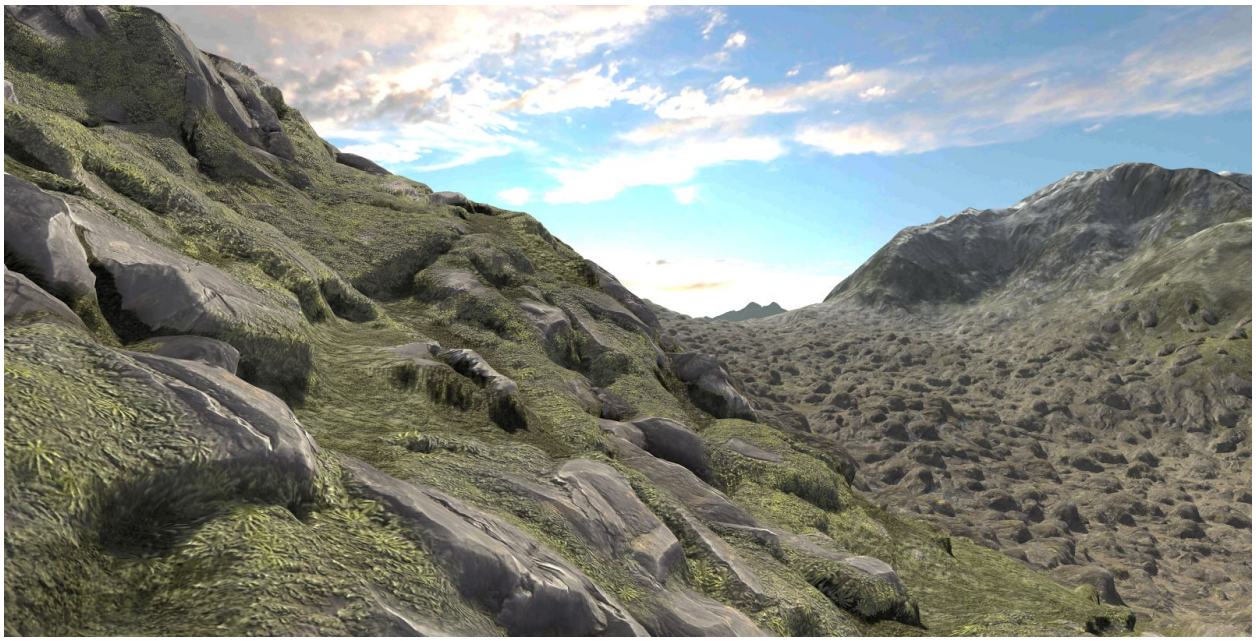
MegaSplat

Documentation, version 1.5



Thank you for purchasing MegaSplat. This document attempts to thoroughly explain the features of MegaSplat. However, sometimes seeing these features in action is a much easier way to learn than written documentation, so I suggest watching the video tutorials on YouTube.

Latest Documentation (might be for a newer version):

Forum Thread

YouTube Channel, with over 20 development logs and quickstart tutorials

Official Website:
http://www.megasplat.com

The forum thread is the best place to get support.

Email gets filtered by spam systems, posting on the asset store review pages does not notify the author of the asset, does not allow the author of the asset to contact the person posting the review, and does not notify you if a response is posted.

Known Issues:

- Unity broke Texture Arrays from Unity version 5.6 to 5.6.1p2. If you are running 5.6, please make sure it is a version greater than 5.6.1p2.
- There are complex issues with using the Metal API on OSX, please see the bottom of this document for more information on these issues.
- There is a shadow issue when using certain Enlighten modes with Tessellated shaders in Unity 5.6 and up. Unity changed something about this workflow with the introduction of the progressive lightmapper, doesn't document the stuff, and hasn't responded to my support requests with any information about the changes. I will continue to look into the issue.

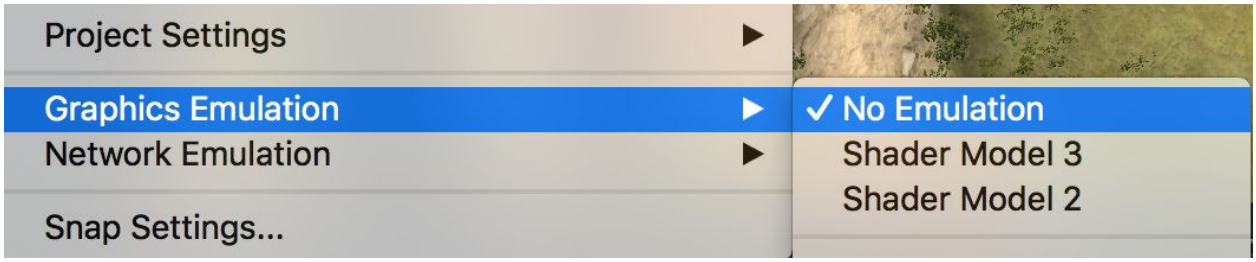
Overview

MegaSplat is an advanced Splat Mapping system for Unity which allows up to 256 textures to be painted on Meshes or Terrains, using a single material and rendered in a single pass with height map based blending. Traditional splat mapping techniques are limited for four or five textures per pass, with the cost of the effect being proportional to how many textures you use. MegaSplat has a consistent per-pixel shader cost regardless of how many textures are used. MegaSplat is based off of Unity's standard shader Metallic workflow, which means that it supports all of the lighting features that Unity supports.

MegaSplat works with Meshes, or Unity Terrains. A shader and toolset is available for both workflows.

Requirements

MegaSplat requires Texture Array support, which was added in Unity 5.4. Texture Arrays are available on most platforms with the exception of DX9 and openGLES2.0. If you see a black terrain when you open up the example scene, it is likely that this is because your Unity is running in emulation mode, or running in DX9. To fix this, check that you are set to "No Emulation" in the Edit>Graphics Emulation settings:



Also make sure your editor is running in DX9 or OpenGL/Metal mode. The current API being used is listed at the top of the Unity window:

Sometimes Unity will revert back to using Emulation, or startup in DX9 or DX11 under DX9 mode when an older integrated graphics card is used. When this happens, the terrain is rendered as black.

Note that when building for OSX, Unity version 5.6 and greater default to Metal in builds, but use OpenGL in the editor. Please see the section at the end of this document for more information about Metal vs. OpenGL rendering and MegaSplat.

Quick Start

The best way to understand how MegaSplat works is to start playing with the examples included. There are **TWO** workflows in MegaSplat, one for working with Meshes (using the Vertex Painter), and one for working with Unity Terrains (using the Terrain Painter). The following tutorial is using the Vertex toolset, so make sure you are opening the correct scenes and toolset.

- Open up the scene in MegaSplat/Examples/Scenes/**Mesh.unity** or MegaSplat/Examples/Scenes/**MeshTessellation.unity**
- Open up the vertex painter under **Window/Vertex Painter Pro** and dock it somewhere in your UI
- Select the meshes GameObject. By selecting this root game object, you can paint on all the game objects parented to it at once.
- Make sure the Vertex Painter's active property is true
- Go to the **Custom tab**
- Click on the "Brush" object selector property and select "MegaSplat_Example" as your brush
- Select a texture from the menu, and begin to paint

Note that all painting with the MegaSplat shaders should be done in the "Custom" tab or "Flow" tab if your painting flow directions in the vertex painter. Painting on the vertex colors or UV channels may have undesirable effects, as this data is used by the shaders.

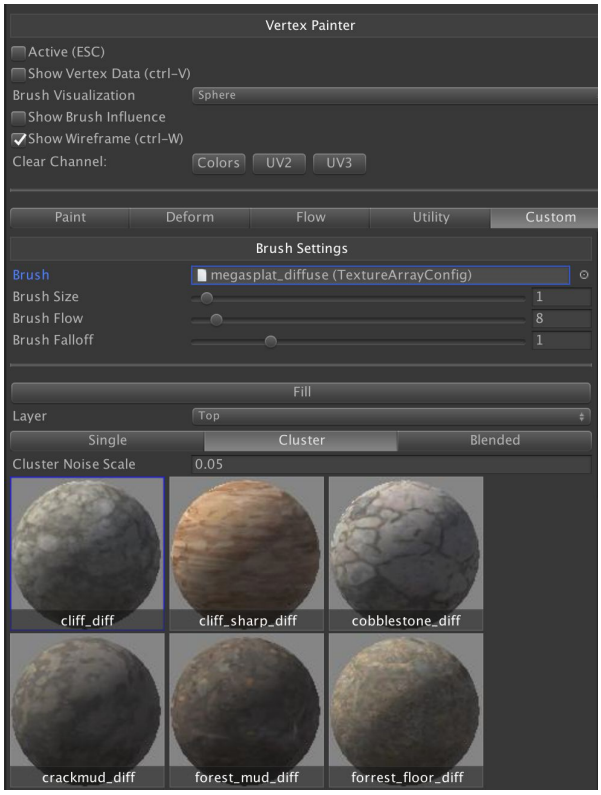
Vertex Painting Window

The first section of this window allows you to turn the vertex painter on or off, show the actual data painted onto the vertices using a preview shader, change the brush visualization, and toggle if the wireframe of the selected meshes should be shown.

You can clear any edits to any vertex channels by clicking on the button in the “Clear Channel” strip. All editing in the vertex painter is non-destructive and undo-able.

The next area holds various brush settings, such as size, flow, and falloff. The vertex painter also supports tablet pressure. You can also fill all selected meshes with the fill button.

Finally, you can change the brush mode and select brushes to paint with.



The brush mode allows you to paint using different layers and smart-brushes. When using a two layer shader, you can choose to paint on the top or bottom layer. You can paint single textures at a time, texture Clusters, which select from a set of textures based on noise functions, angle or height values, or with a Blended brush.

The Blended brush option paints on both layers at once, allowing you to select a top and bottom texture, and use noise to blend between them. These can be used to create very complex brushes.

The strategy I tend to use when painting is to fill the bottom layer with one set of textures, then switch to the top layer and blend in another set of textures, then switch back to the bottom layer to blend in a third set. Although only one texture may exist at each vertex on each layer, working back and forth like this allows me to build up a rich and complex surface in a natural manner.

Once you are finished playing with the example scene, the rest of this documentation will help you setup your own meshes and textures for use with the system, and understand all the tools and features of MegaSplat.

Working with your own data

Process your meshes

If you are working with Meshes, MegaSplat requires specific data on each vertex in order to work. This process may need to split some vertices resulting in a slightly larger vertex count, depending on the topology of your mesh. To convert a mesh, open the converter at Window->MegaSplat Mesh Converter and drag a game object from the project window into the “Game Object” field, and any meshes contained in that asset will be converted. Your mesh will need to be set to **read/write** in the importer settings so that the mesh data is available for processing. Press process, and a new asset will be created next to your original asset with “_splat” added onto the name.

Note that if you plan to use static mesh optimizations, or turn off read-write, you will want to bake your meshes down when you are done painting them. This is discussed later in this document.

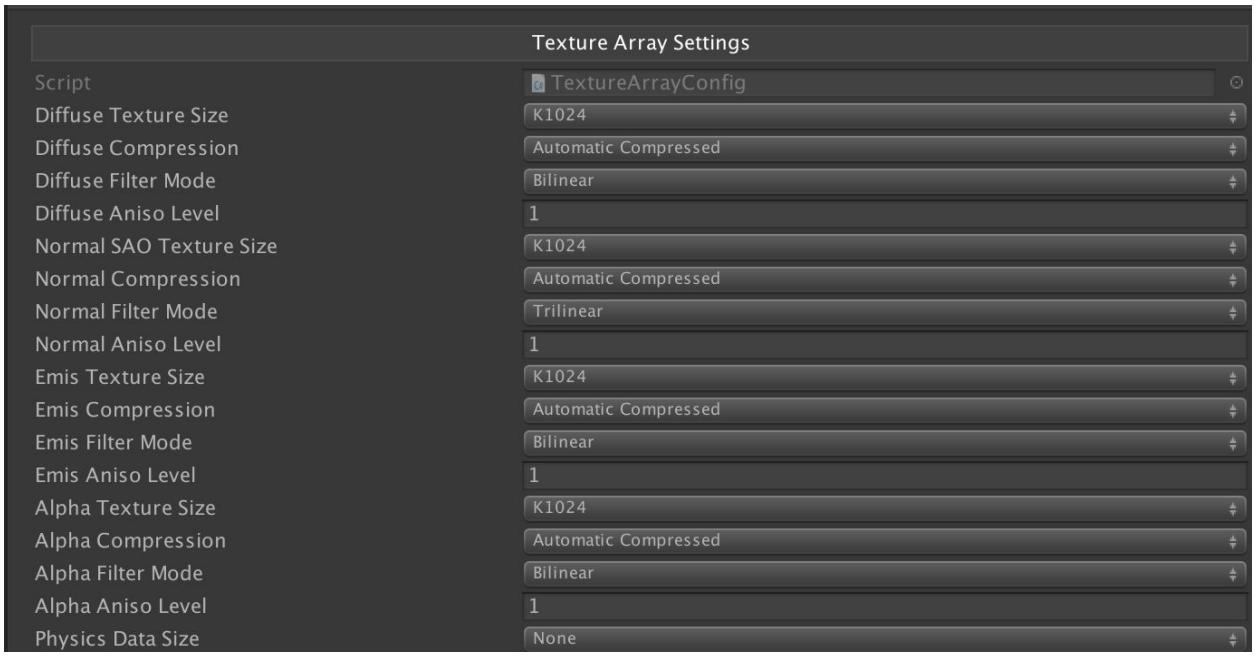
Creating Texture Arrays

MegaSplat is based on Texture Arrays, which are a new texturing feature introduced in 5.4. Unity does not provide tools to manage texture arrays, so MegaSplat provides them for you. A texture array is basically a long list of textures which act as a single resource for the GPU. The texture arrays in MegaSplat are managed by a TextureArrayConfig object, which will automatically pack and convert source textures into multiple texture arrays.

Creating a Texture Array Config

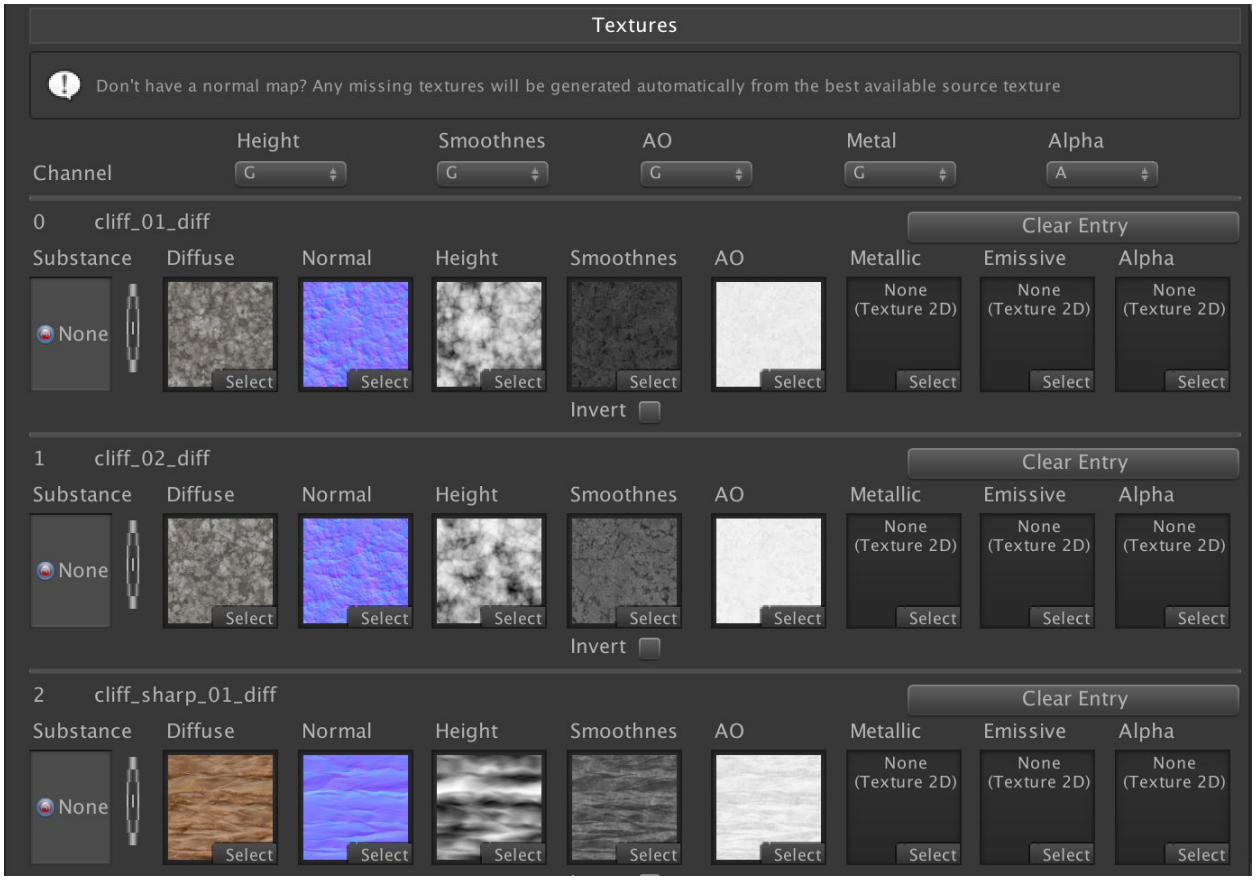
- Right click in the project view inside that folder and go to “Create->MegaSplat->Texture Array Config”
- Select the created config and give it a name.

Once you create a Texture Array Config, you will manage all textures for your materials from this interface.



The settings are controls the final size and settings of the texture's created. You can use textures of any size for input, but GPU's require that all textures in a texture array are sized the same, so the texture packer will convert your texture to the size listed here.

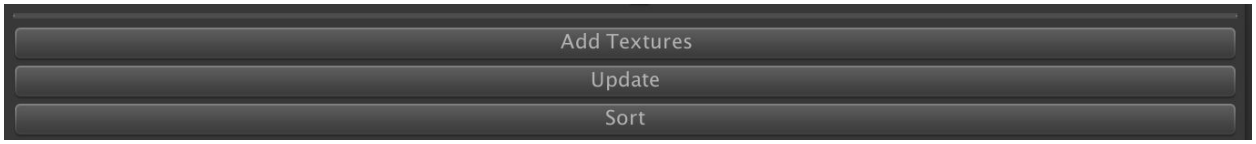
The batch importer allows you to import entire directories of textures at once. You can set the extension name for each texture, and all textures in the chosen directory with these extensions will be automatically put into the config.



The texture list is where you can manage the source data for every texture. For each entry, you can supply a substance, or a list of source textures for each component of a PBR surface. If you are missing one of these textures, the texture packer will create the missing data from the maps you provide (generate a normal map from your height map, etc) as it packs the data into the arrays.

The diffuse, normal, height, smoothness, and ao textures are packed into two texture arrays. If a metallic or emissive map is used, another array will be generated to hold this data. If an alpha texture is assigned, a separate alpha array will also be generated.

Once generated, you can assign these arrays to your materials in the same way you would assign a regular texture.



Pressing Update will compile your array. This may take a few moments, as it has to pack the textures together and generate any missing textures it needs.

