending a flat plane to the CPU we'll alternatively move vertically each point of the plane with +/- **Vertical Range Meters** meters, the bounding box of our terrain patch will then pass CPU culling without issues. If your terrain seems to disappear, increase this value.

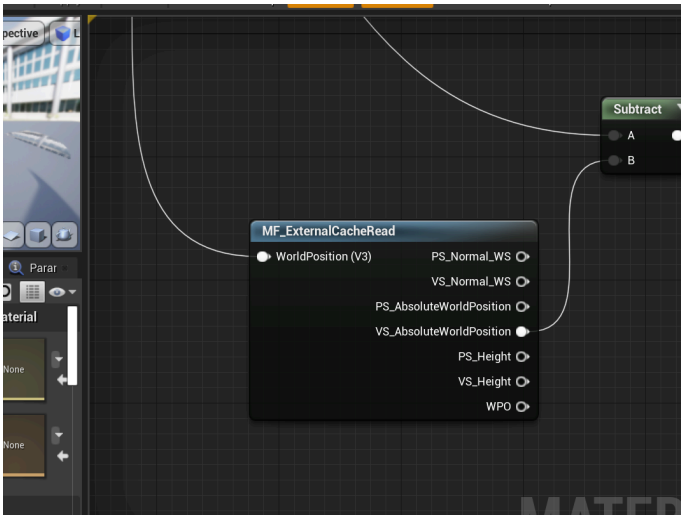
5. Advanced | Dependencies

Allows one ShaderWorld to send its data to another, see **Map_Ocean:** the landscape ShaderWorld sends its height information to the ocean, allowing the ocean to create more interesting shoreline shading and adjust ocean transparency around the shore.

The Landscape ShaderWorld set the Ocean SshaderWorld as a **Receiver**.
The Ocean ShaderWorld set the Landscape ShaderWorld as a **Source**.

The Ocean ShaderWorld can then access Landscape data using the **MF_ExternalCacheRead** material function inside its ocean material.

(*ocean world read heightmap from landscape world in its material for shore blend*)

