MicroSplat

MegaSplat Technique: 256 Textures

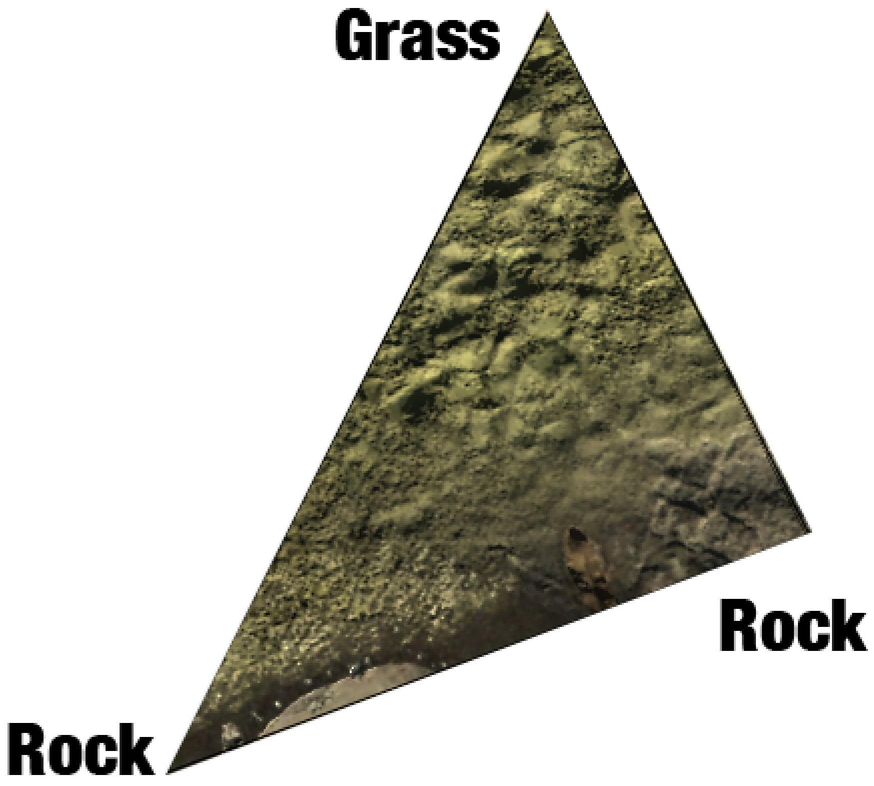
Overview

The 256 Texture module allows you to work with up to 256 unique textures on a terrain or mesh. It is based on a fundamentally different way of storing splat map data, and is called the MegaSplat Technique because the technique was first introduced in my older asset MegaSplat.