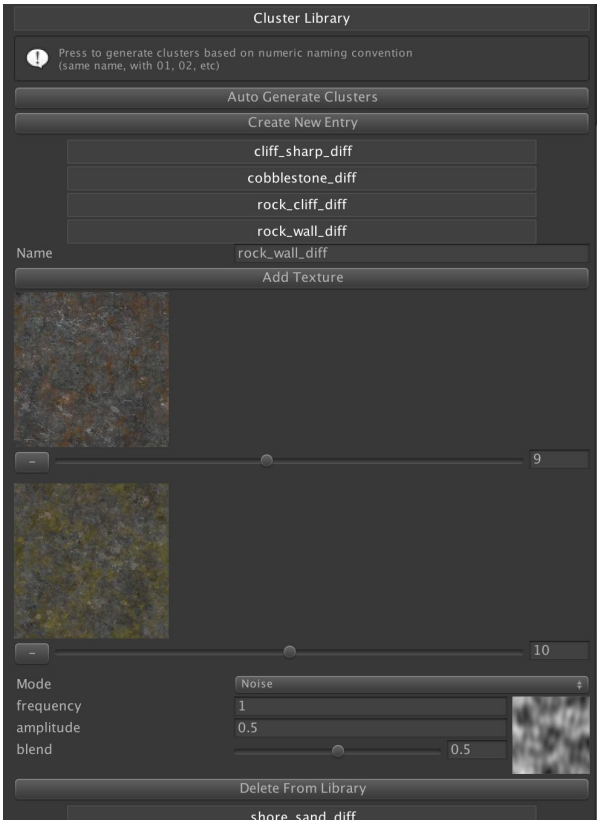
There is also an option to sort your textures by name.

Texture Cluster Library (optional)

Every Texture Array Config also holds a Texture Cluster Library. A texture cluster is a group of textures which should be treated as one when painting. As an example, you might have three variations of a mud and stone texture and want to paint them down in random variations. This can be a huge visual win, as it will break up any tiling. Using this technique, you could favor smaller textures with lots of variation, which can look a lot better than one high resolution texture.

MegaSplat can automatically generate these clusters for you based on naming conventions. For instance, if you name your textures mud_01, mud_02, mud_03 and press “Auto Generate Clusters” in the Texture Array Config editor, it will create a clusters of these textures automatically. You can also add them manually with the “Create New Entry” button.

If you open one of the generated clusters, you will see a list of all the textures it uses and the blending system it uses below them.



You can autogenerate, or create new clusters of textures with the cluster library editor.

When painting, which texture is chosen will be based on one of three functions:

Noise

Use a simplex noise function to randomize the texture painted on each vertex.

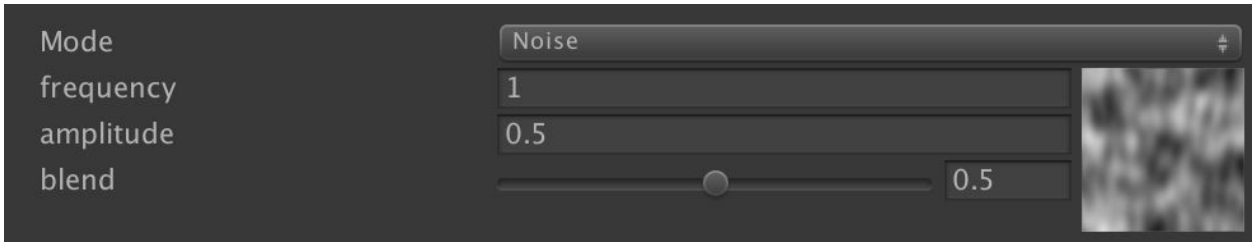
Angle

Use the angle of the vertex normal to decide which texture is painted on each vertex.

Height

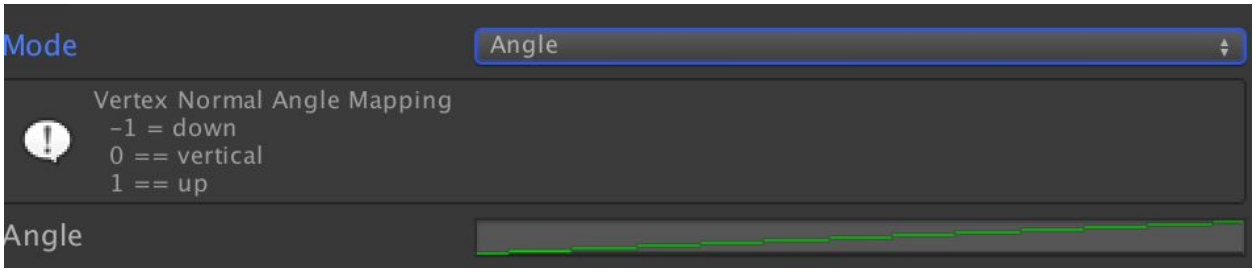
` Use the local height of the vertex

Noise Editor



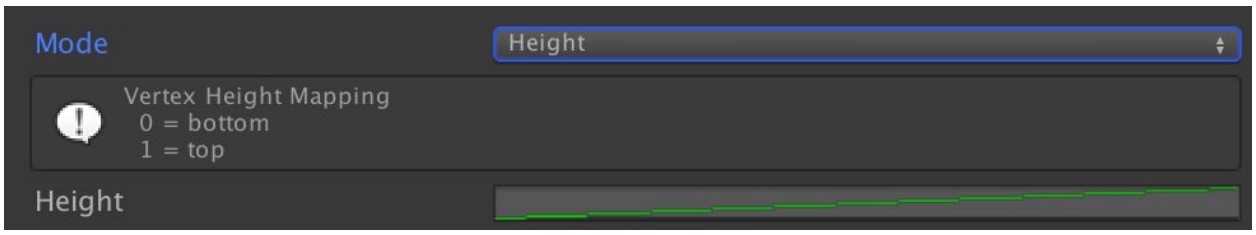
In this mode, a noise function is used to determine which texture goes on which vertex. The noise function exposes a frequency and amplitude allowing you to adjust how often and varied the choices are, along with a blend value, which determines where the center point of the blend is. A small preview of the noise function's amplitude is shown to the right; if the noise function's output is closer to black, it chooses textures in the beginning of the cluster, if it's closer to white, it will choose textures towards the end of the cluster.

Angle



The angle option allows you to choose textures based on the angle of the vertex normal. This is expressed as an animation curve mapping a -1 (down) to 1 (up) angle of the vertex normal to a 0 to 1 value representing the position in the cluster. If we had three textures in our cluster and painted across a surface using the default curve, the first texture in the array would be applied to vertices which are pointing mostly down, the second texture would be applied to vertex normals pointing mostly sideways, and the third would be applied to vertex normals pointing mostly up.

Height



` The height option is similar to the angle option, except that it maps a 0-1 height value instead of a vertex normal angle. In this mode, textures earlier in the cluster will be applied to

the bottom of the object, and textures later in the top. You can use the curve to adjust this mapping.

Setup a Terrain

To get started with a Unity Terrain in MegaSplat, simply select the terrain and open the terrain painter. The terrain painter window will notice that your terrain is not in MegaSplat format and ask you if you'd like to convert it. When you convert the terrain, it will generate a material and shader, create the backing textures needed to store the splat data, and assign these to the terrain. Once this is done, you need to setup the material and shader. You can press the "Select Material" button in the Terrain Painter, set options on the shader, and assign the texture arrays. When you return to the Terrain Painter with the terrain selected, select the texture array config from the menu and the painting tools will appear.

MegaSplat also contains a system for synchronizing the material and shader settings across many terrains. If you select multiple terrains at once, it will add a component to each terrain that contains a reference to a template material. Changes to the template material will automatically be propagated to all the terrains which use it.

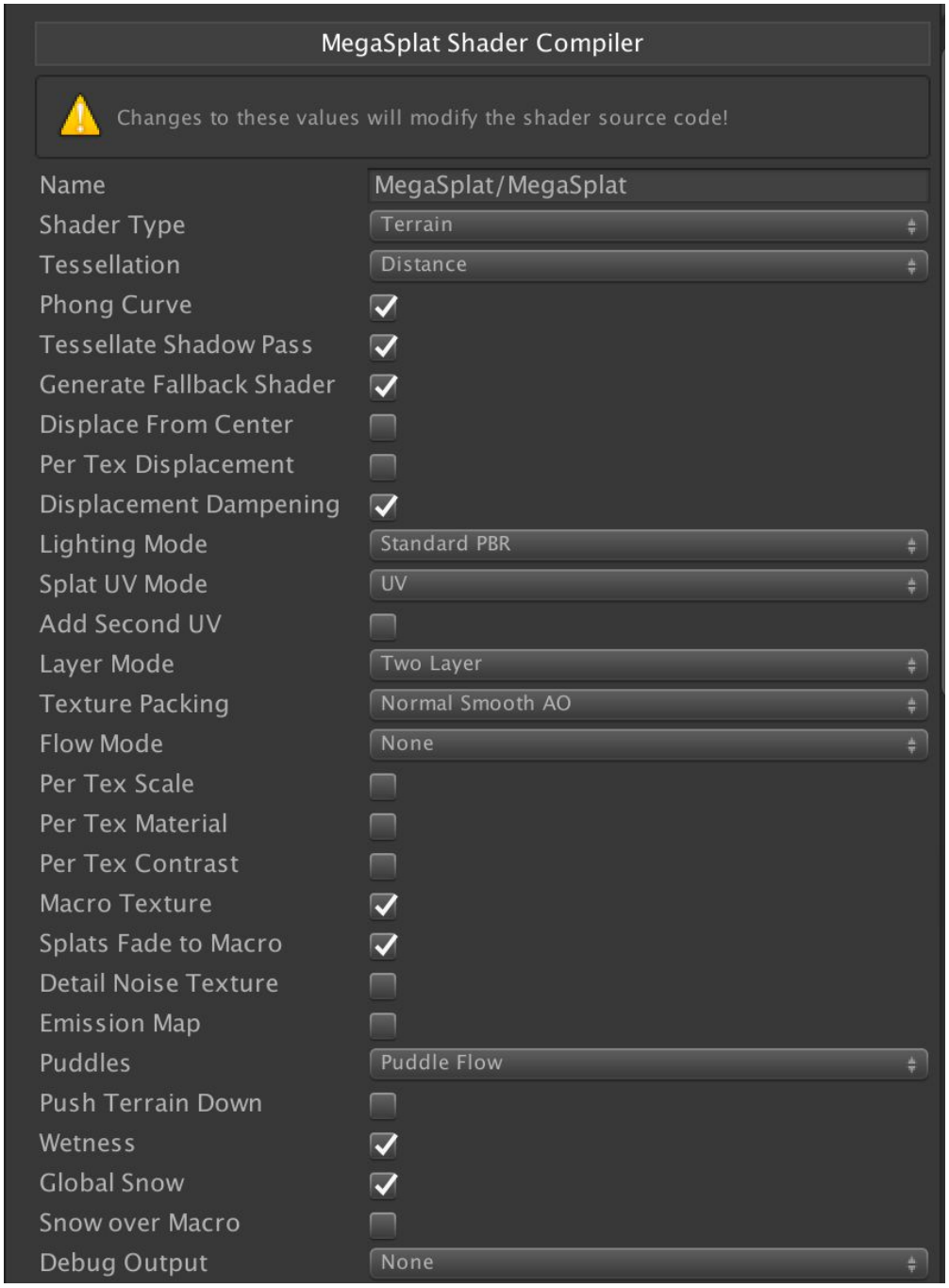
If your terrain was previously textured with another program, you can convert that texturing to MegaSplat format using the terrain converter, described below.

The Shader Generator

MegaSplat grew too large to use Unity's built in shader_feature and multi_compile options for it's shaders. It now features a shader generator, which writes out new versions of the shaders as you change settings on them. This allows MegaSplat to have far more options than a traditional shader, increases my ability to optimize the system, and avoids having dozens of shaders to choose from with arbitrarily chosen feature combinations. Rather, almost all features can be mixed and matched into the exact shader you need.

To create a new MegaSplat shader, right click in your project view and select Create->MegaSplat->MegaSplat Shader.

Next create a material and assign the shader which was created. Select your material and the first section of the interface is the Shader Compiler:



Note the warning on the top- when you change one of these values, the shader this material uses will be regenerated with the new options, changing the possible options and traditional material parameters below it. This means that if multiple materials use the same shader, you will be changing those options on all of them as well. In most cases, you will only need a few unique shaders per scene. You will want to set the shader type to terrain or mesh, depending on which workflow you are using.

Options:

- Name
  - This is the path for the material within the shader selection
- Shader Type
  - Mesh or Terrain
- Alpha Mode
  - Alpha or Alpha test modes are available. Exposes a property for an alpha array. See Alpha Mode for more information below.
- Tessellation
  - None, Edge, or Distance based Tessellation is available.
  - Phong Curve. When enabled, uses Phong Tessellation to smooth the overall mesh.
  - Tessellate Shadow Pass
    - Should we run the tessellation on the shadow pass?
  - Generate Fallback Shader
    - Causes a second fallback shader to be generated for devices which don't support tessellation.
  - Displace From Center
    - When false, displaced vertices move only up along the normal from the original surface. When true, displacement is centered around the original vertex, moving both up and down.
- Per Texture Tessellation Params
  - Allows you to control the amount and Up Bias per texture
- Splat UVMode
  - Controls the UV selection for the mesh.
    - UV takes the UV coordinate from the mesh's first set of UVs
    - UV2 takes the UV coordinates from the mesh's second UV set
    - UV Project uses a planar projection to generate UVs
      - You can choose local or world space
      - Top, Front, or Side projections are available
      - You can choose to use projected tangents, or the models tangents. Note that a model may only have one set of tangents, so your normal map alignment can only be based on a projection, or based on the model, not both.
    - Triplanar uses triplanar UVs, which blend textures projected from top, front, and side local space UV projections. Triplanar can be expensive, because 3x the number of samples are needed.
- Add Second UV
  - This allows you to have a different UV set for the macro texture. By default, the macro texture takes it's UV's from the first UV set on the mesh. However, by adding a second UV set, you can use a projection for the macro texture, a second set of UVs, or even use triplanar texturing for splat maps and the UV coordinates for your macro texture.

    - Turning on this option raises the minimum shader model to 4.0 due to the number of texture interpolators needed to pass an additional set of UVs.
- Lighting mode
    - Current options are StandardPBR and Ramp. Under Ramp lighting, a 2d texture is used to represent the diffuse lighting response. This is useful for non-photorealistic rendering, and removes the need for expensive lighting calculations.
- Layer Mode
    - Single Layer: One layer of MegaSplat data per control point
    - Two Layer: Two layers with a blending factor between them
    - Detail Texture: One layer of MegaSplat, but with detail texture sampled for the second layer
    - Alpha Layer: Allows for the second layer of splat mapping to be blended over the macro texture (the first layer acts as an erase function). Useful when you want to blend splat mapping over traditionally texture mapped objects, such as blending sand to the base of a building, or putting dirt on a roof.
- Bottom Layer Texturing
    - Painting: The default mode, allows you to paint on this layer to choose which texture goes where
    - Project 3 way: Use a procedural projection technique instead of painting
        - When enabled, a space option is provided to choose if this projection happens in world or local space
- Top Layer Texturing
    - Painting: The default mode, allows you to paint on this layer to choose which texture goes where
    - Project 3 way: Use a procedural projection technique instead of painting
        - When enabled, a space option is provided to choose if this projection happens in world or local space
- Per Tex Flow Mode
    - None: no flow mapping
    - Flow: flow mapping is on
    - Flow: flow mapping with refraction (two layer shader only)
- Per Tex Scale
    - Allows you to control the uv scale per texture
- Per Tex Material
    - Allows you to control metallic, smoothness and porosity per texture. Note that metallic and smoothness are only used if you aren't providing a per-pixel version. Porosity controls how a material reacts when wet, materials with lower porosity values have darkened albedo when wet.
- Per Tex Contrast
    - When using PerTextureProperties, you can optionally set the blend contrast for the splat maps per texture. Note that the layer blends are still controlled by the standard contrast slider.

- Per Tex Glitter
    - Turns on the glitter feature and provides a slider for glitter strength per texture
- MacroTexture
    - Use a macro texture over the whole mesh or terrain?
- Splats Fade To Macro
    - Sets the blend factors so the splat maps show up close and fade into the macro texture and hides the next two properties
- Layering
    - For blending, is the macro texture on top, or the splat maps?
- Blend Mode
    - How to blend the macro texture with the splat maps
- Detail Noise Texture
    - Adds the ability to have a single detail texture set for the entire splat map data. This is very cheap, much cheaper than the detail texturing mode, and still looks great.
- Per Texture Detail Strength
    - Allows you to control the detail noise strength per texture
- Distance Noise Texture
    - Basically the same technique as Detail Noise Texture, but instead of blending in close, blends in as the object gets further away. This can be used with low scale values to put a “macro” noise on the objects in the distance.
- Parallax Map
    - Should parallax mapping be used? This samples the diffuse textures a second time to create an enhanced depth effect. Also works with tessellation to create an even better depth effect (which might allow you to lower the tessellation factors).
- Emission Map
    - Should a separate Emission splat map be sampled?
- Distance Resampling
    - Resamples all splat maps at a second UV scale and blends them into the original samples. This can be used to help reduce tiling in the distance (though I personally prefer well designed texture clusters, which are much cheaper).
- Puddles
    - Allows you to paint into UV3.y to make puddles appear on the terrain. In the Terrain painter, a separate Puddles tab is available for painting puddles.
    - Multiple Puddle modes are available:
        - Puddles
            - A basic puddle for stagnant water
        - PuddleFlow
            - Stagnant water style puddles, but with flow mapping
        - PuddleRefractive
            - Refractive water, which distorts the terrain under it
        - Lava
        - For paintable lava surfaces

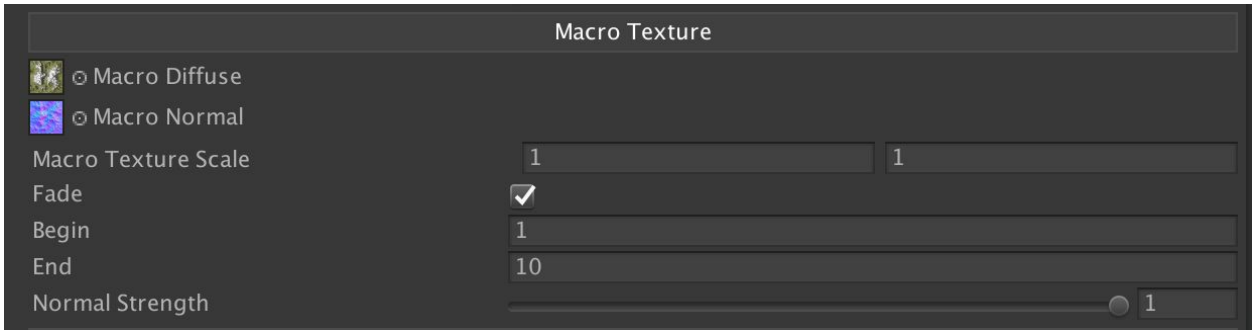
    - There is also an option (“Push Terrain Down”) for controlling how the puddles interact with the height map. When false, painting puddles builds water up on the current surface. When true, the terrain is pushed down as if being pressed below the water line. In many cases, this is a trivial difference, but if you are animating the collection of water into an area, you wouldn’t want the stones to move, but rather have the water build up. However, if you are dealing with mostly static terrain, then having the rocks pushed down as you paint water into an area keeps that area flat with a level water plane.
    - Puddle Foam is available when using PuddleFlow or PuddleRefractive modes, which allows you to pack a foam texture into the B channel of the normal map. Foam will appear around rocks and across the water surface.
    - Puddle Refraction Depth Dampen, available when using the PuddleRefractive mode, adds a slider which controls how much refraction is dampened in the deep areas of the terrain. The idea here is similar to foam, in that we want areas of the water which are deeper to have calmer, wider refraction, while areas around the rocks, which are shallower, to have more turbulent refraction.
    - A glitter option is available to add glitter to puddles
- Rain Drops
    - Allows for animated rain drops to appear in the puddles. Rain can be ramped up and down as your weather changes.
    - No Rain On Walls option allows you to filter raindrops based on normal angle, so that they don’t appear on walls
- Wetness
    - Wetness allows you to paint how wet a given surface is. This can be very useful in conjunction with puddles, or as a standalone effect on cave walls and such. Wetness is painted into the UV1.w channel, or using the terrain painter’s wetness tab when working with Terrains.
- Snow
    - Snow allows for a global snow texture to be used across the scene. The normal map texture is always in NormalSAO format. Snow is designed to interact with puddles, allowing you to melt away the snow into puddle formations or streams, revealing the rock below it.
    - A “snow over macro texture” option is also available, which will apply the snow after the macro texture is applied, allowing dynamic snow to cover a distance texture.
    - A snow glitter option is available to turn on glitter for snow
- Disable Bottom Layer Blend
    - Disable height based blending on the bottom splat map layer, selecting only one texture for each triangle. This is a useful optimization when texturing non-organic surfaces with MegaSplat. Imagine that you have a house with a wall, roof, and floor texture. You could assign all vertices on the roof to one texture, all vertices on the floor to another, etc, to fully texture the house avoiding the need to pack textures into a texture sheet. This would work fine, but you’d have the issue that

each texture is being sampled 3 times and blended across the face. Turning on this option will make it so only one texture is chosen per triangle and no blending is used, greatly optimizing the shader.