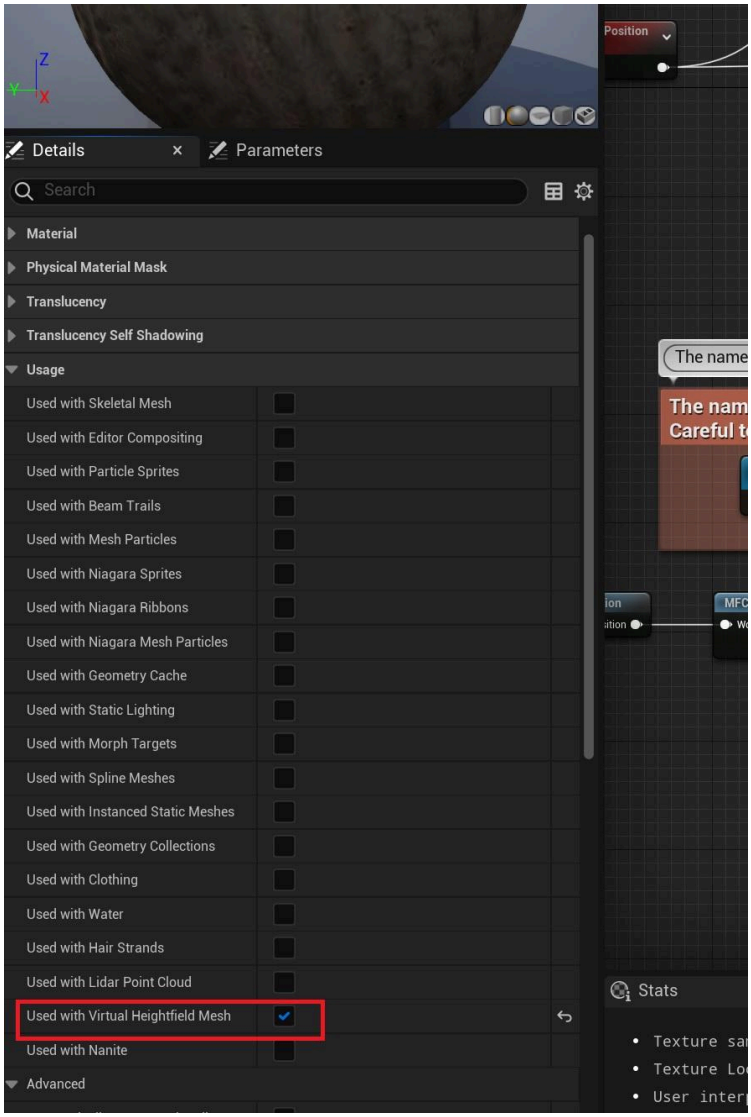


3. Creating a new World

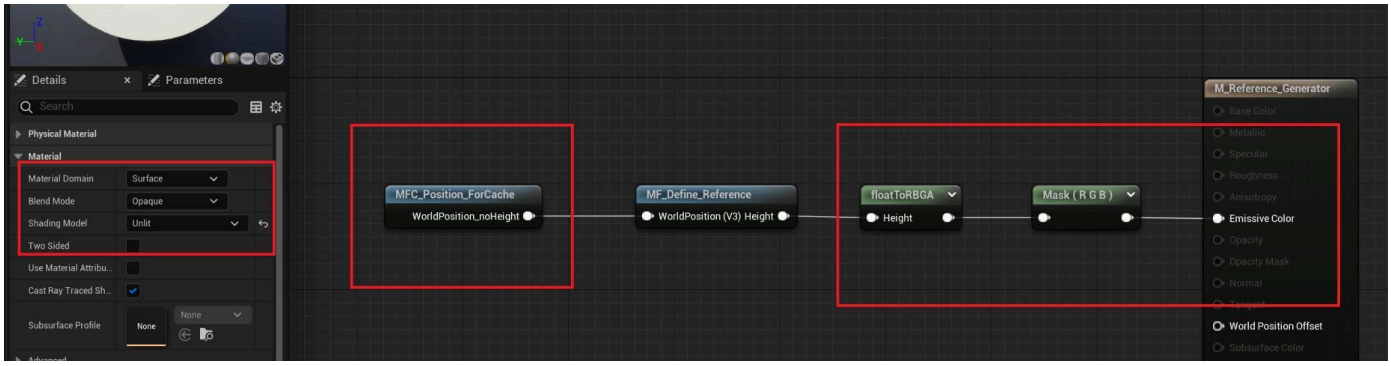
Creating a new Shader World consists at first in creating a generator material and its recommended material function, as well as a landscape/ocean material.

We would suggest creating a dedicated folder in your content browser and creating two new materials, one for generating the world,and the other to simply be the landscape or ocean material.

The terrain/ocean material (not the generator material) has to toggle usage “Used With Virtual Heightfield Mesh”, available inside the material editor, detail panel, usage tab.