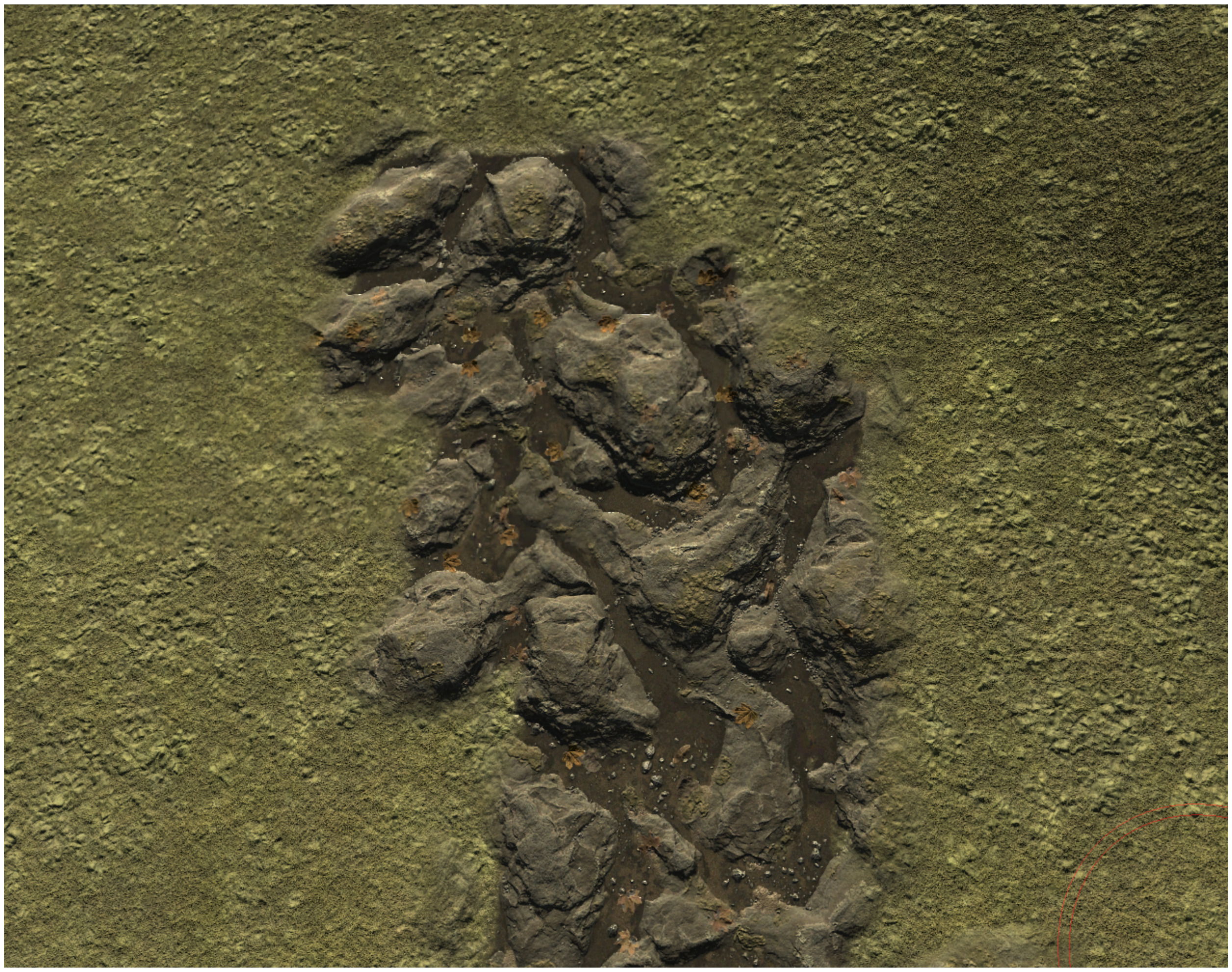
Understanding the Technique

Traditional splat maps store a weight per texture used - so if a terrain has 16 textures, it needs four RGBA splat maps which each store 4 weights for a total of 16 weights. This can become prohibitive with higher texture counts, as 256 textures would need 64 splat maps. MicroSplat makes a fundamental optimization of only ever sampling 4 of the final textures, but it still has to sample all of the control maps, and by default Unity terrain stores these as uncompressed textures on both the GPU and CPU, so they can consume a lot of memory as well.

Instead, the MegaSplat Technique stores the data in a fundamentally different way. Instead of storing a weight for every possible texture, it stores an index of which texture to use for a given vertex, samples these textures, and blends them across the face. So, as an example, a triangle might have a grass texture defined at one vertex, a rock rock texture on the other two vertices. At the center of the triangle, these textures would have equal weight, but at the vertex, a single texture would be 100% weighted.