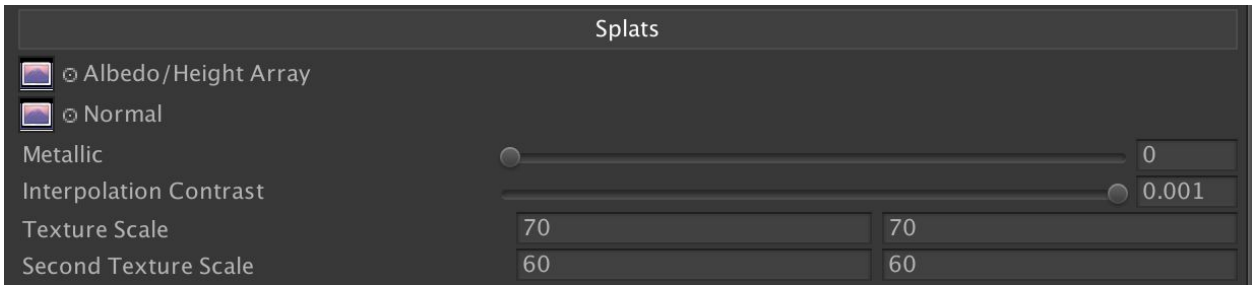
- Low Poly Look
    - Invokes the geometry shader to give each triangle a hard edge. A second option is available which allows you to blend between hard and soft edges.
- Process Mesh
    - Does the mesh conversion process in the geometry shader. It is highly recommended to do the preprocessing of the mesh data instead of using this option, but if you absolutely cannot preprocess your mesh or construct your mesh to MegaSplat requirements (or just want to see the results before doing that), this option can be enabled to do this work in the geometry shader. However, this is much slower.
- Curved World
    - If the Curved World asset is installed, this option will generate shaders with curved world support (See Curved World for more information on the asset store).
- Custom Code
    - When turns on, MegaSplat will generate several special files for your shader that allow you to insert custom shader code at various parts of the pipeline, allowing you to customize MegaSplat beyond what's provided in the package
- Shader Level Override
    - Allows you to increase the shader level on the compiled shader. This can be useful when you want to explicitly force compatibility with a specific platform, or need extra texture samplers for your custom code. Note that if MegaSplat needs to increase the shader model (say, to 4.6 when enabling tessellation), it will increase it above this setting to make sure the shader runs.
- Debug Options
    - Allows you to view the outputs of each channel (albedo, height, normals, etc). This is used by the Render Baker to bake out textures, but is also useful for debugging.

Shader Options

The rest of the material settings are traditional options for the shader. The are explained below, but the options you see will depend on your shader settings:



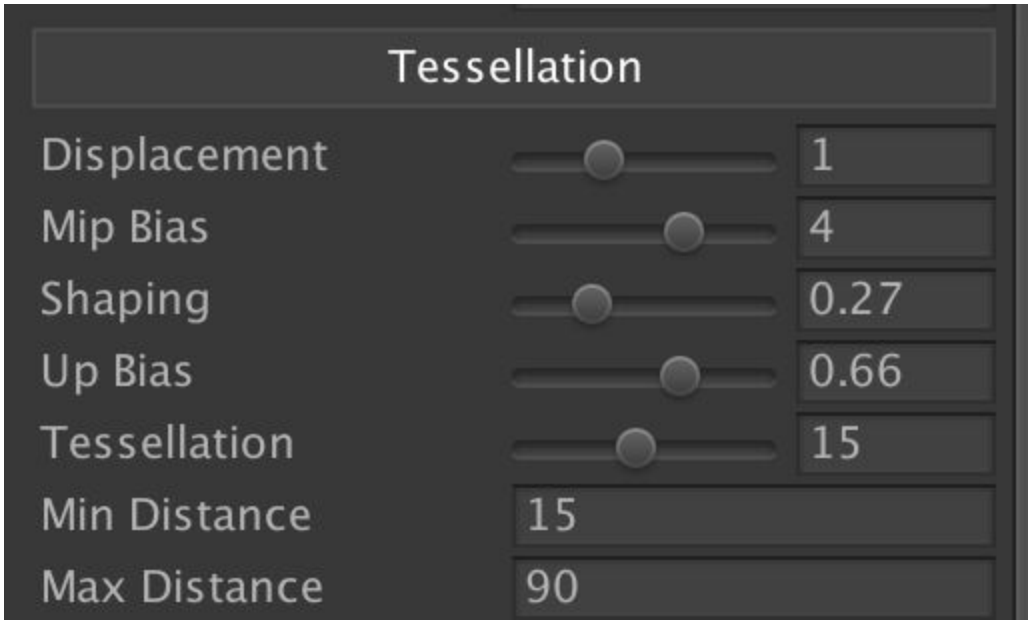
Macro texturing allows you to assign a texture for the entire object or terrain. A number of options for how the macro texture is blended with the splat maps are provided in the shader generation section. The most common technique is to blend between the splat map data and a macro texture, which acts as an optimization in the distance, allowing you to stop performing the splat mapping functions and only sample the macro texture. However, macro texturing can be used in other ways, such as a global tint, or as a horizontal striping texture for striated terrains (use a second UV set to project the texture horizontally across the landscape).



The Splat Map section of the shader. The Interpolation contrast controls how sharp the transition is between textures. As you go lower in values, the transition becomes softer, and as you go higher, becomes much more exact. This setting will have a huge effect on the blend between textures! You can also control this per texture with the PerTexContrast option.

The Texture Scale property allows you to scale the size of the splat map textures vs. the main UV channels (or triplanar UVs).

The Parallax height (shows when Parallax is enabled) allows you to create an increased depth effect at the cost of resampling the diffuse textures. Values that are too high can show small artifacts, and a value of 0 will have no effect. A parallax fade distance specifies the distance at which the parallax effect will start fading out. At 2 times this distance, the parallax effect will stop being computed, making the shader cheaper in the distance.



The Displacement is the total distance from the original surface a tessellated vertex can be pushed.

Mip Biasing allows you to reference a lower mip map level when performing displacement. This is both slightly faster, and usually provides a visual quality increase by not allowing the displacement to create as many spikes from small details in the height map.

Shaping is a separate contrast control for how Splat Mapping layers are blended together.

Up Bias allows you to bias the displacement towards an upward vector instead of along the normals. On surfaces such as pebbles and small rocks, having them protrude out from the side of the slope looks wrong.

Up Bias and Displacement amount can be adjusted per-texture when per-texture properties are enabled.

In Distance based tessellation, you have a tessellation factor (how many times to divide a given triangle), along with a min and max distance in which this value is interpolated.

In Edge based tessellation, you have a maximum screen size that an edge will get before it is subdivided.

The min/max values are also used to fade out the displacement, hiding the transition.

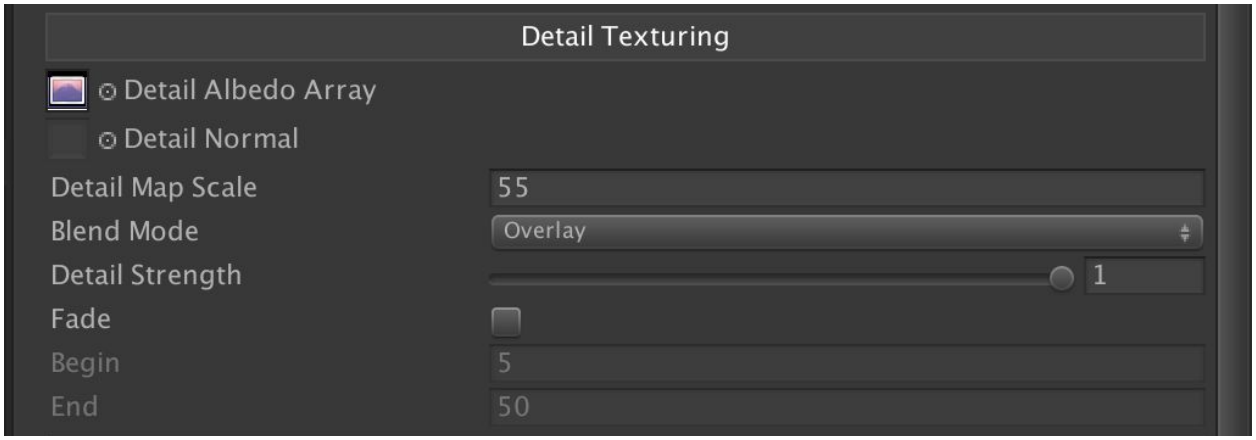
Tessellation can be a very expensive feature, so it is suggested that you keep the tessellation factor or edge length as conservative as possible.

Working with Tessellated Meshes

Cracks can appear in meshes when using Tessellation. These can be caused by a vertex which shares the same position, but not the same normal (a hard edge) or UV coordinate. You can paint displacement dampening into the UV3.z channel of the vertices to suppress displacement at this points. To do this, switch to the Paint tab in the vertex painter and select UV3_z as the target channel. Set the brush value to 1 and paint the vertices. You can

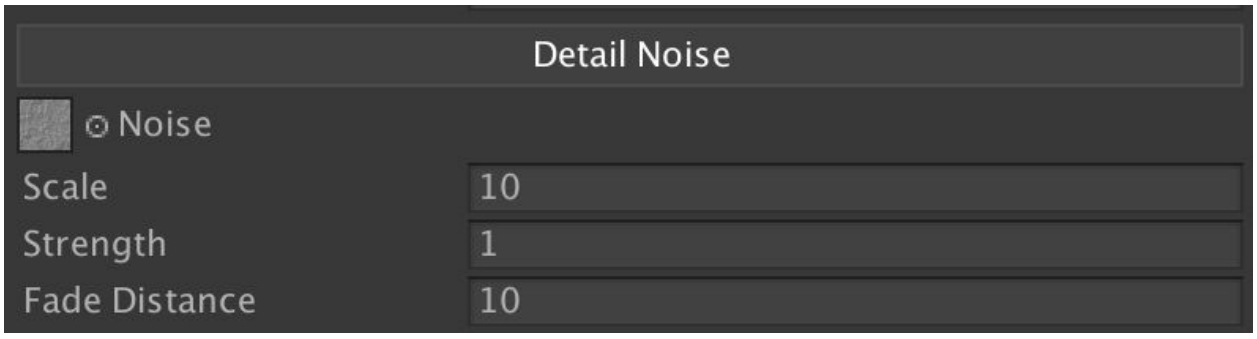
also use this to control the amount of displacement per vertex. A value of 0 will have maximum displacement, while a value of 1 will have no displacement.

Detail Texturing

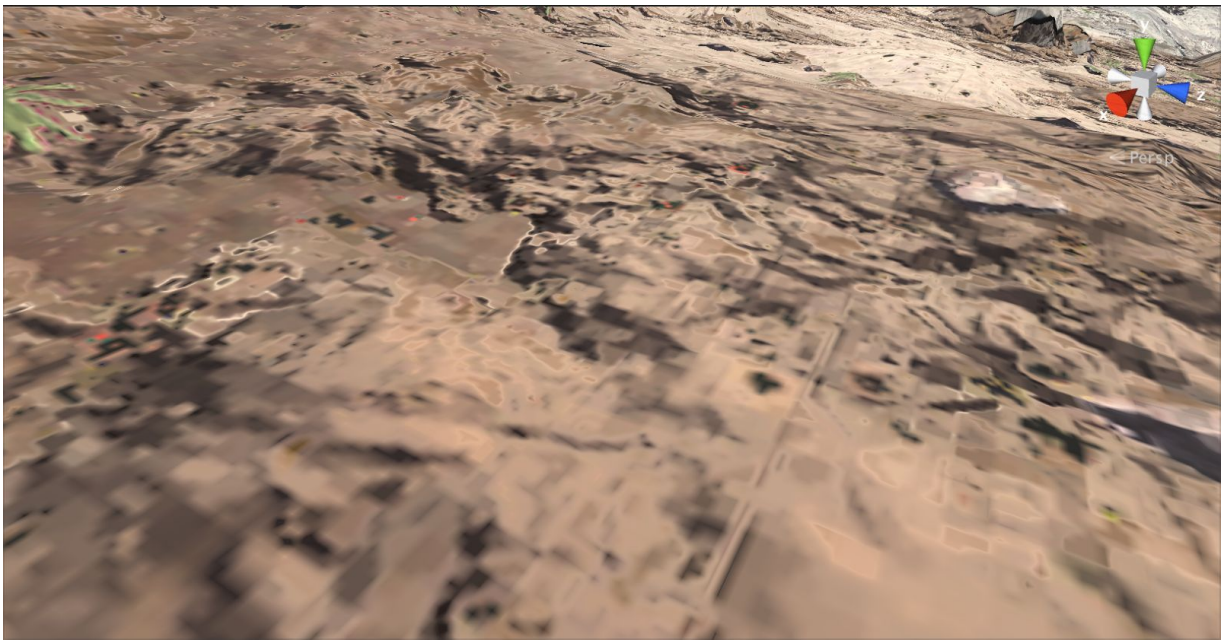


When using the Detail Texture mode, Detail textures can be blended in as you get close to the surface, providing an enhanced sense of resolution. Note that the detail textures in this mode expect a texture array, and the detail textures pull the same index as the main splat textures do. Otherwise, the scale, blend, and fading options are similar to the macro texture options. For many use case, Detail Noise is a more preferable option as it doesn't require an entire detail texture array and doesn't use MegaSplat's second layer for detail texturing.

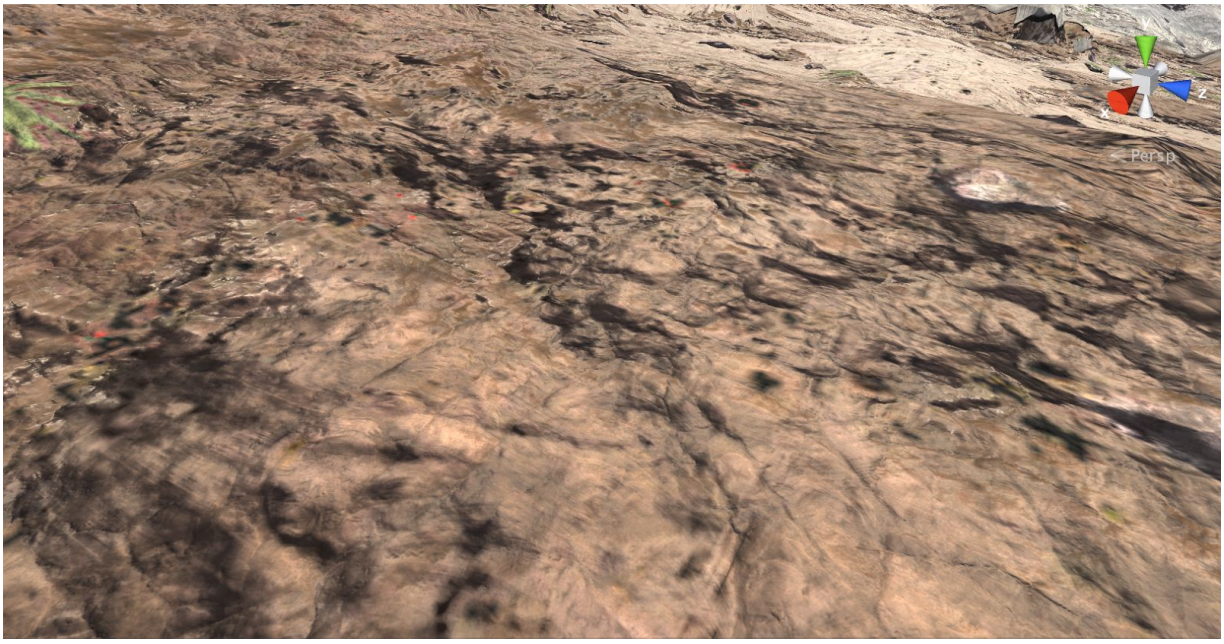
`



The Detail Noise option is a much cheaper form of detail texturing which works with all the other layer modes and UV modes. With this option enabled, you have one detail texture for the entire terrain, and do not have to have a separate texture array for detail textures. This makes it much cheaper than using the Detail Texture mode, and can still look amazing, preventing things from ever getting blurry due to lack of texture resolution.



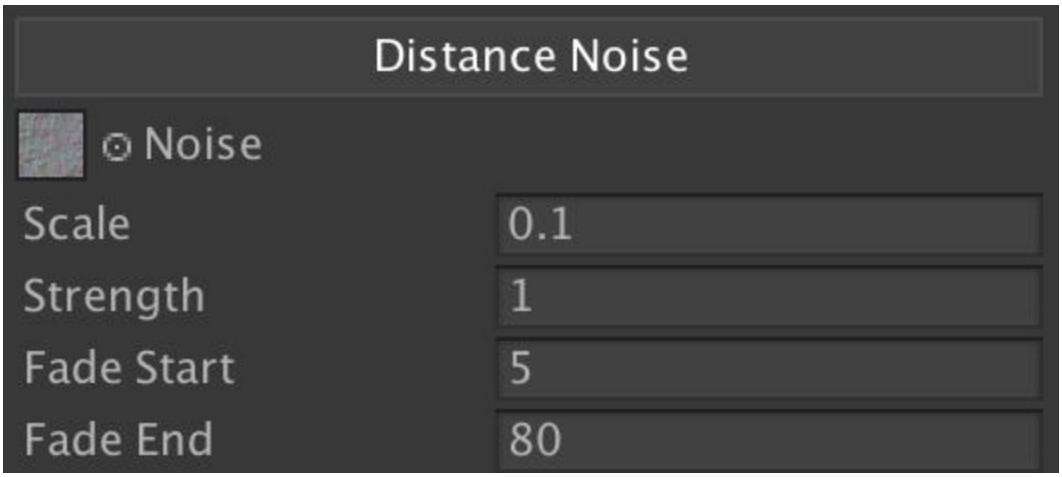
Without Detail Noise, Notice the blurry textures when the camera is too close



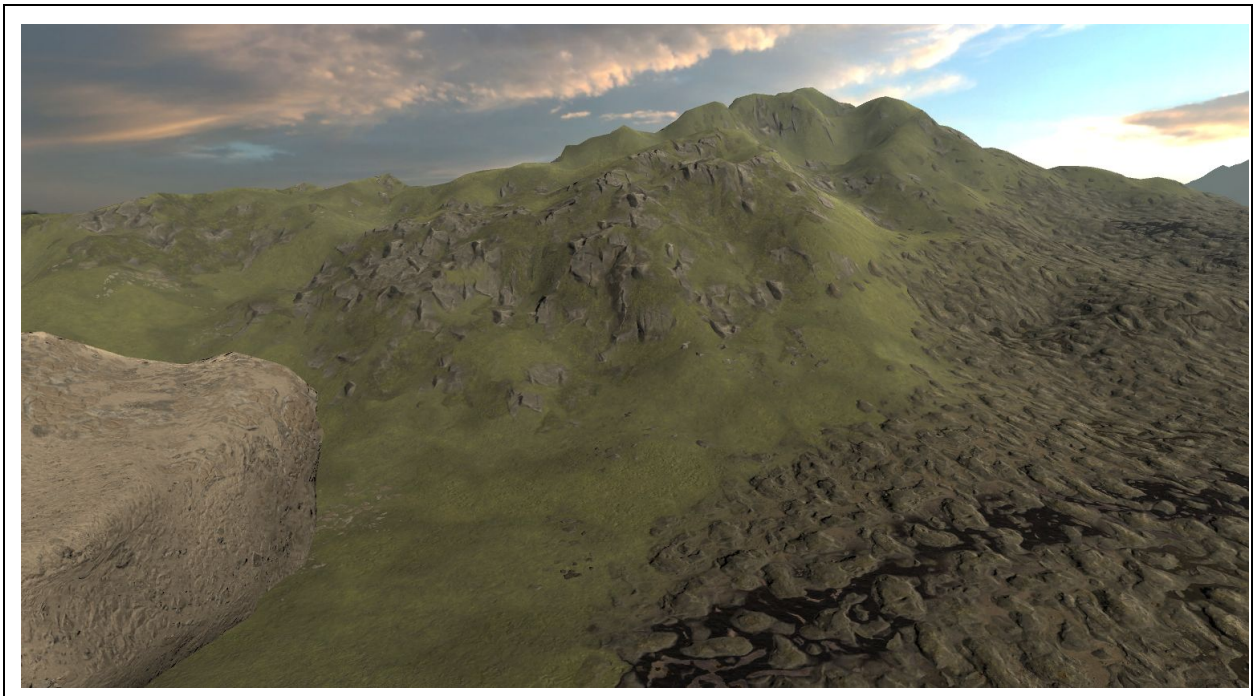
With detail noise, blurry pixels are a thing of the past

The textures Blue channel is used to modulate the albedo, which the Red/Green channels modify the normal, and the Green channel introduces a subtractive noise into the smoothness. You can think of these as a normal map + luminance in the blue channel, but to be honest it doesn't really matter that much as long as the noise is centered around 0.5 in the image and the texture is linear. You can enable Per Texture Noise Strength in the shader generator to control how much this detail shows up on each texture.