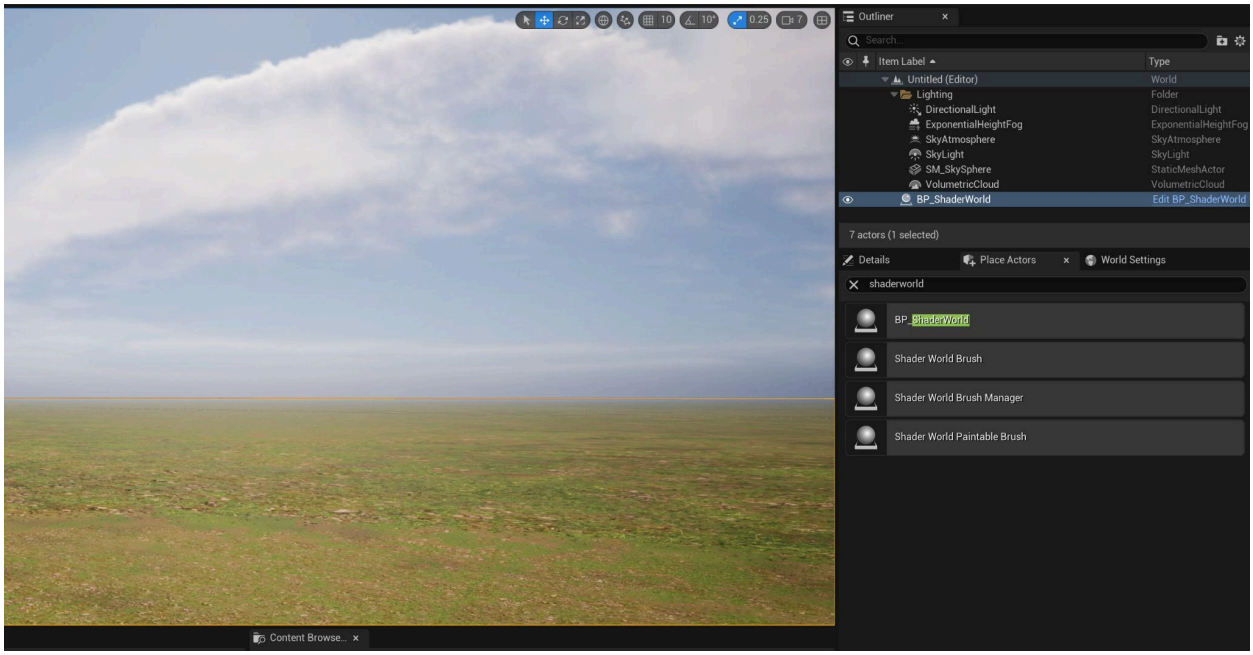


The Generator material has to use the blend mode Opaque, and shading model Unlit.



Generator Material : simply copy the highlighted elements from the demo maps generators (MFC_Position_ForCache, floatToRGBA)

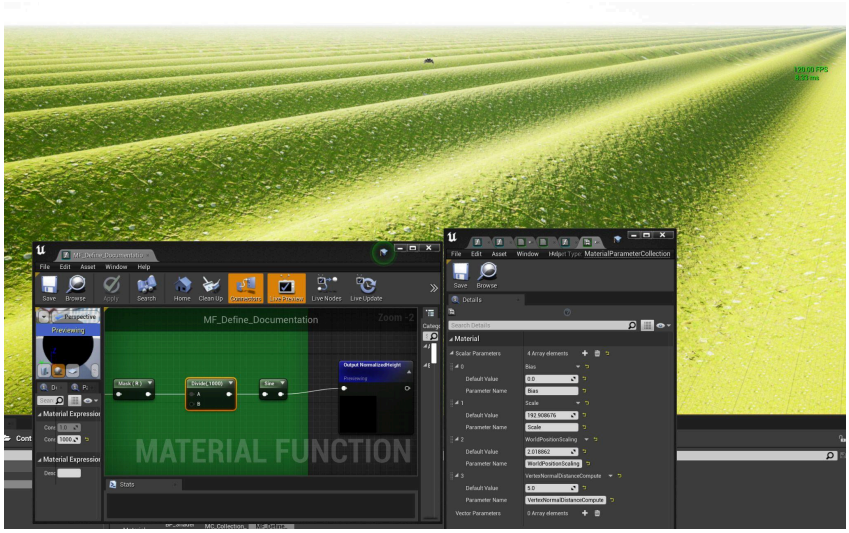
Look for Shader World in your **Place Actor** Unreal Engine editor tab,and drag it into the world, replacing both its **Material** and **Generator**. You're done !



Inside the various **MF_Define_*** from the demo map you can find the available premade nodes, noise, functions, from reading heightmaps, packed 16 bits heightmaps, noises, terracing, crater…

Good to know: **BP_ShaderWorld_*** should not move at runtime. If you move your world around, even in editor, don't forget to check **rebuild** in its "World Settings" tab, to guarantee the world is properly set up.

The LODs are defined by your vertical distance to the land/ocean: when collision is enabled we're using the collision mesh grids to accurately know the viewer altitude to the ground, otherwise we're using the altitude to the reference plane of the world.



III. Dynamic Blueprint Brush System

1. Presentation

ShaderWorlds can be further shaped and customized using a Blueprint brush system, via a **BP_BrushManager**, of class *AShaderWorldBrushManager*, placed in the same level and linked in the 'Brush Manager' tab.

Using the brush system, the ShaderWorld will send to the BrushManager its originally created heightmap, which in turn will apply the relevant layers and their related brushes for this given heightmap.

Brushes are Blueprint actors of class *AShaderWorldBrush* placed in the world as well, and linked to the **BP_BrushManager** from a layer. Brushes are defined by a material to be applied to the heightmap, and a world footprint on the XY plane, visible as a box, which allows to efficiently compute only the necessary brushes for a given terrain LOD. *Brush box dimensions are the responsibility of the user*, which can either use the scaling of the brush actor to increase its footprint, or adjust the dimension of the BoxComponent directly.