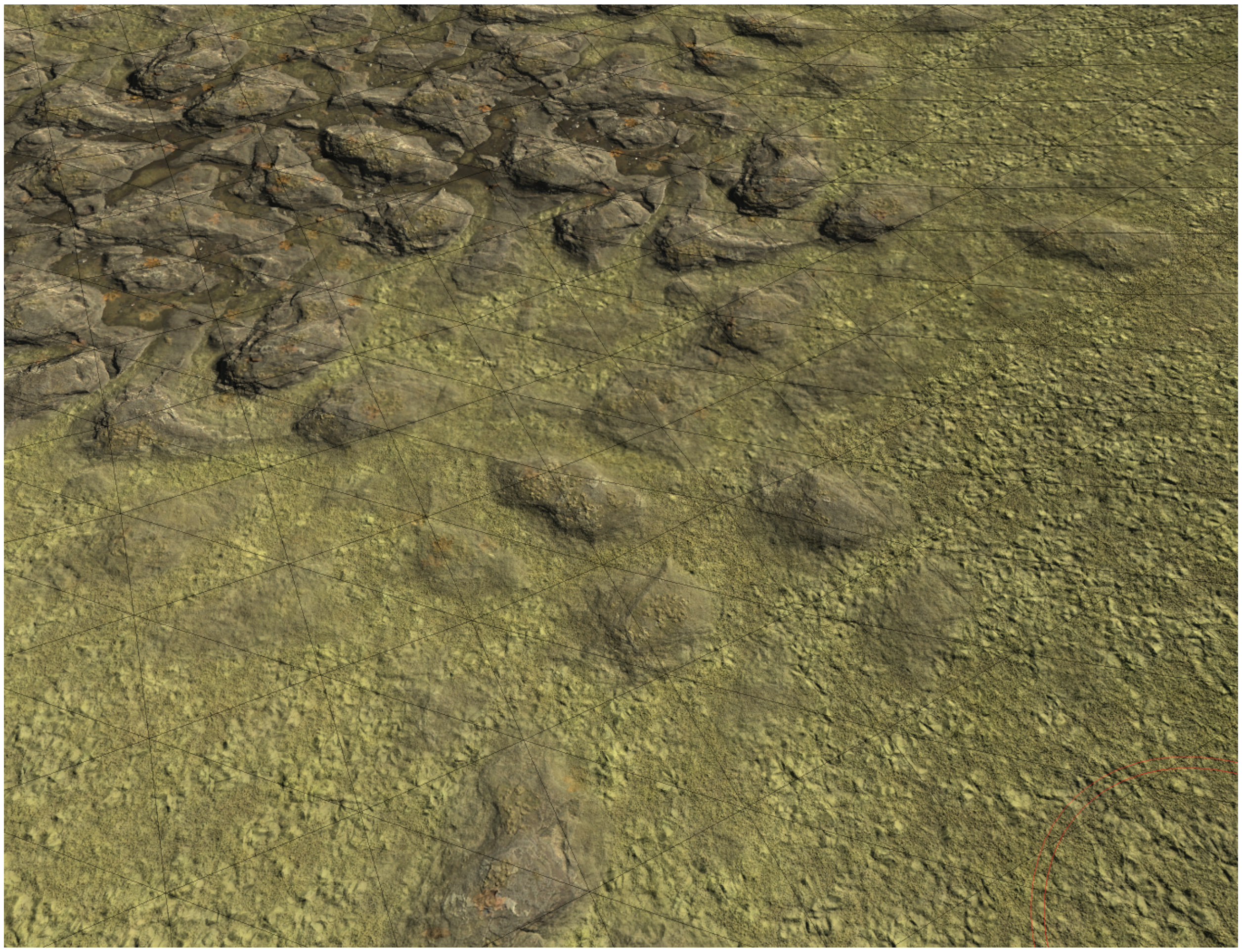


These two textures are being blended across the face, but at each vertex the texture is 100% rock or grass. This can create a harsh transition in some real world scenarios - for instance, with very small triangles, or even with well spaced triangles where you just want something softer:



The height map blending makes this transition look great, but often we want a much wider transition area, or just want the rock and grass to always be blended together for a nicer look.

To fix this, the technique allows for a second 'layer' with its own set of textures, and a single blend weight between these two layers.



Here we see an example where grass was painted on one layer, and rock on the other, allowing for a blend between the two surfaces that is not defined by the size of the triangles, but rather by the blend value painted from the brush.

Understanding how this technique works will make it much easier to understand the concepts in the tools, such as layers, and why you might get different results when painting than you expect.