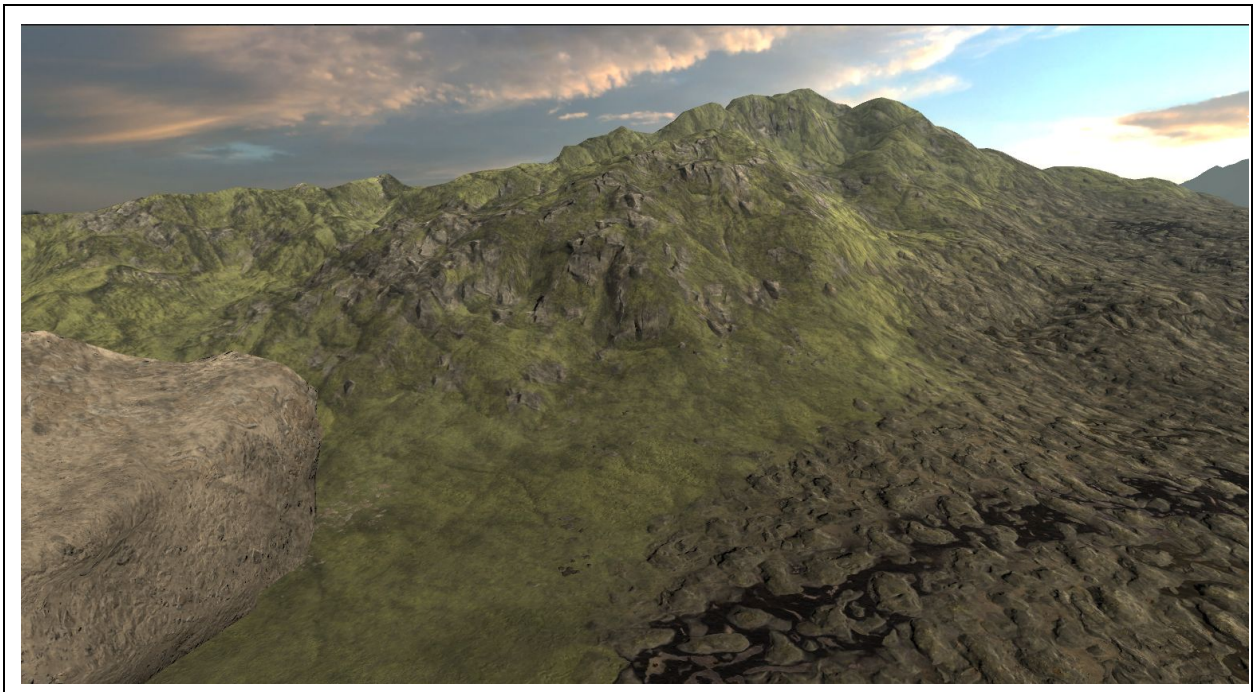
DIstance Noise Texture



Distance Noise Texturing is the same basic technique as Detail Noise, but begins to fade in at the start distance and is fully faded in by the end distance. This can allow you to introduce a ‘macro’ style noise texture in the distance.

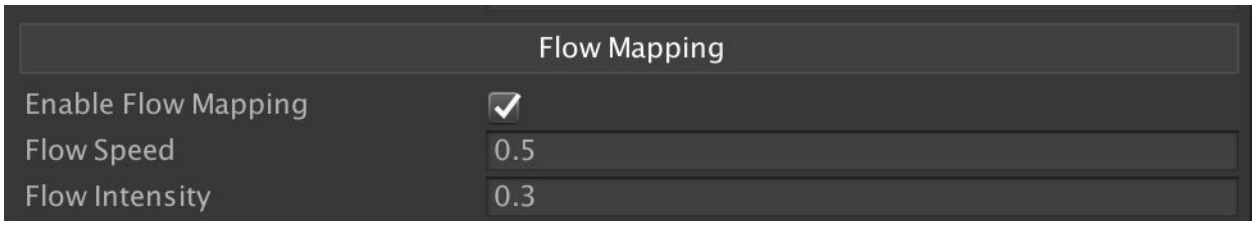


Without distance noise



With Distance Noise

Flow Mapping

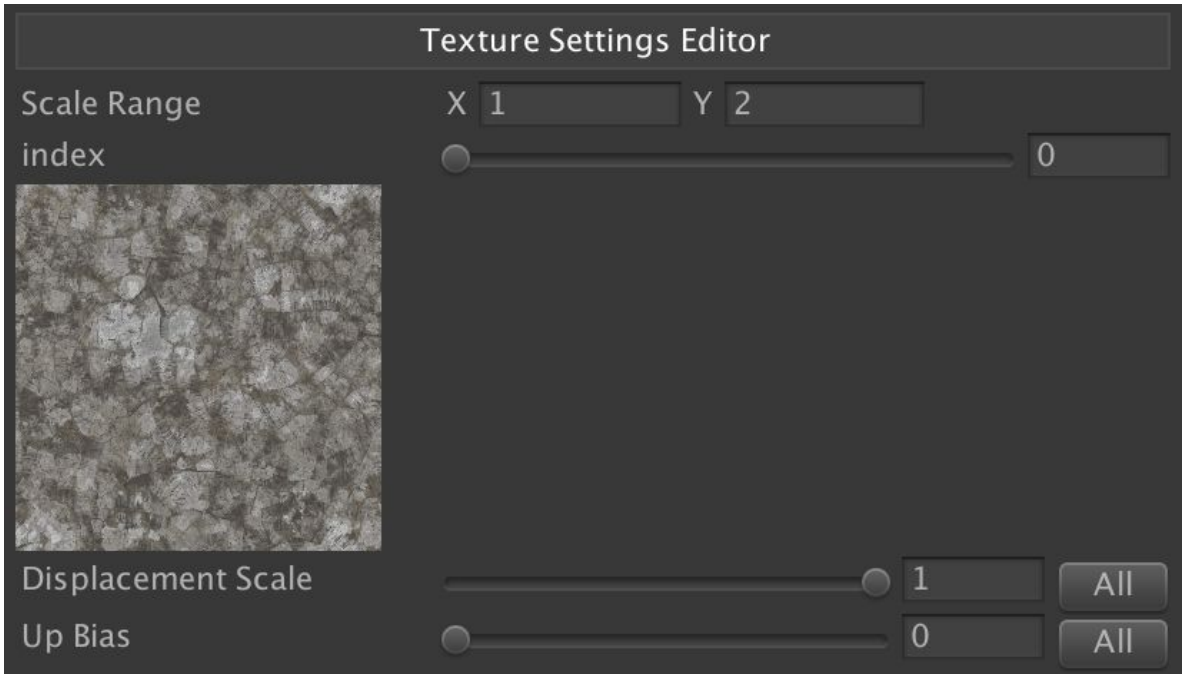


Per Texture Flow Mapping allows for the appearance of moving surfaces. Users can paint the flow direction using the painting toolset, for meshes or terrains. When enabled, the global speed and intensity settings are exposed, though the user can adjust the flow speed per-texture in the next block. By default, no flow mapping is applied to each texture; you must set a speed value for the textures you wish to flow map.

Flow direction is painted into the UV2 zw channels, or via the Flow tab in the terrain painter. To paint flow directions on meshes, switch to the Flow tab, select the UV2_ZW channels, and paint onto a texture in which you have turned up the flow speed. You can also turn on the preview mode in the Vertex Painter window, and preview flow directions on a water texture or as arrows across the surface.

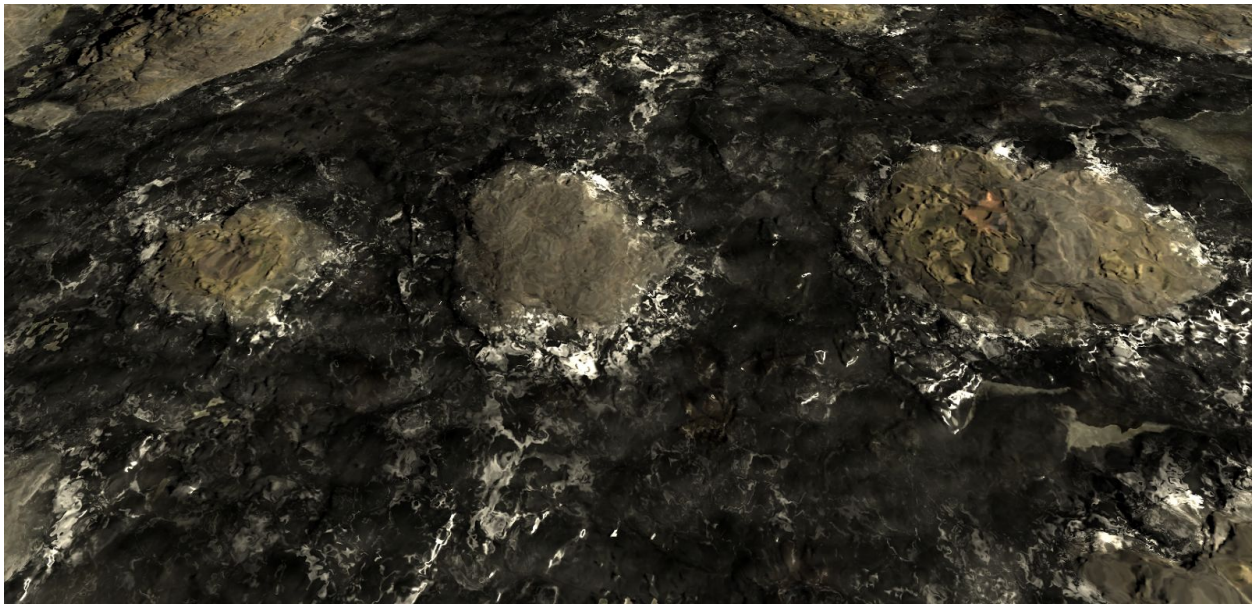
To see flow maps animating in the editor, you will have to turn on Animated Materials in the menu bar. Note that when using a two layer shader, only the texture painted onto the top layer will flow; this is so it can be optionally alpha'd with the bottom layer.

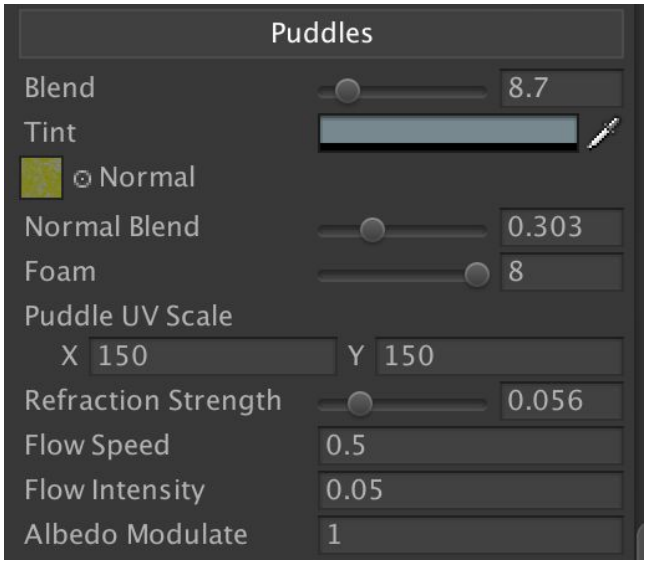
If you are using the layered shader, flow mapping is applied only to the **top** layer. Alpha and Refraction values are available, allowing you to create water like effects which show and distort the texture of the first layer. When refraction is on, the per-texture metal and smoothness properties are instead use to scale the alpha and refraction values of each texture. This allows you to have both water (transparent/refractive) and lava (not transparent/refractive) in the same shader. For this to work, make sure the global Flow Alpha is set to 1, and then set the per-texture flow alpha on each texture.



It can often be useful to have settings for each texture in the array; however, Unity does not provide support for array properties. To work around this, when any per-texture property is enabled in the shader generator, or when Flow mapping is enabled, a 256x4 texture is written out to store various values for each texture.

This window will only show the properties you enable. Select the texture using the index slider and adjust any properties for that texture. If you wish to set all textures in the array to use that value, press the All button, and the current value will be applied to all textures.





The puddle system in MegaSplat is capable of producing stagnant water puddles, like those seen on a city street, as well as streams which flow across the landscape complete with foam which interacts with the surface. The main benefit to using the puddle system over the flow mapping system is that it interacts with the terrain in a more believable way, and is usually cheaper than flow mapping.

When a puddle mode is enabled, you can blend in small puddles by painting in the UV3.y channel or by using the puddles tab in the terrain painter. A value of 0 will show no puddle, and a value of 1 will be fully submerged.

Puddles and wetness introduce some new material concepts you may not be familiar with, such as porosity. Porosity represents how porous the material is- very porous materials will collect more water than non-porous materials, and thus exhibit a shinier and darker surface when wet. Because porosity shares many similarities with roughness, per-pixel porosity is

calculated based on the smoothness maps, and then modulated with a per-texture or global porosity level.

The puddles feature contains multiple modes, ordered from least expensive to most expensive.

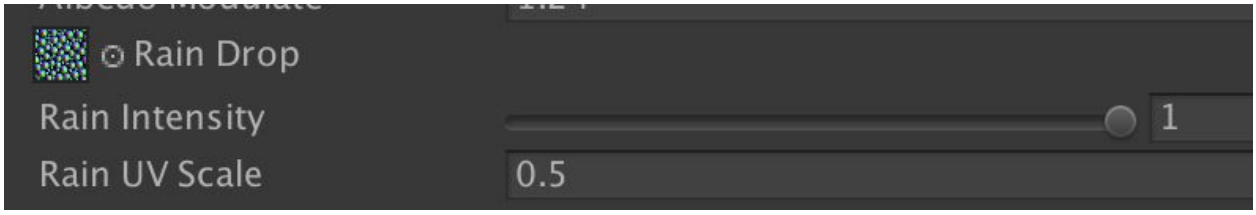
- Puddles
    - This is useful if you need simple, stagnant pools of water.
- PuddleFlow
    - For a little bit more cost, you can flowmap a normal map across the puddle surface.
- PuddleRefractive
    - This allows for the puddle normal map to refract the surface under it
- Lava
    - This is a separate system which uses the puddle input data to paint lava flows

When in PuddleFlow or PuddleRefractive modes, an option for Foam is available. The foam texture is packed into the B channel of the normal map used for the puddles. Foam will be most prominent around rocks which intersect with the water.

The Normal Blend slider allows you to control how the normal is affected by water. If you were to look at a stagnant puddle, the surface of the water would be completely flat. When NormalBlend is set to 1, water areas have completely flat normals. When set to 0, the normal of the original surface is used. This value should be set close to 1 for stagnant water, and closer to 0.5 for something like flowing water.

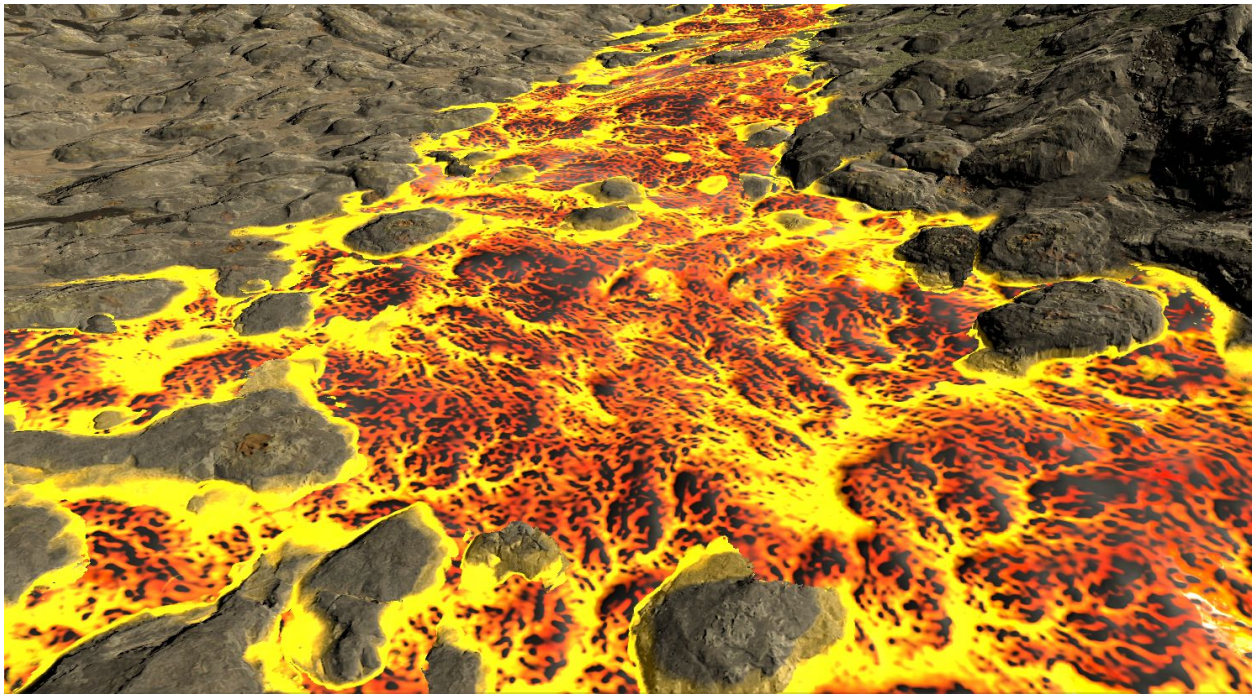
A water tint can give the water a subtle tint. A value of 0.5, 0.5, 0.5 has no effect. Finally, flow settings are available to control the speed and intensity of the flow mapping.