2. How To

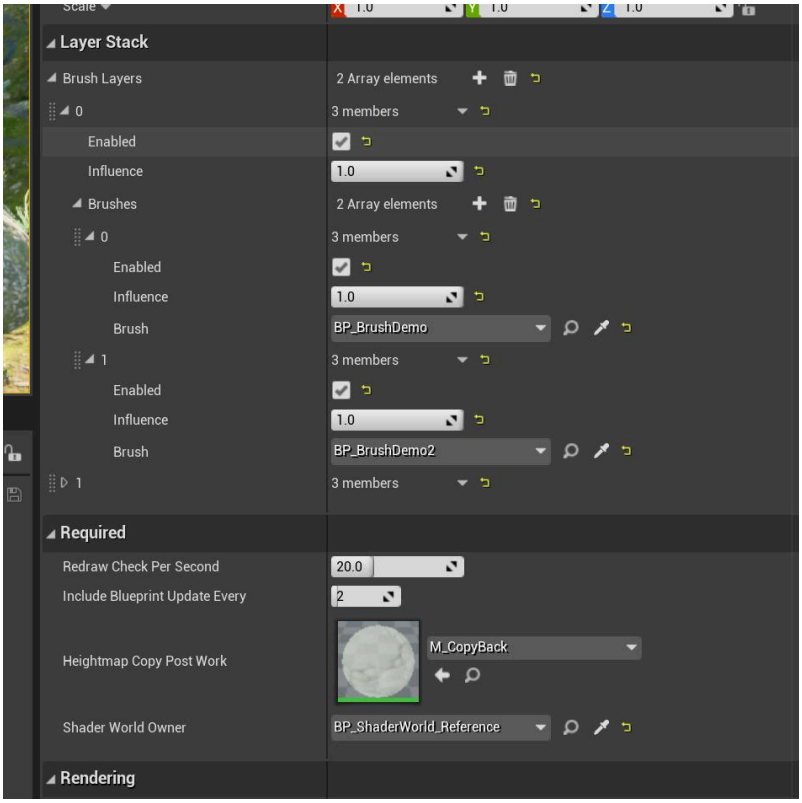
For a brush system demo, please refer to the map **Map_Reference** within the plugin content folder.

Brushes actors are basically managing a dynamic material which will be applied to the ShaderWorlds heightmap. From the brush blueprint you can access the dynamic material and therefore manage any shader parameter you might want to add to your brush.

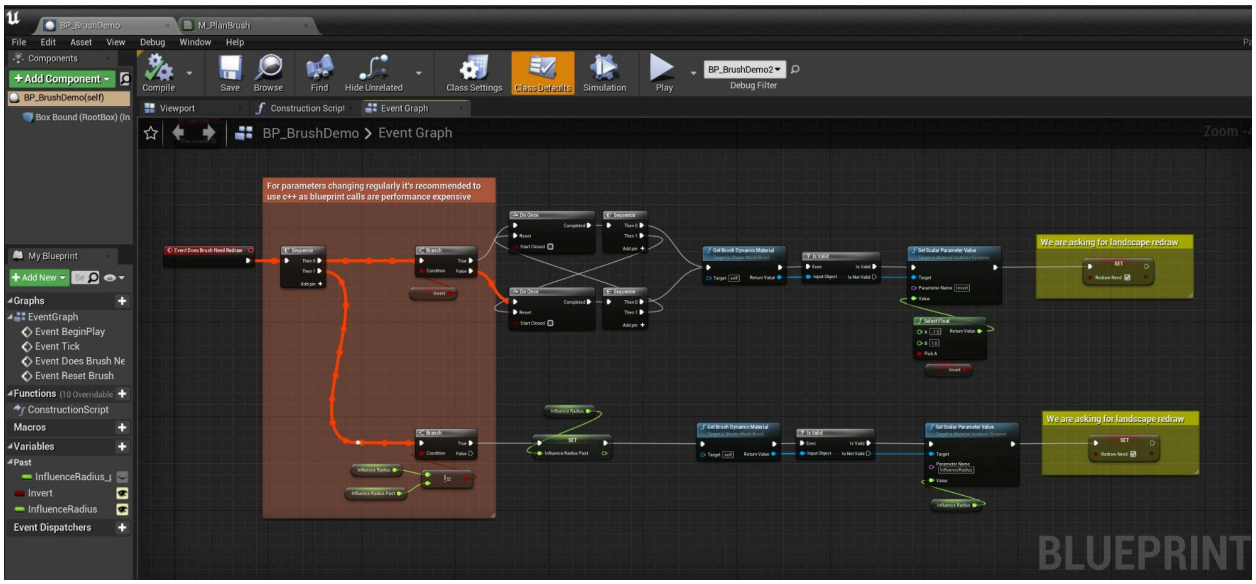
A blueprint function named **DoesBrushNeedRedraw** is called from c++ at a frequency defined in the **BP_BrushManager** 'required' tab. It's based on the update rate of the layer stack defined in **RedrawCheckPerSecond** and **IncludeBlueprintUpdateEvery**.