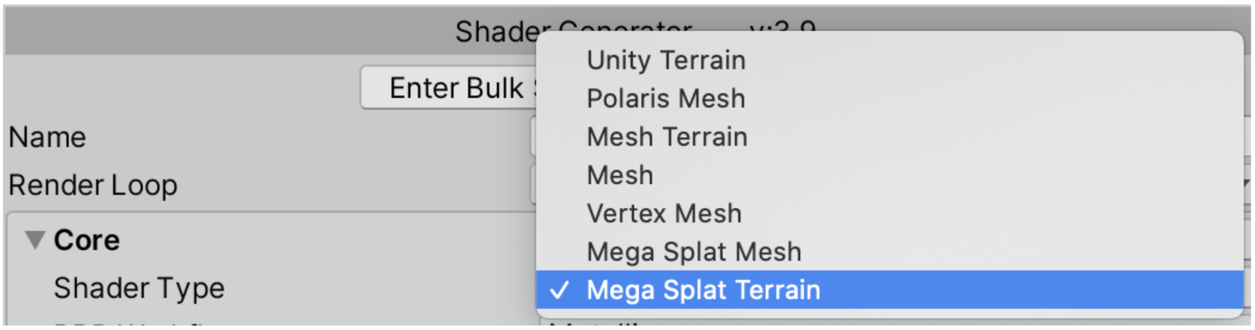
Data Formats

The Terrain format stores two weights and the blend value in a texture. This is stored as the first index divided by 255 in the red channel, the second index divided by 255 in the G channel, and the blend weight in blue.

When using the mesh version, each vertex can store 2 texture indexes, a blend weight between them, two layers of scatter, and 4 fx weights, which are mapped to wetness, puddles, streams and lava. This format is described below.

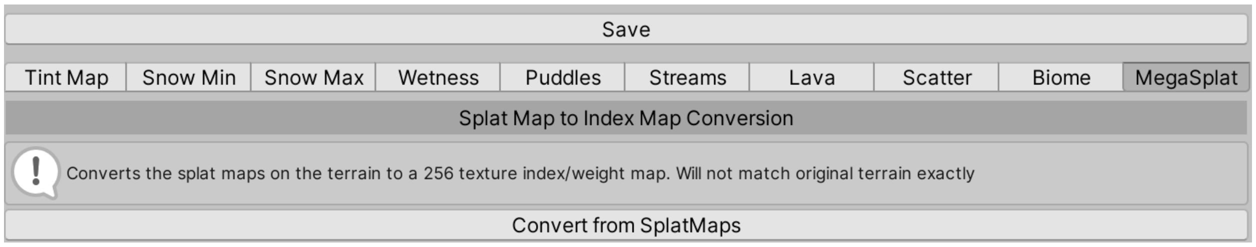
Getting Started : Terrain

First, convert your terrain to MicroSplat as described in the main documentation.