Raindrops

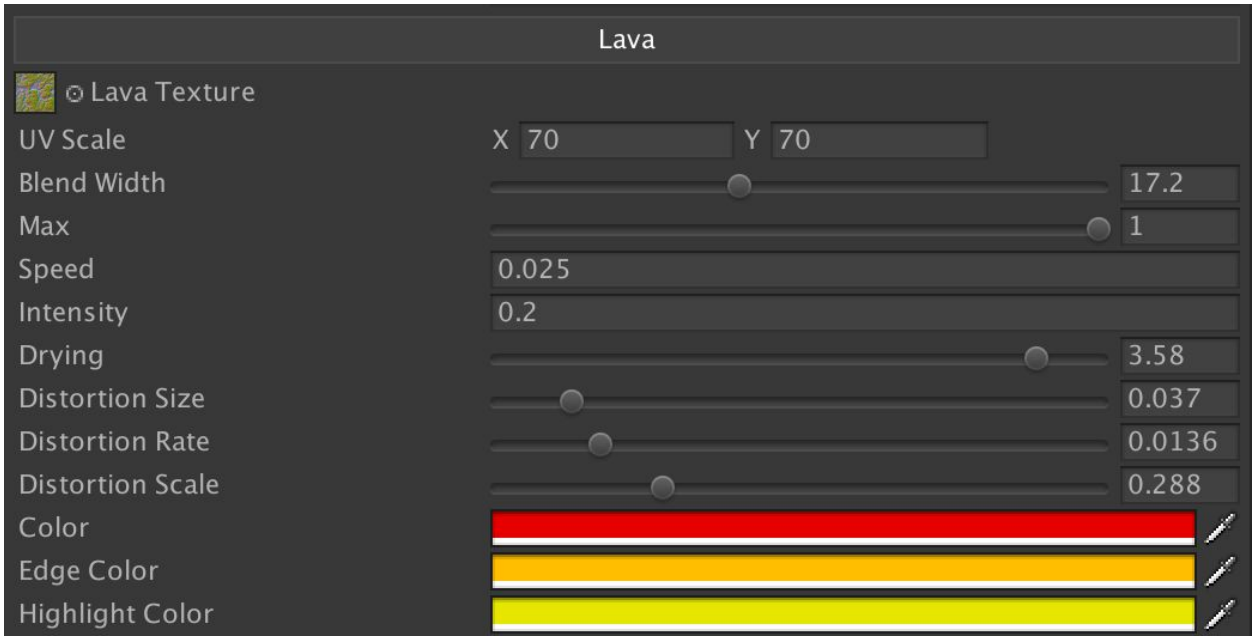


When raindrops are enabled, animated rain drops can appear in puddled areas. Raindrops require a specially constructed texture which is included in MegaSplat (water_ripple, in the examples/texture directory). Rain intensity controls how visible the puddles are. Rain UV Scale allows you to scale the rain relative to the texturing on the puddles.

Lava



When you set the puddle type to Lava, a separate interface is available for adjusting lava parameters. Lava is painted the exact same way as puddles, thus a material may either have lava or puddles, but not both.



The texture used for Lava uses a custom packing format. The red and green channels of a normal map are packed into the red and green channels of the lava texture. The blue channel contains a sort of height map, used to color the surface. The alpha channel controls the drying

of the lava; a value of 0 will be purely liquid, and a value of 1 will be dried, darkened lava. Two example textures (lava_flow and lava_flow2) are included in the examples folder.

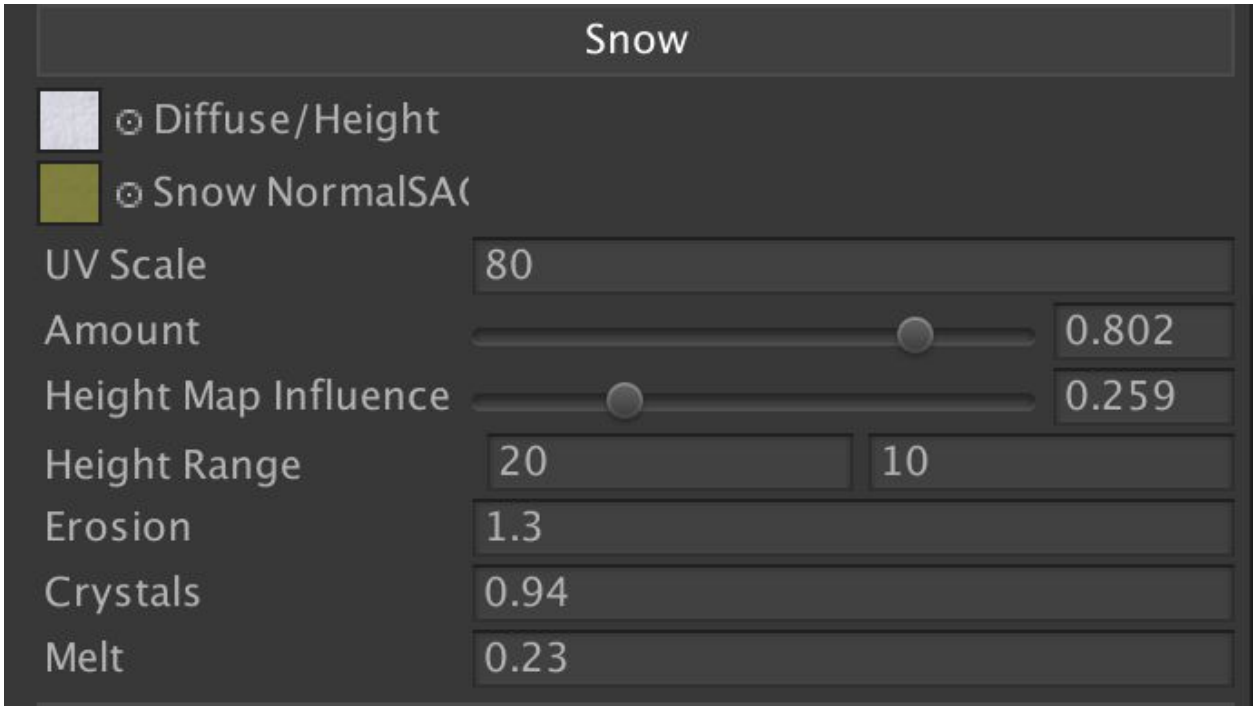
The UV Scale property can be used to scale the lava texture. The blend width controls how quickly the transition between lava and the underlying surface happens. The max value allows you to ramp the maximum amount of lava down, removing lava from the entire scene. Speed and Intensity control the speed and cycle time of the flow mapping on the lava; think of speed as the constant motion, and intensity as the amount of rolling in the flow map effect.

Drying controls the amount of cooling on the lava. When drying is set to a high value, large areas of cooled lava will appear in the texture. Note that this is based on the alpha channel of the lava texture.

Distortion size, rate, and scale effect the amount of distortion, the speed of the distortion effect, and the size of the distortion texture. Note that all 4 channels of the laval texture are used as inputs to the distortion function.

The gradient in the blue channel of the lava texture is used to blend between the main lava color and the highlight color. The edge color outlines the lava area, creeping up onto the rocks.

Snow



When Snow is enabled, you can use height and amount filters to control how much snow appears across your landscape. Snow uses an albedo + height map texture, and a NormalSAO texture (Normal RG in RG channels, smoothness in B, AO in the alpha channel) . The UV scale is based off of the macro textures uvs, and scales accordingly. The amount controls the total amount of snow across the terrain. The height range controls the starting point for snow to start appearing, and the number of units to blend to full snow coverage. The heightmap influence

controls how much of the terrain's original height values are used to affect where snow gather. Positive erosion values cause snow to melt on the high parts of the surface, while negative values will cause it to melt on the lower parts of the surface first. The crystals value controls the specular response of snow- higher values look more icey, while lower values look more fluffy. Finally, melt affects the BDRF around the edges of the snow to make it look more wet.

You can control the 'up vector' of the snow. By default, it points up, but changing the values can allow snow to build up more on one side of a mountain than another.

Puddles and tessellation all interact with snow. If you paint puddles over snow using a low blend value on the puddles, they will first melt the snow revealing the surface underneath, then water will accumulate in the shallows of the surface. As snow fills in the cracks, the surface will rise, become smoother, and take on the displacement aspects of the height map in the snow texture.

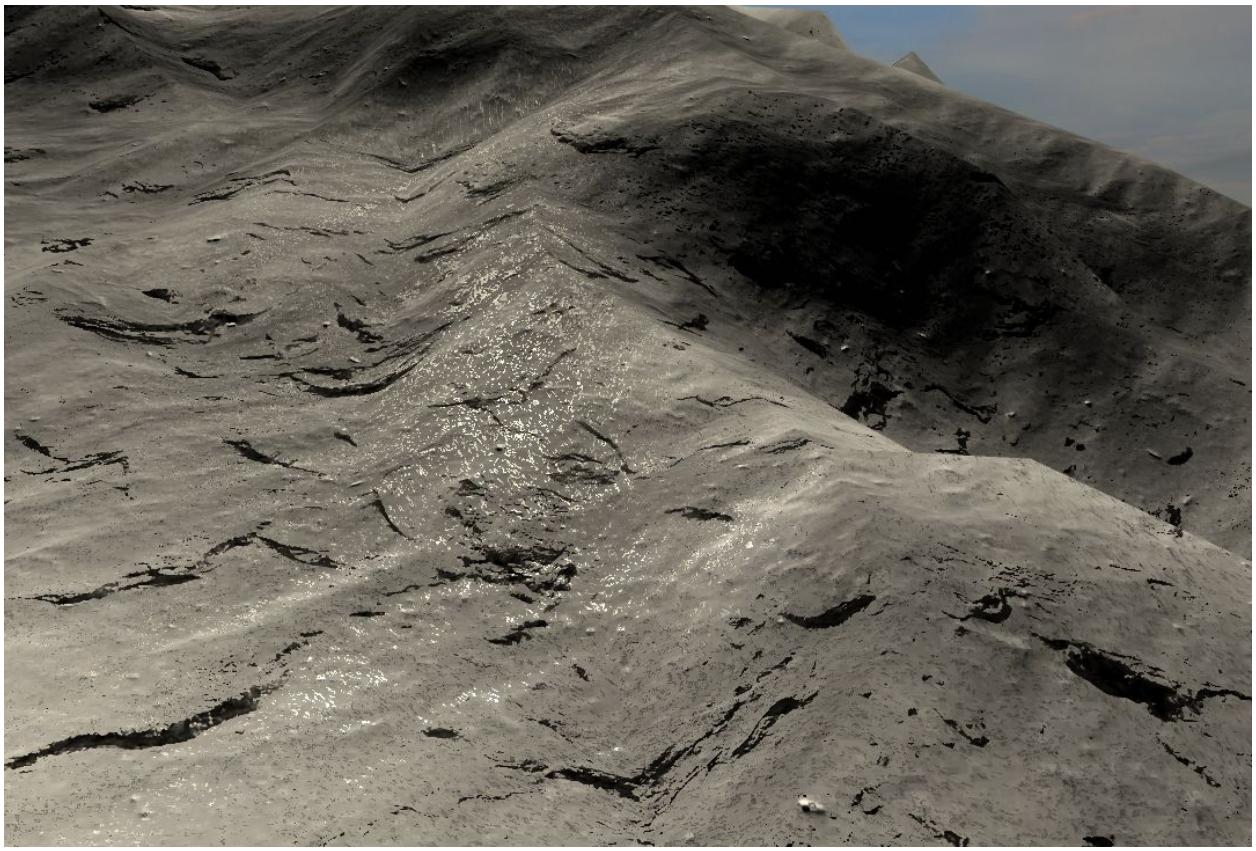
When animating snow to be added or removed from the scene, it's best to adjust multiple values at once. Snow should accumulate with a low height map influence, erosion, crystal, and melt factor. Once show has finished accumulating, these values should increase to harden existing snow into ice (crystal), remove snow from the tops of rocks (erosion/height map influence), add wet around the edges of snow (melt) and finally reduce the overall amount of snow. There is an example component for animating these values, and an example animation can be scene by enabling the component in the test scenes and sliding the time slider.

Wetness

Wetness can also be painted. On meshes, this is stored in the UV1.w channel, or using the Wetness tab in the terrain painter. Painting wetness darkens the albedo and increases the smoothness of the surface based on the porosity and roughness of the underlying texture. Porosity can be set as a per-texture property, or globally. When surfaces are porous and rough, water collects in these areas and produces a darkening of the albedo and increased reflectivity.

	Wet rock with high porosity in nature. Notice the darkening and increased specular response.

Glitter



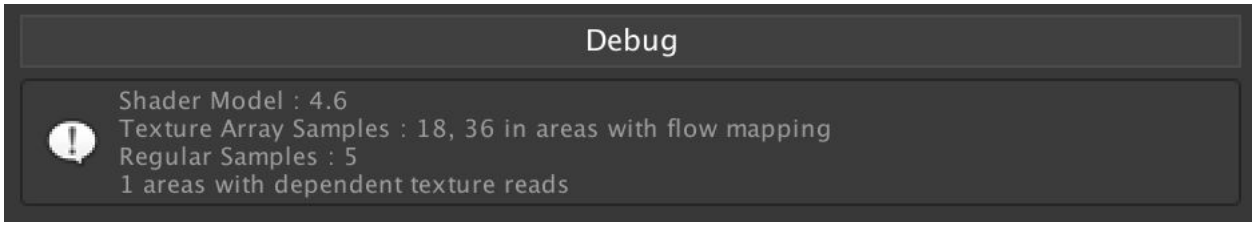
Glitter adds small sparkles to surfaces, and is available for snow, puddles, or on a per texture basis. A small texture is used with sparkles in the red channel (example texture is glitter_noise), and an noise texture in the green channel which is used to cause the glitter to shimmer. This texture should use the fade mips option.

A UV scale is available for the glitter texture, along with a reflective value to control how strong the glitter is. Snow and Puddles also have a shininess value, which controls the width of the shine area.

Alpha Mode

When alpha mode is set to Transparent or Cutout, a extra texture array is exposed to hold the alpha value of each splat texture, and the material is drawn accordingly. Alternatively, if all you want to do is cut holes into a terrain, you can enable the “Alpha Hole” option, which allows you to specify a texture index for holes. If you set the index to 0, for instance, then anywhere you paint the first texture in your texture array will create a hole.

Debug

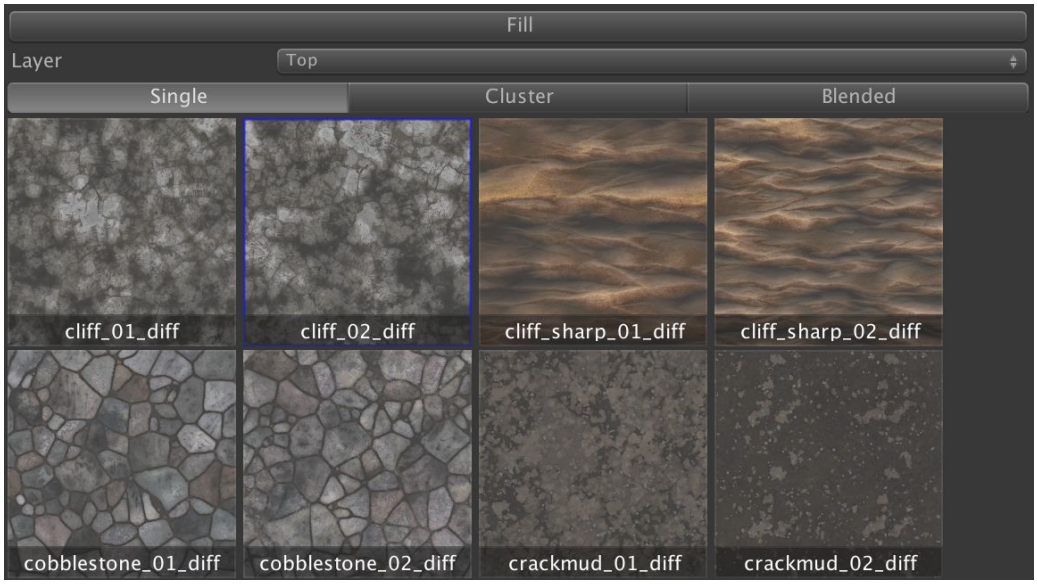


The debug panel gives you some indication of how expensive your shader is, as well as which shader model is required. Adding features such as tessellation, snow, or a second UV can increase the minimum shader model required. See Performance Considerations for more information on this area.

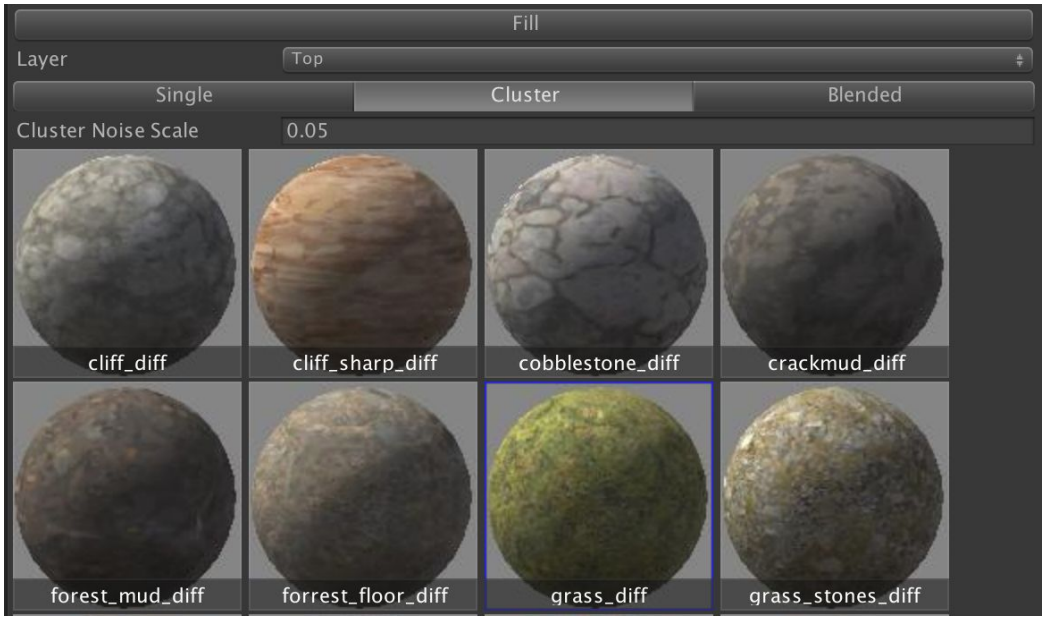
Paint a mesh

Tutorial Video:
https://www.youtube.com/watch?v=RNT94fruCl0&t=20s

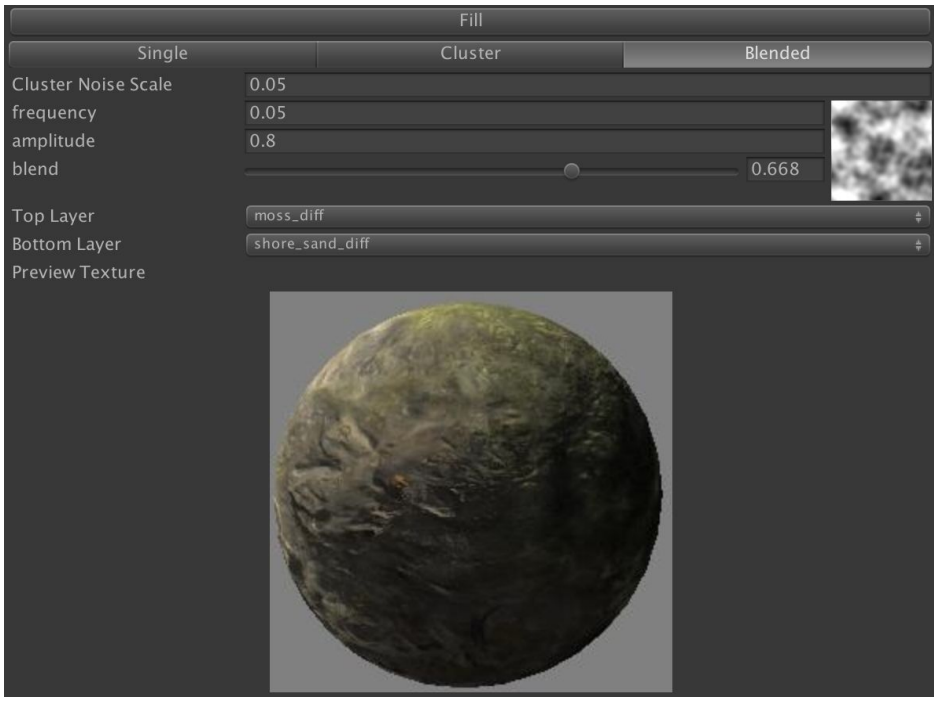
- Drag your converted Mesh into the scene.
- Assign the material to your mesh
- Open the Vertex Painter window at Window->Vertex Painter Pro
- Make sure your mesh is selected
- Go to the ‘Custom’ tab
- Assign your Texture Array Config using the “Brush” field
- Change the brush mode to “Single Bottom”
- Make sure the “Active” toggle is checked in the Vertex Painter window
- Select a texture from the brush UI and begin painting



The “Single” brush mode allows you to select and paint a single texture onto your mesh.