Blueprint workload being performance intensive, the blueprint implementable functions will only be called once every **IncludeBlueprintUpdateEvery** update of the layer stack. Preferably, Blueprints should rarely update parameters outside of the context of modeling the landscape in Editor. If a given brush has to update parameters regularly, it's recommended to create it using c++ by inheriting the *AShaderWorldBrush* class and overriding *ApplyBrushAt* and *Reset* functions.

Update of the layer stack allows it to check if any enabled brush has moved, as well as layers and brushes being toggled enabled/disabled, or influence changed, in which case it'll ask the ShaderWorlds to redraw the impacted LODs, update the appropriate collisions, and spawnables.



***BP_BrushManager** Layer Stack*



BP_BrushDemo *updating parameters of its dynamic material from blueprint*

Runtime changes of a Blueprint Brush Location

If spawnables locations aren't updated, they were most likely out of the Brush footprint bounds
Force a vegetation update by checking 'rebuild vegetation only' in Spawnable tab

IV. Smartphone/Tablet Android&iOS

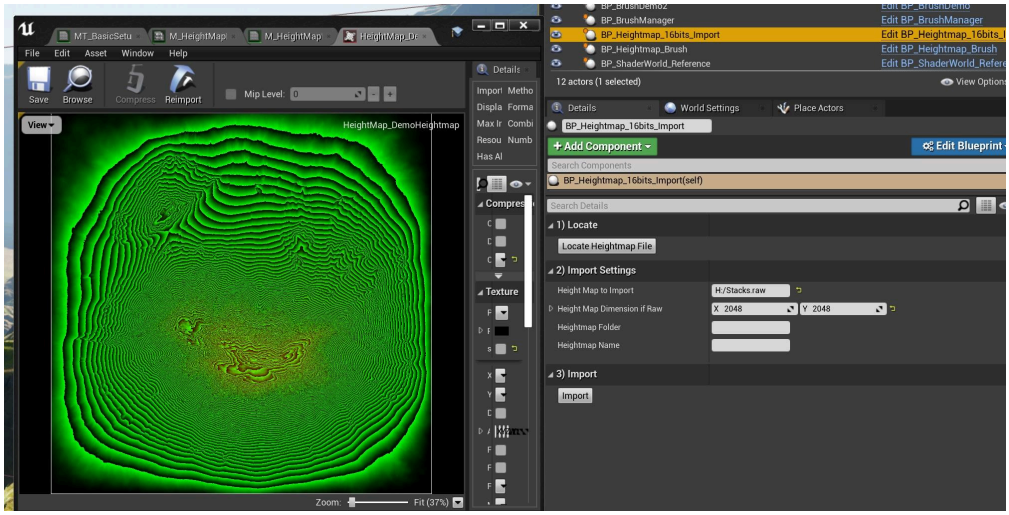
Shader World plugin is now fully compatible with smartphones and has been tested on both Android OpenGL and Vulkan. Note that **UE5** currently has a breaking bug for ShaderWorld on Mobile, we would suggest using the UE4 version while targeting mobile for now

The ground material used during mobile testing can be found in :
Demo_Reference/Material/SmartphoneTest

V. Heightmap Import (16 bits Raw/PNG)

A blueprint tool has been made available in ***BP/BP_Heightmap_16bits_Import***
Simply **Locate Heightmap File** you wish to import, currently supported formats are 16 bits RAW and PNG. If your file format is RAW then you need to specify the dimension of the texture as this format does not hold this information. If your file format is PNG the heightmap dimensions should be automatically defined.

Specify an output folder and a name for the heightmap and then press **Import.** The output file will split the 16 bits information between two channels. You can use ***BP/ToolBox/UVing/MF_BicubicPackedHeightmap*** to make a high quality bicubic read of this heightmap as featured in the Heightmap Brush ***BP/Brushes/Heightmap/M_HeightMapBrush***



V. Heightmap Brush

A new brush has been added to Shader World: the heightmap brush ! Youtube Teaser which showcases a new capability of the brush system: to write into the data layers of the terrain.