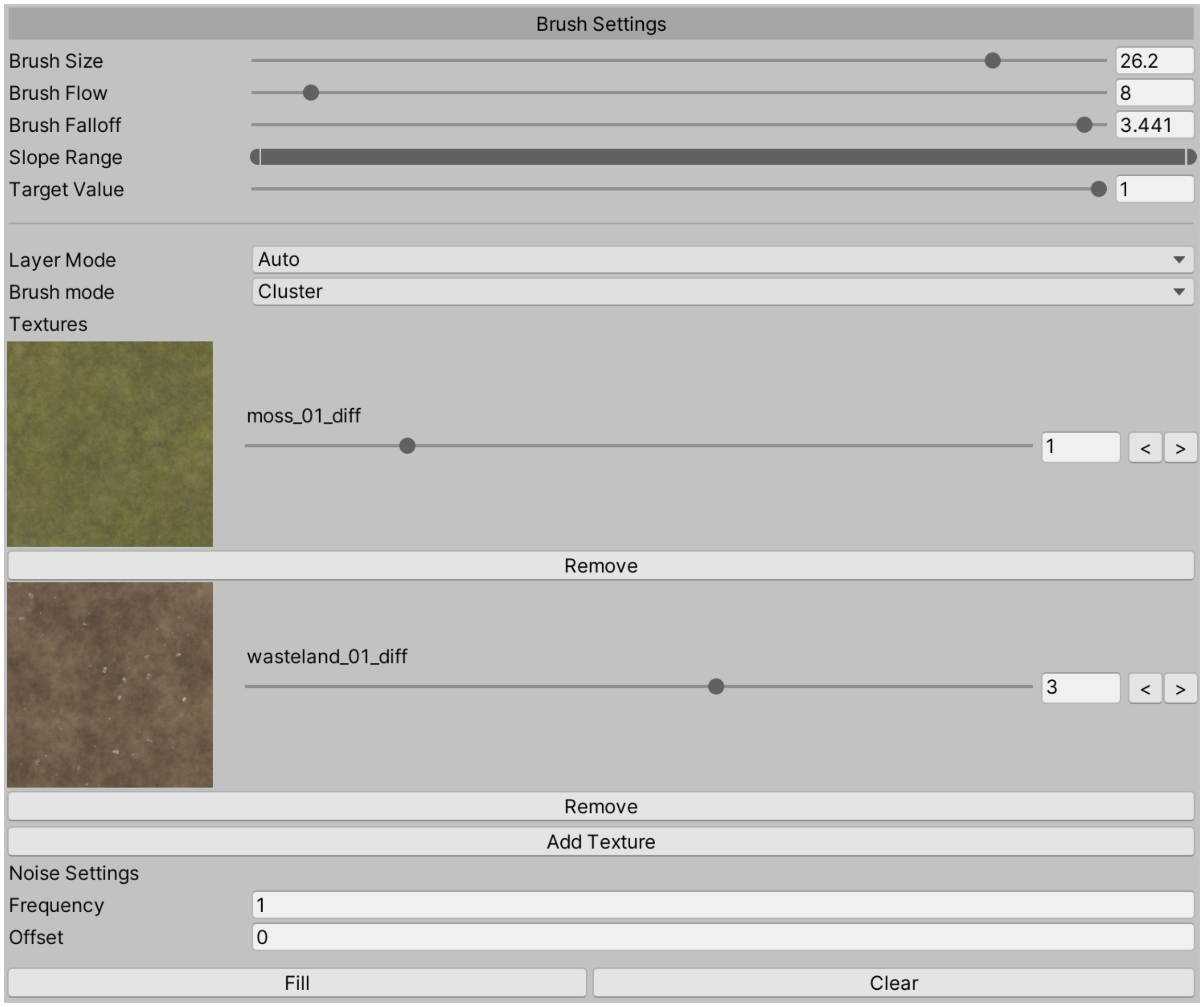
Once installed, two new Shader Type options are available on the material. Mega Splat Mesh, which is used for vertex based workflows, and Mega Splat Terrain, which is used for terrain based workflows.

To convert an existing terrain, change the shader type to Mega Splat Terrain, open the Terrain FX Painter from the windows/MicroSplat menu, select the terrain, and select the MegaSplat tab. All texturing of the terrain in MegaSplat mode is done with the Terrain FX Painter, not with Unity's tools.



Converts the splat maps on the terrain to a 256 texture index/weight map. Will not match original terrain exactly

Please notice the giant Save button at the top to save changes. Just below that is a button that will convert the current splat maps on the terrain into the index map that the technique uses. Note that because the formats are very different, it will not always be able to perfectly convert an existing splat map exactly, especially in areas where the existing splat map blends many textures in one spot.



Brush Settings

Brush Size	26.2
Brush Flow	8
Brush Falloff	3.441
Slope Range	
Target Value	1
Layer Mode	Auto
Brush mode	Cluster

Textures

moss_01_diff	1
wasteland_01_diff	3

Noise Settings

Frequency	1
Offset	0

Below that button is the brushing interface. The top section is the same as used in the other painting modes and described in the core docs, allowing you to set the brush size, falloff, capacity, etc. Below it you will find MegaSplat specific controls

Layer Mode

The megasplat technique is a ‘layered technique’. You can paint on the top or bottom layer, or use the Auto mode, which always paints new textures on whichever layer has the lowest weight.