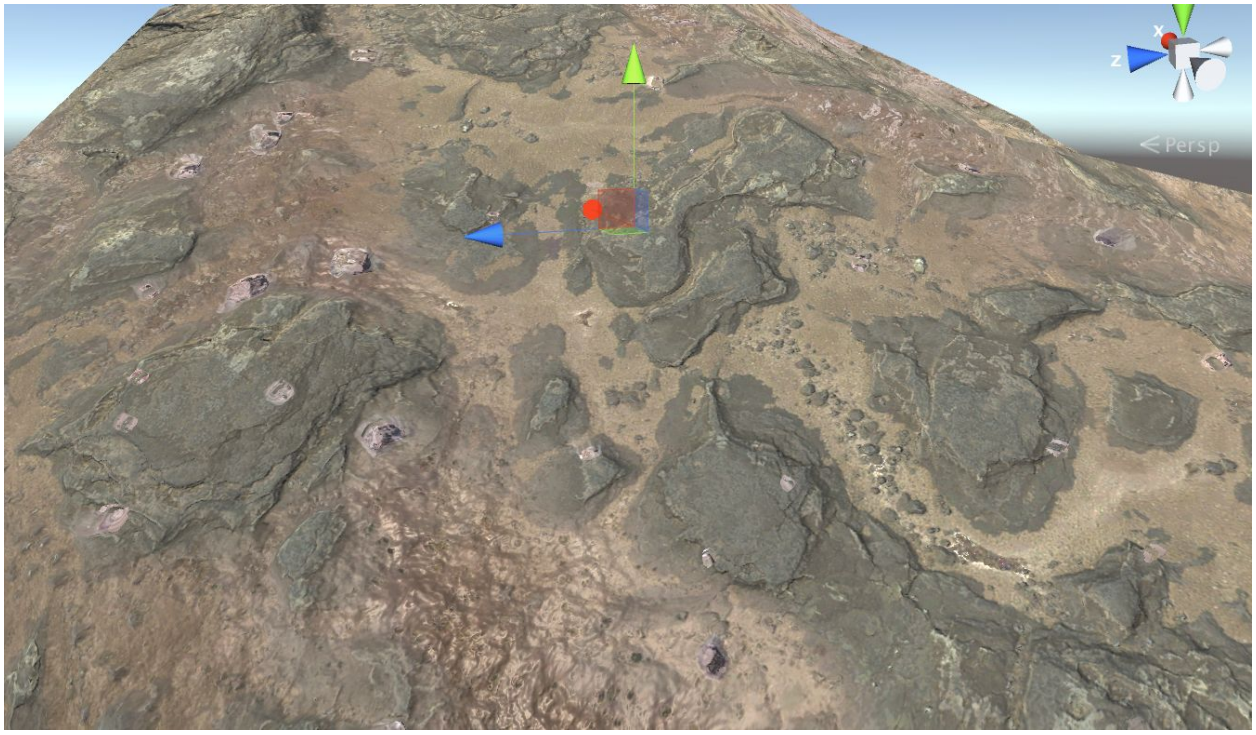


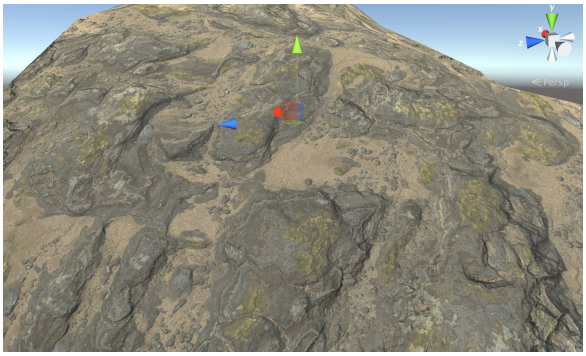
The “Cluster” brush mode allows you to paint textures chosen from a Texture Cluster onto your mesh.

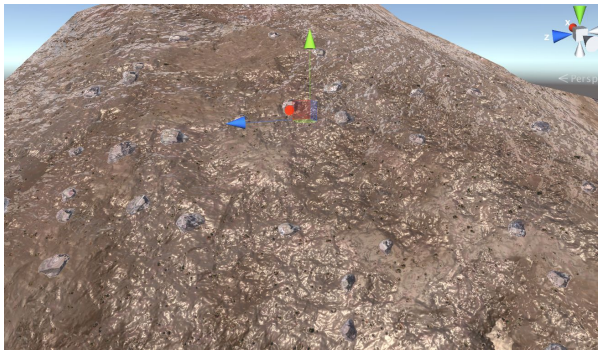


The “Blended” brush mode produces rich results from blending two Texture Clusters together via a noise function, painting onto both layers at once. A live preview is updates as you change the parameters of your brush.

Below is an example of the types of rich surfaces that can be created with this technique. Two texture clusters are being blended, each of which select from several source textures.







The first cluster mixes 3 mud textures with a noise function, while the second cluster mixes 5 rock textures together, and the two are mixed via a third noise function. Tiled over a large area, there are no visible repeats, and the detail holds up even when close.

When working with a 2 layer shader, there is also a ‘Auto’ option for the layer mode. This will look at the current information and paint on whichever layer has the lowest weight.

Painting on Unity Terrains

MegaSplat also works with standard Unity Terrains. Create a new terrain and open the terrain painting toolset from Windows/Terrain Painter. Select the terrain, and you’ll be prompted

with a window explaining the steps that need to be performed to setup a terrain. However, you can just click “Set Everything up for me” and the terrain painter will create the required shader and material and setup the terrain to use them. Once this is done, you’ll need to select the material from the terrain’s setting menu and edit it, at minimum you will need to assign a diffuse texture array.

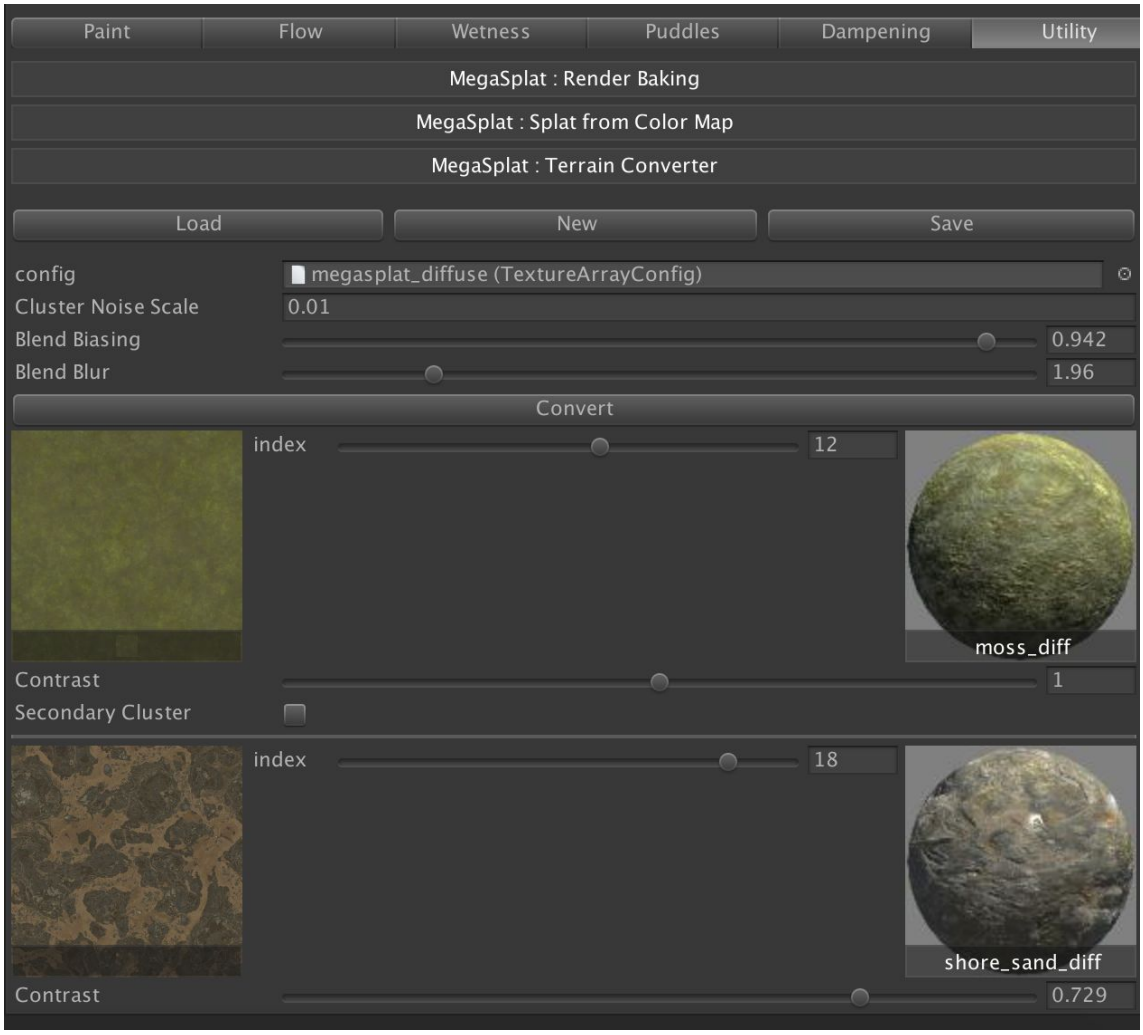
Unity does not allow developers access to the low level vertex data, and since Unity uses an LOD system for terrain, the vertex data is not consistent anyway. So when working with Unity terrains, the system will write all terrain data to a texture next to your terrain object. You must press SAVE when you finish painting to save this data to disk!

Converting existing Unity Terrains

Original Tutorial Video (rather old now, but still relevant):
https://www.youtube.com/watch?v=ykFWlhwX2_0&t=23s

If you have an existing Unity Terrain, perhaps one you’ve created manually, or generated from a landscape generation system, you can easily convert that Terrain to use MegaSplat’s shader system. To do this, select the terrain and open the Terrain Painter window, and use the automatic setup button to setup the material, shader, and terrain. You will need to select the generated material, setup the shader settings and assign the diffuse texture array.

With the terrain selected, switch to the Utility tab and open the Terrain Converter tool. Select your diffuse texture array.



This interface shows you the current Unity Terrain textures on the left. On the right, you can select a texture cluster which matches this terrain. When you have mapped all of the source textures to clusters, press convert to convert the current splat mapping data to MegaSplat format.

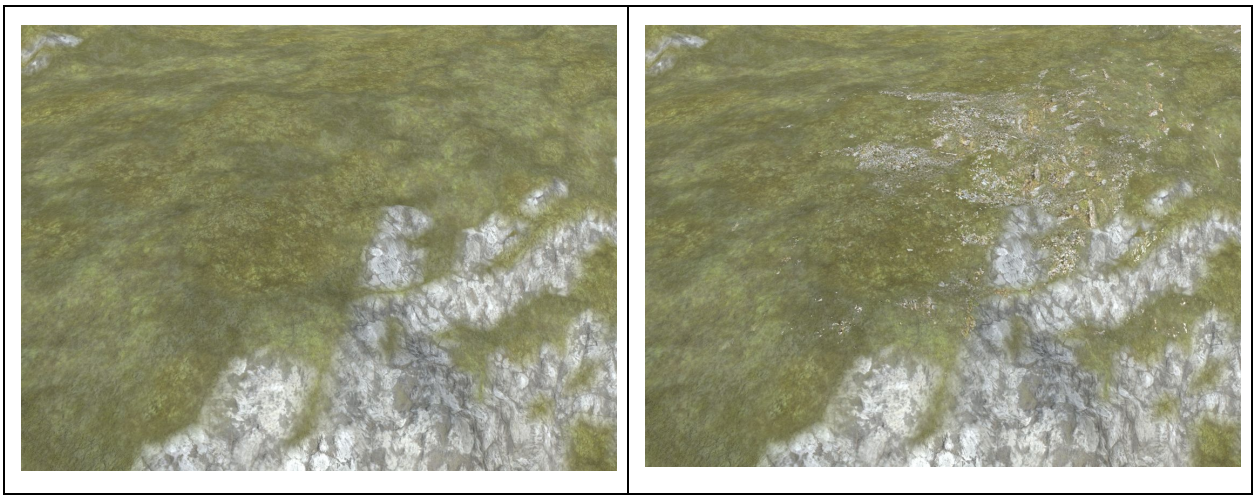
Note the settings on the top, above the texture->cluster mappings. The cluster noise scale allows you to set the global scale for cluster noise, the primary anti-tiling technique offered in MegaSplat.

Blend Biasing allows you to bias the blend between the two layers towards the center point, such that the two textures are mixed in more areas over the blend area.

Blend Blur will blur the original splat map image. This will create wider blend areas between the layers.

Each texture cluster mapping also has a contrast slider- this allows you to bias the blend to make one texture more predominant than another in the conversion.

Note that Blend Biasing, Blend Blur and contrast are only really useful with a Two Layer shader.



You can turn on a second cluster option for any given mapping. This allows you to blend in a small amount of another texture cluster in, increasing textural complexity. On the left, you see grass with no secondary texture, and the right, a secondary texture is blended into the grass area.

Once Secondary Cluster is true, the interface expands to show a second texture choice along with several parameters. The Noise Frequency controls the noise function used to decide where the second cluster will be blended. Higher noise values will produce more frequent but smaller patches, lower values will produce less frequent and larger patches. The noise range controls the minimum and maximum value of the blend. In this shot, the secondary texture will never fully blend, as it's maximum value is reduced. When this is combined with the height map blending MegaSplat uses, it means small areas of the taller stones will poke out from the moss

texture in these areas. Finally, a noise seed is also available, which offsets the position in the noise and changes where these patches show up.

Some Tips:

- A Two Layer shader is recommended for conversion from traditional texturing
- If you are generating a terrain in a procedural program, it is best to have wide and blurry transitions between your terrains. These might not look good on the regular shader, but will allow MegaSplat to do more interesting blending, making the terrains look much better.
- You can use blur to widen the terrain blends; however, this may make some textures disappear into more dominant textures near them. You can use the contrast setting to bias these back into the mix
- Picking the right interpolation contrast can do wonders to the quality of the conversion. Low interpolation factors will produce a softer blend, which can look more like the original terrain.

Render Baking

One optimization that's possible with MegaSplat is to cross fade into a Macro Texture for meshes or terrains which are far away. In some cases, it's desirable to render this texture from the painted MegaSplat data. MegaSplat includes a Render Baking tool, which will bake out the paint job on a series of meshes or terrains to a set of textures.

To use this feature, select the terrains or meshes you'd like to bake to a single set of textures. Note that if you are baking several meshes to one set of textures, then they must not have overlapping UVs, or the data rendered into that section of the texture will be whichever mesh was drawn last.

Open the Utilities panel in the Terrain or Vertex painting toolset. Select an output path and resolution. Then select which "features" you want to export and press Export Selected.

If the feature set is set to everything, then MegaSplat will render out a diffuse, normal, ambient occlusion, emissive, height, metallic, and roughness map for all the meshes in the selection (or for your current terrain). You can then pack these in your favorite image editing software and use them as macro textures.

Alpha Layer mode

The shader's Alpha Layer mode can be used to paint splat maps over regularly textured objects.

In this mode, the macro texture is used to texture the object, and the top splat layer's weight is used to blend in splat data. Painting on the bottom layer acts as an erase tool to remove splat mapping.

To set this up, create a new MegaSplat shader and material with the Layer Mode set to AlphaLayer, putting the original textures into the Macro Texture area. Then open the vertex painter, set the brush to the bottom layer, and press fill. This will make the object appear with no splat maps. Set the brush to the second layer, select a texture or cluster to paint with, and paint splat maps onto your mesh.

By default, the shader will use the same UV coordinates for both the macro texture and splat map data. However, in the shader options there is an option called to use a second UV set for your macro texture.

Troubles with Tangents

When working with multiple UV sets, keep in mind that the mesh only has one set of tangents available, and most modeling programs (as well as Unity's import settings) always generate those tangents based on the first set of UVs. In practical terms, what this means is that if you have mirrored and rotated UVs in your model, the tangent space will be mirrored or rotated for all UV channels in respect to normal mapped lighting. If your second set of UVs is not rotated and mirrored in the same manner, then it will display normal mapped lighting which appears flipped or discontinuous. By default, tangents come from the model, but there is an option when using projected UVs to generate tangents along with the projection.

Alternative to Alpha Layer mode

Another use case for texturing many objects in a scene using MegaSplat is to use the texture array system to avoid having multiple materials for many different objects. To use this on non-organic surfaces, like houses, you can pack many textures into a texture array and paint each sub-object with a different texture. The “disable bottom layer blend” option can then be used to turn off the extra samples and blending calculations needed for the splat mapping. Note that you can still use a two layer shader to blend in another texture on top of the traditional texture in a similar manner to how the Alpha Layer blend works. This can be a powerful technique which allows you to massively reduce draw calls across a scene.

Performance Considerations

Performance of shaders is a complex topic, but on most hardware these shaders will be bound by texture sampling and memory bandwidth cost, not arithmetic cost. However, all texture samples are not created equal. For instance, sampling a texture multiple times in a similar area can be much faster than sampling multiple textures. With that said, the Debug rollout on these shaders will give you an exact count of how many samples are done per pixel with the current set of options enabled. While not a perfect benchmark, and certainly not a linear one (12 samples is not 6 times slower than 2 samples, for instance) it is likely a reasonable guide to performance cost.

As an example, here is how sample counts are broken down, by feature, for several given shaders. Note that the standard unity terrain shader with 8 textures is going to draw two passes of the object, which has additional cost over just the texture sampling cost.

Shader	Unity Terrain Shader w/ 4 textures	Unity Terrain Shader w/ 8 textures	MegaSplat (Single Layer)	MegaSplat (Two Layer)

Diffuse Only	5 samples per pixel	10 samples per pixel	3 samples per pixel	6 samples per pixel
Diffuse + Normal	9 samples per pixel	18 samples per pixel	6 samples per pixel	12 samples per pixel
Diffuse + Normal + Specular + AO	13 samples per pixel	26 samples per pixel	9 samples	12 samples

Here we see that this technique can be more efficient than standard Unity terrains, with much better quality. However, MegaSplat supports a lot of techniques not available in Unity's terrain shader, or even in Unity's standard shader. It is quite easy to produce a shader which can have 30 or more texture samples per pixel.