While the brush system was previously able to only move the terrain around given the shader deformation defined in the material of the brush (read a texture to make a road, flatten a terrain, add a noise, etc…) brushes are now able to write into the landscape data layers.

As a reference implementation you can check.. ***Maps/Map_Reference !***

The landscape **BP_ShaderWorld_Reference** has a dummy data layer named **HeightLayer** which doesn't do anything (we could derive the normal map to generate curvature information?!) while the brush **BP_Heightmap_Brush** has specified this precise layer as a

target in its Details panel *Layer Write* tab, as well as provided a material which will draw a *flow map* into this layer.

Examining ***BP/Brushes/Heightmap/M_HeightMapBrush_layer*** reveals that the flowmap is read as well as faded away at brush influence radius borders, and used to change the material of the terrain by editing the material ID stored in the Red Channel of the **HeightLayer** landscape layer. If Flowmap > 0.5 draw material 1, otherwise material 0.

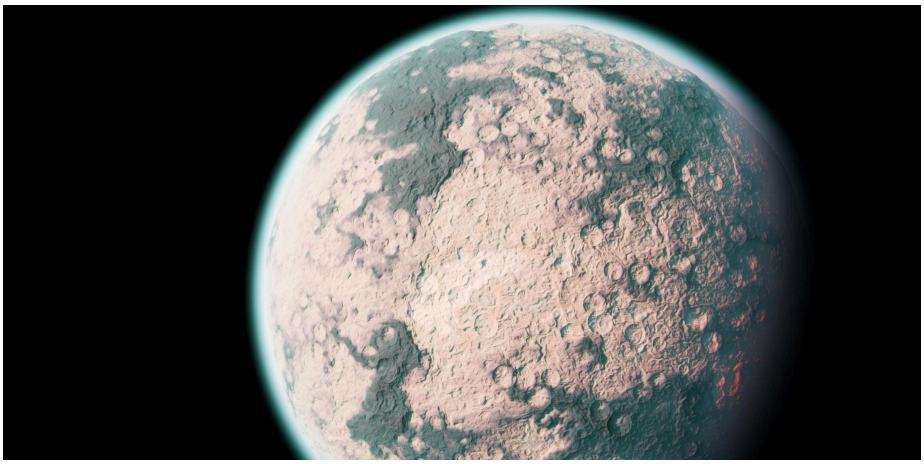
This layer is then read in the landscape material as any layer **BP_ShaderWorld_Reference**: **Cached Material,** and we use its value to change the index of the material we're reading from the ground material texture 2D arrays. (*GroundDiffuse/GroundORDisp/GroundNormal*).