Brush Mode

The brush mode can be Single, Clustered, or Blend. In Single mode, a single texture is painted. In Cluster mode, you can add as many textures as you want, and they will be chosen based on a noise function (you can adjust the frequency and offset of this noise). Finally, the blend brush will always ramp the blend weight between the layers towards the target value specified. If set to 0.5, this will create an even blend between the two texture layers.

Getting Started : Mesh

Mesh Conversion

For the MegaSplat technique to work, the shader needs to access the barycentric coordinates of a triangle. However, these are not available in most shader pipelines, even though the GPU computes them internally. To get around this, each triangle is stamped with exactly one red, one blue, and one green vertex.

You can convert static meshes to this format with the included tool, under Windows/MicroSplat/MegaSplat Technique Mesh Converter. You can drag a mesh prefab from your project folder into the GameObject field, press convert, and it will write a copy of the mesh with the data burned into it next to the prefab with the _splat extension. You can then replace your Mesh Renderer’s mesh with this one.

If you are building a voxel world, or generating meshes dynamically, you will have to burn this data into your meshes at creation time, otherwise texturing will not look correct.

Data Format

The data for texturing is packed entirely into the color channels of the vertices. Each component holds 4 8 bit values, giving us 16 total values in just the color data. The format is as follows:

Color.r contains the barycentric color (red, green or blue)

Color.g contains the first texture index, second texture index, and blend weight between the textures

Color.b contains the first scatter index and weight, and the second scatter index

Color.a contains the second scatter weight and the first two effects channels (wetness, puddles)

Texcoord0.z contains the stream and lava fx weights

Texcoord0.w contains the snow mask min and max values

The Example Scene

If you open the "MegaSplat_vertex" example scene and press play, a plane will randomly re-texture itself every few moments. The TestMegaSplatTechnique.cs script which is on the plane will show you how to set the data on the vertices. Note that currently there are no tools for manually painting this data.

Texturing Tips

This texturing format offers some advantages and disadvantages over the standard weight based techniques, like the one Unity terrains use. Let's consider what happens on a single layer of texturing across a triangle. One vertex might say it uses the grass texture, while another uses a rock texture, and a third uses sand. When texturing, all three textures will be sampled, and blended such that the given texture has the total weight at its vertex.

In practical terms, what this means is that at a vertex, only a single texture can be seen. If you have very small triangles, it means your texturing will rapidly transition between choices, and not have much area to blend.

However, the shader provides support for 2 texture indexes and a blend weight - essentially 2 “layers” of this technique. This means that if you want to slowly blend in a texture over a wide area, you will want that texture to be written on a separate layer, and use the blend weight to slowly blend between them. So instead of a vertex just being “Rock”, it’s now “Rock at 75%, grass at 25%”.

One nice benefit of this technique is free texture clustering. MicroSplat offers several forms of texture clustering already, but these are done by taking extra samples of additional or the same textures. When using the MegaSplat Technique, there’s nothing stopping you from having several variations of a rock texture, and simply selecting different variations on each vertex based on some noise or hash function, randomly varying the surfaces over the space. This is exactly what the Cluster Brush does in the Terrain painter. This is basically a much cheaper form of texture clustering, as it doesn’t increase the number of texture samples needed, only means sampling a wider range of textures per pixel, which can slightly reduce cache performance.

Optimization

So if you fully understand the technique, you’d realize that each layer of texturing takes 3 sets of samples. So a two layer shader would take 6 sets of samples. However, MicroSplat was written to support only 4 sets of samples per pixel, and rather than expand that, I sort the weights and cull the two lowest weighted ones, renormalizing the weights. In theory this might cause rendering artifacts, but in practice it’s very unlikely to ever have more than 4 textures visible on a single pixel, and I have not been able to create an artificial case where this is a problem.