Here's a breakdown of how some features effect sample counts:

- Parallax mapping
    - Requires another sampling of the diffuse textures, so 3 or 6 more samples, depending on which MegaSplat shader is used.
- Flow Mapping
    - Flow Mapping requires a straight doubling of all texture samples in the area where the flow map is active. Conditional branching is used in areas where the flow map is not active to reduce the number of samples on those pixels.
    - Flow mapping also creates a dependent texture read, which is slower as the GPU has to wait for one value to be read before it can compute the UVs for the next sample.
- Per Texture Properties
    - Because Unity 5.4 does not support array properties on materials, we store per texture properties in a 256x4 texture. Each pixel which must be sampled (depending on feature set) adds 3 extra samples (per layer). Though it is worth noting that due to the small size of the texture, this is likely much faster than other samples in the system.
    - However, because this texture read must happen before the arrays can be sampled, this can stall the GPU (this is known as a dependent texture read), so consider if you really need per-texture properties or not before enabling them.
- Triplanar Texturing
    - Triplanar texturing requires sampling once for each projection, so multiplies the total number of samples by 3. When using triplanar, you'll want to keep other features to a minimum
- Macro Texturing
    - Macro texturing adds 1-2 samples for the main texture based on your packing mode. However, if you crossfade to the macro texture completely the shader will

early out without doing any splat mapping. This allows you to have a distance texture which is extremely efficient.

- Tessellation
    - Tessellation can easily bring any GPU to its knees. Adjusting the parameters correctly is of vital importance. On some GPUs, the cost for having tessellation on, even if you don't actually generate more vertices, can be high - so it is recommended to switch to a fallback shader when the object is out of tessellation range.
    - Edge Length (when using edge mode) and the Tessellation properties control the maximum amount of tessellation. When edge length is low or Tessellation is high, more triangles are generated.
    - Adjusting the min/max distance for tessellation will reduce total tessellation in the distance when in Distance mode. However, these are only used to fade out the displacement when in edge mode, since in edge mode the screen space size of the edge is used to determine the tessellation factor.
- Resample Distance
    - Doubles the number of splat map samples; prefer Texture Clusters to defeat tiling over this technique.
- Low Poly Look
    - Low poly look uses the geometry shader to generate hard edged triangles. Using the geometry shader is generally slow, and in this case produces more vertices than the mesh originally had.
- Puddles
    - Simple puddles are very cheap on meshes, requiring no texture samples. On Terrains, a second control texture is sampled for data about where the puddles go. Puddle's with flow require 2 standard texture samples per pixel, so are still quote cheap. Puddles with foam and refraction cost additional texture samples.
- Snow
    - Snow requires 2 texture samples per pixel. When tessellation is enabled, those samples must also be done in the displacement stage as well.
- Per Texture Properties
    - Per texture properties are stored in a small 256x4 texture. While these are fast to sample due to the texture size, in some cases they can introduce a small dependent texture read, where the GPU must wait for these samples to complete before starting the next ones.

Note that the terrain shader performs an extra 3 samples for the control texture per layer and has to wait for those samples to finish to perform its next set of samples. This texture stores the information that would be imprinted into the vertices of the mesh shaders. Because of this, rendering terrains can be slower than rendering regular meshes. Using a smaller control texture may help, making it more likely that the needed texels are in the cache. Displacement Dampening, Flow Mapping or puddles also require sampling an additional control texture for their data.

Collision Response

MegaSplat includes components to help you determine which texture you've hit with a raycast, as well as offsets for working with physics and tessellated/displaced surfaces. When you press play in the example scenes, you'll see the texture you are currently above displayed in the top right of the screen.

To set this up, select your diffuse texture array and turn on "Export Texture List". This will export a scriptable object with a list of texture names to be used at runtime.

Then, on any terrain or mesh you want this information from, add the MegaSplat Collision Info component, and assign the texture list to the Texture List property of the component. If your shader is a two layer shader, turn on two layer, and the most dominant texture will be the one returned.

Finally, you will need to turn on 'Keep Runtime Data' on the Vertex Instance Stream. This will ensure that the mesh data is kept on the CPU during gameplay.

The camera has a "DisplayTextureName" component on it, which is a good example of how to use the system to extract the texture information from a raycast or collision.

Physics and Tessellation

Tessellation with displacement mapping looks amazing; but there is an issue that forced many developers to dial it way back, and that's physics integration. The most obvious solution to this would be to tessellate and displace the physics mesh as well- however, this would quickly make physics meshes extremely large and unmanageable. So MegaSplat includes a unique set of tools that allow you to integrate physics with displacement mapping to get accurate results, as well as simple examples of those tools in use.

The basic idea is to keep a low res texture around for each surface and perform the displacement for a collision point on the CPU. If you set the Physics Data Size on your diffuse texture array, MegaSplat will generate this data for you automatically. For most needs, a data size of 32 (pixels) is adequate, though larger sizes are available if you need to have things crawl through small cracks in the displacement data.

Static Meshes

For the vertex painter to function, it needs to keep a copy of the mesh on the CPU via the read-write flags in the mesh importer. When the game starts, it applies the changes to the meshes via the additionalVertexStream feature. However, this is not compatible with Static batching and some lighting features which require meshes be marked Static. To fix this, there are tools in the Vertex Painter's Utility panel that allow you to bake out copies of your meshes with the vertex modifications baked into them. However, doing this one mesh at a time can be tedious if you paint on many meshes in the scene. There is also a tool called "Bake Scene to Mesh Cache", which will perform this operation on an entire scene, saving all meshes which

have been painted to an asset cache folder, and replacing the ones used in your scene with them.

While you can use this manually, an ideal workflow is to insert this into your build machines build steps. Before building and baking lightmaps, execute the SaveMeshCache.Bake function, passing in the relevant information, save your scene, then compute lighting and build your project and revert your scene changes using your revision control system.

Using the Vertex Painting toolset

The vertex painting system included with MegaSplat is a full featured toolset, capable of painting vertex colors, flow directions, editing mesh geometry, baking ambient occlusion or lighting information into vertices, and much more. More detailed documentation about its use, along with video tutorials, can be found on my GitHub account at:

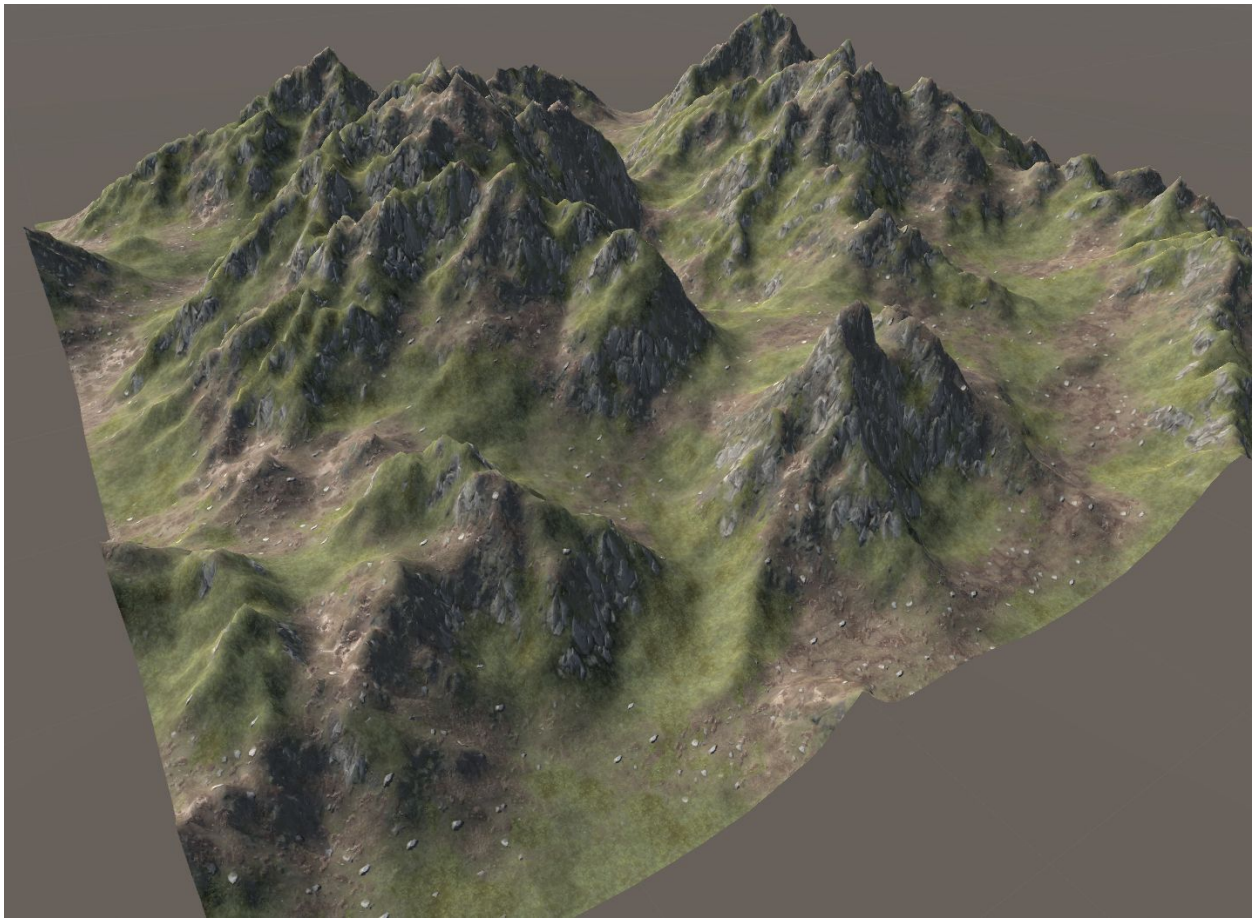
https://github.com/slipster216/VertexPaint

There are additional tools and custom brushes also available, for doing things like blending normals between objects and baking various information into vertex data.

Terrain Painting

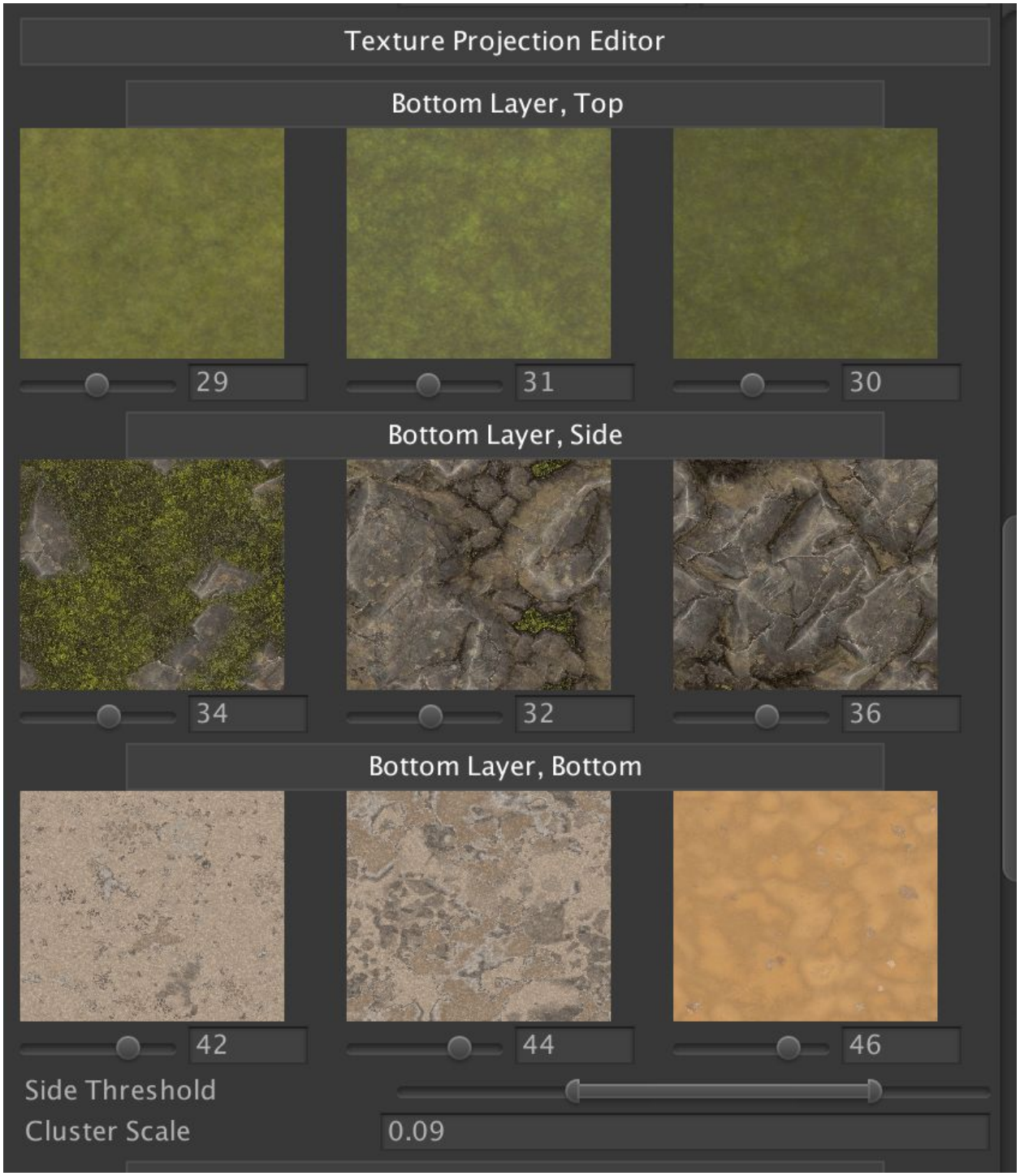
The terrain painting system is based off the same code and workflow principals used in the vertex painter. However, it only paints on Unity terrains, and doesn't add any additional components to the scene, as the data is stored in textures.

Procedural Texturing



An example mesh, textured procedurally, with no painting, at runtime

When working with Meshes, a runtime procedural texturing system is available which can texture your terrain based on rules, rather than painting. When a layers texturing mode is set to Project 3 Way, the procedural texturing interface is shown. This mode allows you to texture a layer, or multiple layers, entirely on the GPU without having to paint your mesh. This can be very useful as a way to quickly paint large terrains, or as a way to paint the terrain in scenarios where painting is not an option (at runtime).



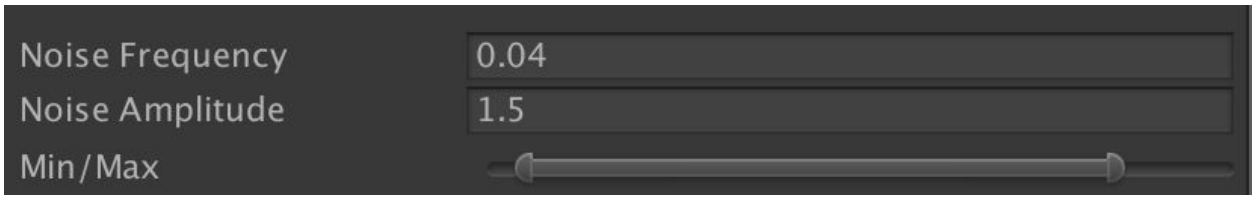
The basic idea is similar to other triplanar projection shaders- a texture is applied to the top, side, and bottom of geometry automatically. However, as MegaSplat works differently than traditional shaders, a custom version of this technique is used which takes advantage of MegaSplat's unique abilities.

For each projection (top, side, bottom), you can choose 3 textures which are blended together in a manner similar to using Texture Clusters. It is important to get the cluster scale

value set correctly - values which are too high will produce noisy patterns, and values which are too low won't show decent variation. Finally, you can use the threshold slide to determine which angles are considered to be the 'side' projection.

This is very powerful, because it lets you filter your selection of top/side/bottom into three arbitrary angles. You could, for instance, use it to texture all geometry facing up to be the top, all geometry with a slight slope to be the side projection, and all geometry facing sideways or down to be the bottom projection.

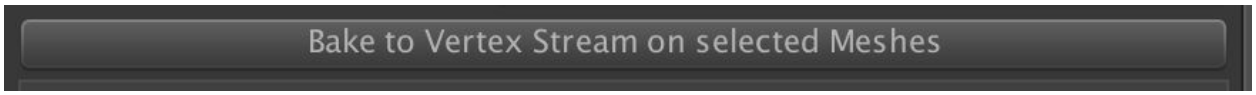
If you are using a two layer shader and the second layer is set to use a 3 way projection, then an additional set of texturing choices is available, and a set of blending choices for the two layers:



The noise frequency, amplitude, and min/max can be used to control the blending between two procedurally layered textures. Again, getting the noise scale correct is the key to this looking amazing. Adjusting the amplitude will create a quicker blend between the two layers, or negate the blending when moved below 0.

The min/max slider can be used to control the minimum and maximum amount of blending between the two layers.

So, rather than simply projecting 3 textures, one from the top/side/bottom and blending them, MegaSplat allows you to project 3 arbitrary projections on two layers, with a total of 19 textures being blended together to create your terrain. This provides a ton more variety, defeats texture tiling, and makes for a much more visually interesting scene.

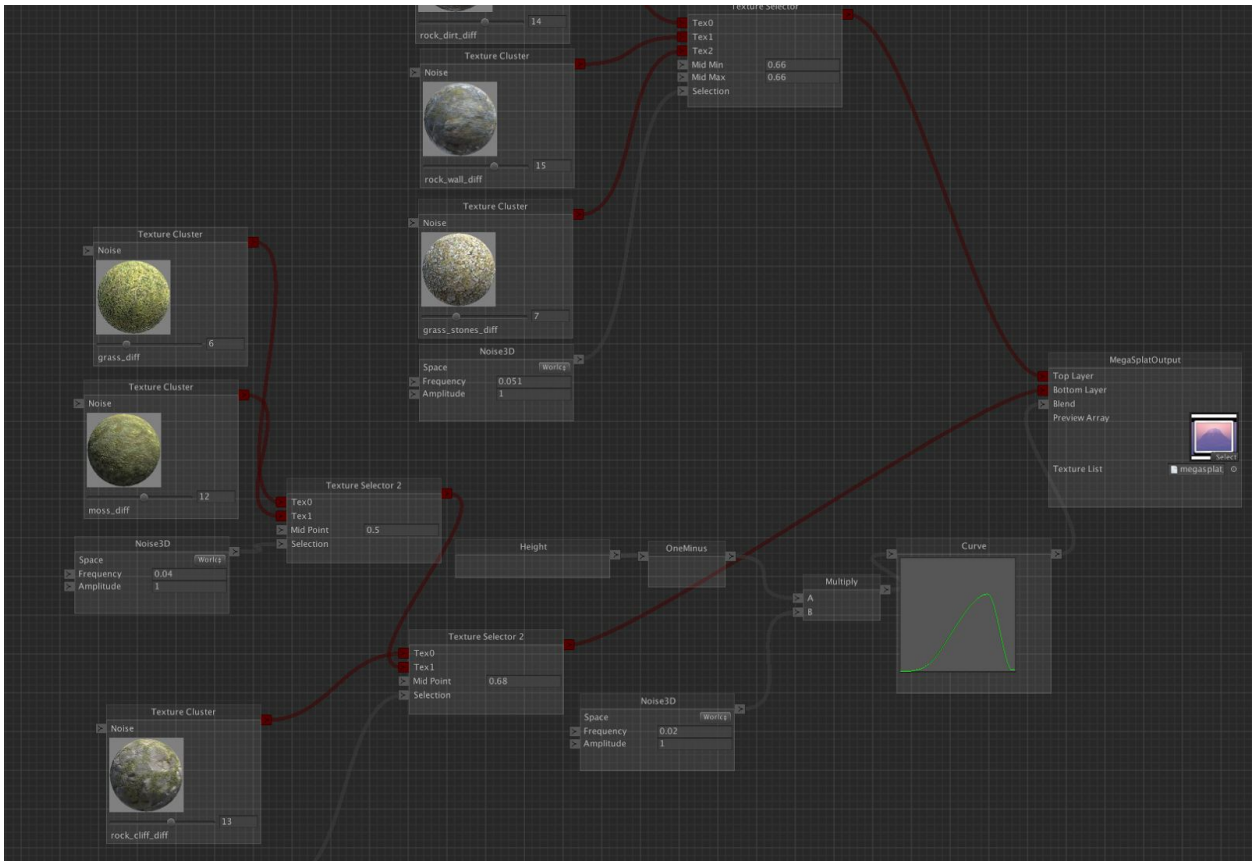


Finally, at the bottom of the interface you will find a "Bake to Vertex Stream on selected Meshes button. When pressed, any meshes in the current selection which are setup to use the material and are setup to be vertex painted will have the procedural texturing burned into the vertices. At this point, you can switch the shader back to the painted mode and the texturing will remain, allowing you to further modify the mesh via the painting interface.