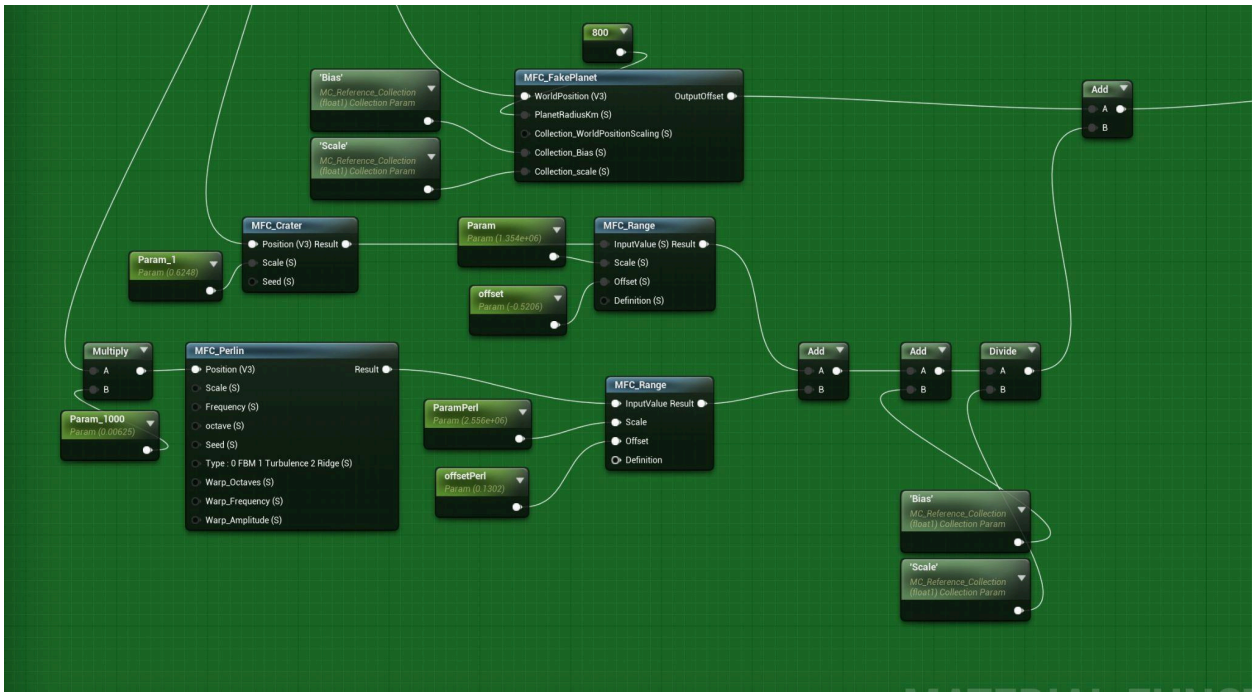
VI. Annex

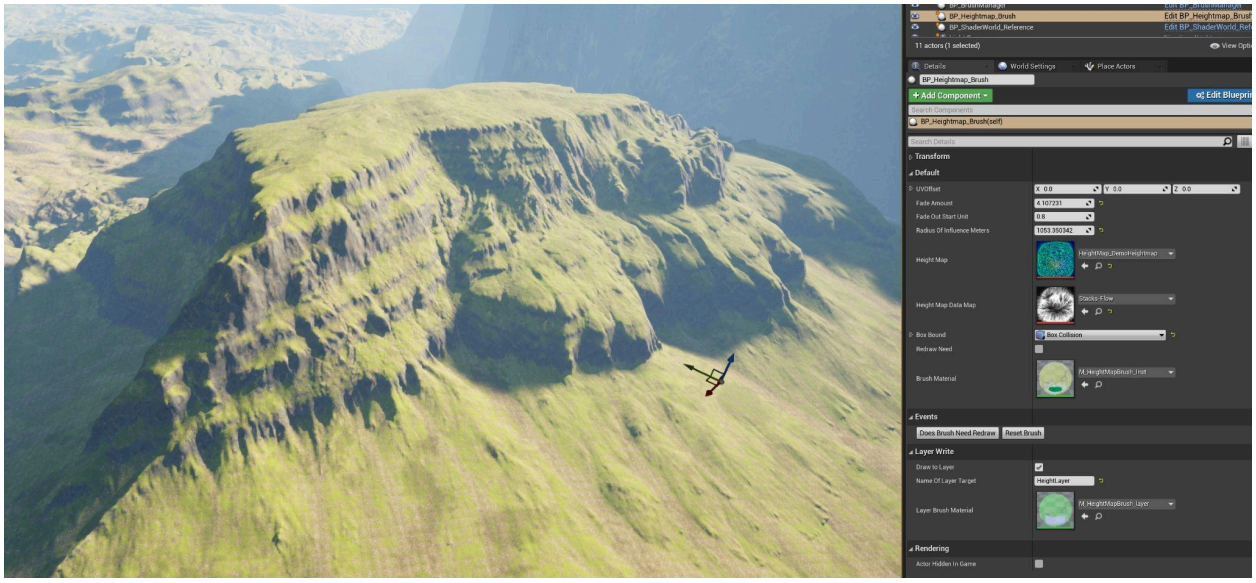
1.Fake Planet Node

Landscape plan projected on sphere, viable from space as long as you're around the north pole. Most of the area in the picture below is fully playable.

Check the demo_reference map for an example.







Known Issues:

- Spawned assets are currently refining their position as the camera gets closer to them using the runtime generated cache (normal map/height map), goal being to be able to spawn assets from very far away (based on cache of higher LODs/ low precision), but having them not flying above ground while getting closer to them. **Problem**: if a ground normal of higher precision suddenly meets the requirements for an asset to spawn, then an asset will pop in while there weren't any before. The other way around, a higher precision terrain data might make a previously viable spawning location unviable and an asset will disappear. Please use the **No Popping Range Meters** to prevent such asset popping within a certain range.

- World Origin Rebasing is currently supported for the landscape and its collision but not for the spawnables, note that this feature is UE4 specific as it is not required on UE5 thanks to 'large world coordinates support.