This allows you to use the runtime procedural texturing as a way to apply a quick base coat to a mesh before painting. A neat trick is to keep a copy of the material/shader around for such uses, apply it to new meshes, bake the texturing back into the mesh, then switch to the version of the shader with procedural texturing turned off since the data is now baked into the vertices, saving the vertex calculations required by the procedural technique.

A demo of this feature is done in MegaSplat Dev Log 15 in my YouTube channel.

Procedural Texture Graph



Essentially, it's a domain specific shader graph for creating procedural terrain texturing systems with MegaSplat.

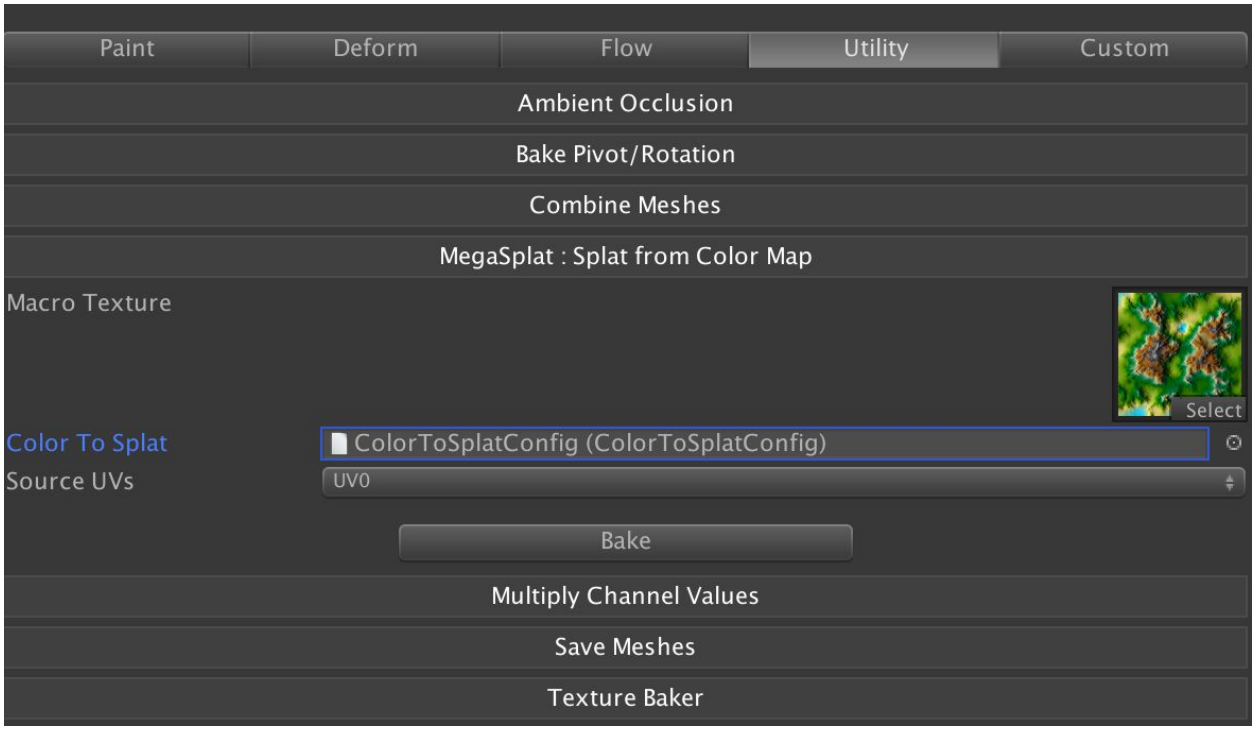
Unlike a traditional shader graph, where you have inputs to the lighting equation on the master node, the master node contains inputs for a texture choice for each MegaSplat layer and a blend value between them. This format is discussed below.

Nodes in the graph work with one of two data types; Texture Indexes (red), or float values (grey). The 3d noise, normal angle, and height nodes will output their result as a 0 to 1 value. Selectors can be used to select between several texture indices based on one of these values. Texture Clusters are also available for use in the graph, and you can easily remap float values with curve controls.

If you want to experiment with this feature, I suggest watching MegaSplat Dev Log 16 on my YouTube channel, where I walk through creating a graph and texturing a terrain with it. Additional documentation is included next to this document in the MegaSplat folder.

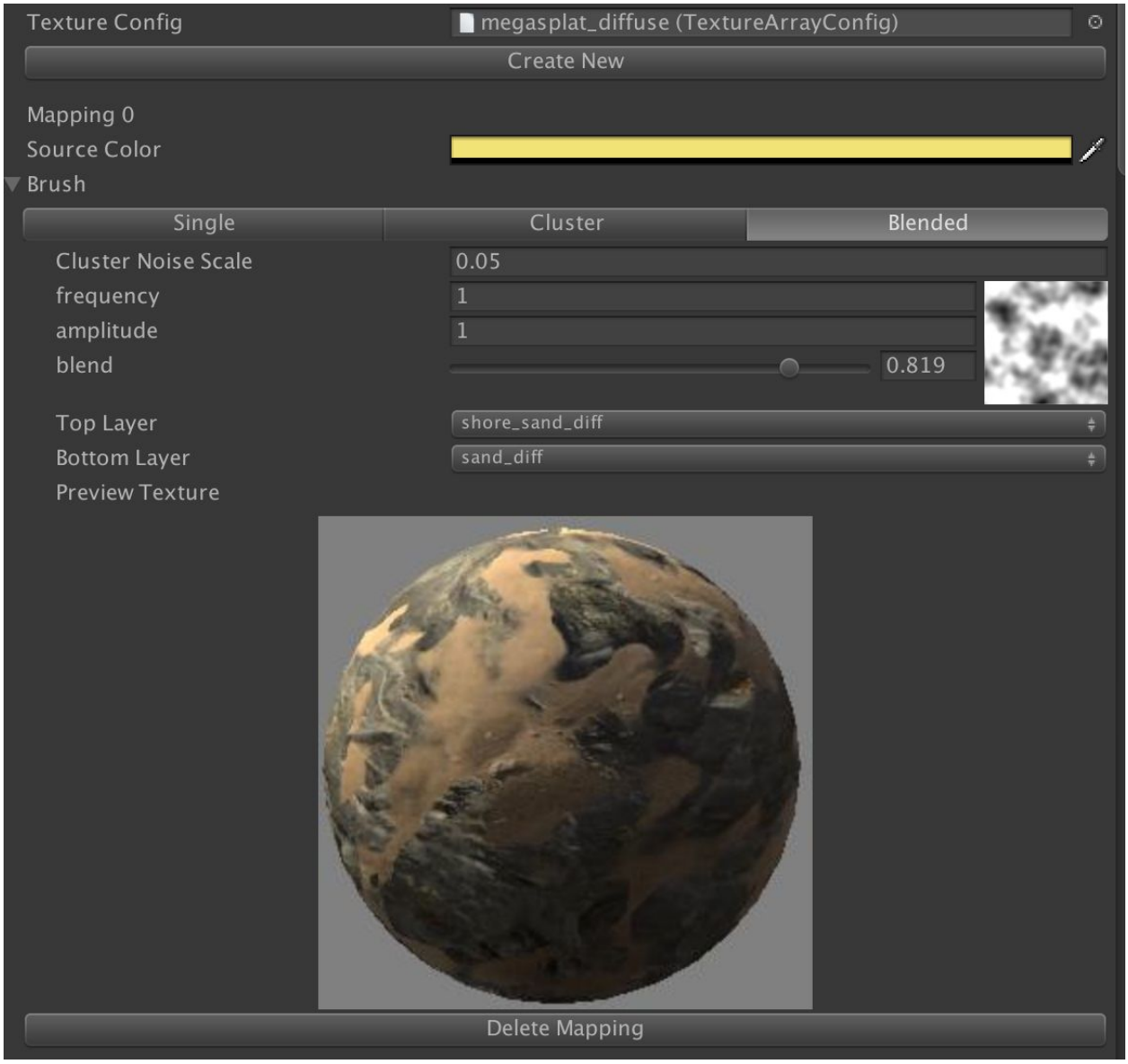
Texturing large terrains from external tools

Many landscape generation programs will output some sort of guide image or diffuse map for your terrain. You can use these images to rapidly apply texture across an entire landscape at once using the Splat from Color Map tool in the Utilities panel.



This tool takes a color image and a configuration file. An example is included with the package, and can be used with the t03_guide image found in the Examples/Terrain folder.

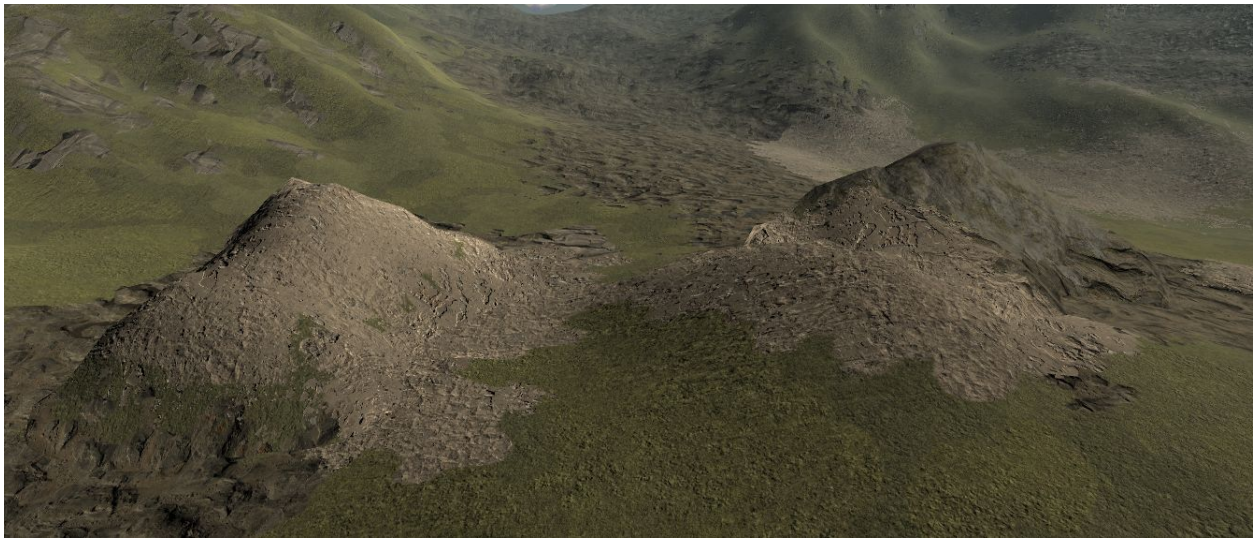
You can create your own config by right clicking in the project window a selecting Create/MegaSplat/ColorToSplatConfig.



This is a single mapping within this config. You may have as many mappings as you need. Each mapping is from a source color in your image to a Brush which will be applied over that area of the image. In this example, we are using a blended brush which is going to apply a texture from the shore_sand Texture Cluster to the top layer, and a texture from the sand Texture Cluster to the bottom layer. A noise function is used to decide between these two clusters, and a preview sphere is updated in realtime as you adjust the parameters.

You may also create mappings using indexes stored in textures, or a traditional splat texture image.

Object Blending



An example of blending objects into a mesh based terrain is included in the examples directory. If you zoom into the boulder objects, you'll see that they seamlessly blend into the terrain. This is done by:

- Ensuring that the UV coordinates used for Splat Mapping are the same in the blend region
- Ensure that the painting is the same on both objects.
- Ensuring that the normals and tangents in the blend region align.

In this example, the first requirement is met by using a world space triplanar projection for the UV coordinates on both objects.

For the second requirement, simply select both objects and paint over the intersection with the same megasplat cluster or texture.

For the third requirement, a Normal Blend Brush is available in the Vertex painter. To use it, go to the custom tab and select the NormalBlendBrush as the custom brush (the same way you select the brush for MegaSplat). Select the boulder for painting, and the terrain mesh below it as the "Target object" in the brush options. Then paint along the intersection, and the normals and tangents will adjust to face a value similar to the target mesh. Note that since the vertex positions will rarely line up between the two objects, the normals and tangents may be slightly different- However, the normal/tangent which are chosen to align to are based on the interpolated normal at a given position, so you can usually adjust your angle slightly until they match up well enough.

Transfering Paint Jobs

Sometimes you spend a bunch of time painting a mesh to get everything looking great, but then realize you really need to modify that mesh in some way. This can be problematic, since the data is burned onto the mesh. However, MegaSplat includes two utilities which can help solve these issues.

The first is the Render Baker, which was covered earlier in this doc. It can render bake out a “SplatData” texture, which is in the same format as the terrain system uses. Note that for this to work correctly, you’re meshes must not have overlapping UV coordinates.

The second utility is the Splat Data to Vertex Data tool found under the Utilities tab in the Vertex Painter. This can read the Splat Data texture and apply it to the vertex data. It can also be used to transfer the paint jobs from Unity Terrains to meshes.

Using these two tools, it’s possible to save off a paint job, modify mesh geometry, and reapply that paint job. An example of this being done is included in MegaSplat Dev Log 16 on my YouTube channel.

Integration with other vertex painters or procedural geometry

MegaSplat’s shaders can be used with other vertex painters or procedural geometry. The data layout for the vertex information is as follows:

Color RGB = Vertex Weights
Color Alpha = index of bottom texture layer (1/255 increments)
UV1.w = Wetness
UV2.zw = Flow Direction
UV3.X = Blend between first and second layer (layer shader only)
UV3.Y = Displacement Dampening when using Tessellation
UV3.Z = Puddle amount
UV3.W = index of top texture layer (1/255 increments, layer shader only)

The data layout of the control texture (for use with Unity Terrains) is:

Color.R = index of bottom layer (1/255 increments)
Color.G = index of top layer (1/255 increments)
Color.B = blend between layers
Color.A = displacement dampening (1 = no dampening, 0 = maximum dampening)

A second params texture is sampled when features are enabled that need it. It’s layout is:

Color.RG = flow direction
Color.B = wetness (0 = no wetness, 1 = saturated)

Color.A = puddles (0 = no puddles, 1 = fully submerged)

If you would like to apply the texture at runtime, simply adjust the vertex color or UV channel as appropriate, or adjust the pixels of the control texture when working with Unity Terrains.

If you are generating procedural geometry, then you will need to make that geometry fit the requirements of the shader. Each face must have one red, one green, and one blue vertex, with each face only having one of each color. Doing this may require creating extra vertices, but for most geometries the number of vertices added is quite small.

It does not matter which color each vertex is, as long as each vertex on a given face only has one of each color present. These weights are used to filter the texture index data, allowing each face to know the three textures it must sample for a given face.

Painting at Runtime

Example components are included and used in the demo scenes to show how to efficiently paint at runtime. These scripts are extensively commented, especially on the mesh workflows, as caching and unrolling the math used can provide an order of magnitude speedup over the simpler version. I suggest you use them as a template for runtime painting.

For them to work with Meshes, you must turn on the “Keep runtime data” property on the VertexInstanceStream of the mesh. Terrains have to jump through a few hoops to restore the texture data to its non-play mode state while in editor mode. This is also documented in the components.

Several of the example scenes feature balls rolling down the terrain, some of which will paint a given texture onto the terrain.

Extending MegaSplat shaders with your own code

Comfortable with writing shader code? When MegaSplat generates a shader, it does so from a number of fragment files, which can be easily modified. However, if you were to upgrade MegaSplat, these changes would get lost. So MegaSplat provides a way for you to write custom code which will be included at specific points of the shader, allowing you to customize the way MegaSplat works in a non-destructive workflow.

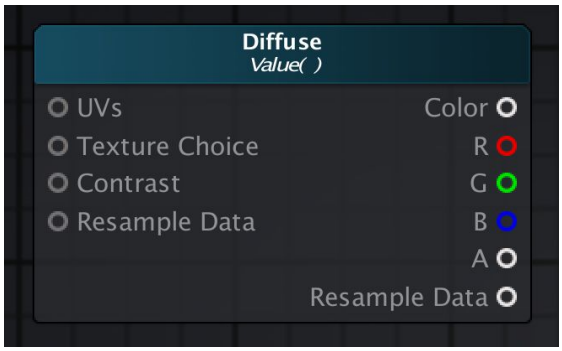
The custom code option is described in Development Log #20 on YouTube. When you turn on the Custom Code option, MegaSplat will write out a .prop and .cginc file for you to customize. These files must remain next to the MegaSplat shader and material, and all MegaSplat shaders in the directory with the custom code option will use the same .props and .cginc files, making it easy to have several shaders which share the same modifications.

The .props file can be used to declare new material properties. These must be prefixed with the name _Custom_ to show up in the MegaSplat shader editor. The .cginc file which is written contains several functions, along with documentation on those functions and the structures they use. You can customize these functions to do things like modify the vertex

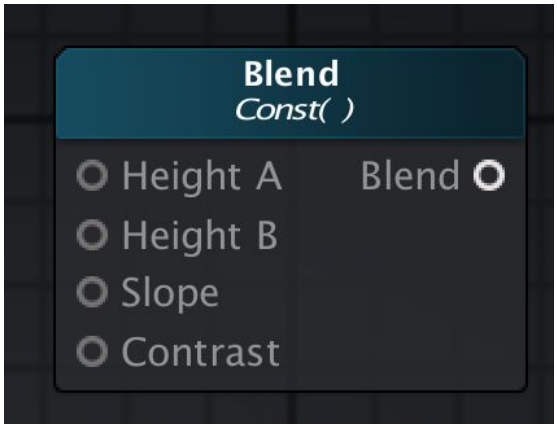
positions, modify the inputs to the lighting equation (Albedo, Normal, etc), and adjust the tessellation/displacement results. Please consult the .cginc file and youtube video for more information.

Amplify Shader Editor integration

Two nodes have been included for the Amplify Shader Editor that allow you to roll your own shaders using the MegaSplat technique.



The MegaSplatSamplerNode works like a texture sampler, but takes a texture array instead of a texture. It also takes a Texture Choice parameter (stored in color.a or UV3.w in MegaSplat's vertex layout) and contrast used for the blending. The node currently expects the height map to be in the alpha channel, performs the blending, and outputs a final color along with "Resample Data", which can be plugged into a second MegaSplatSamplerNode in order to sample a normal map or additional maps using the same resulting blend.



The MegaSplat blend mode allows you to blend multiple layers of megasplat data together, as seen in the MegaSplat two layer shader. You would plug the resulting height value from the Sampler Node of each layer into the height (A/B) inputs, and a blend weight (painted into UV3.x by MegaSplat's brushes) into the slope value.

Using these two nodes you can create your own MegaSplat style shaders.

Extras

The Vertex Painting toolset ships with a set of traditional splat map shaders that allow for up to 5 layers of splat mapping with flow mapping. Because the shaders take several minutes to import, they have been archived in a zip file, along with other examples for the Vertex Painter package.

A note about OSX support and the Metal API from Apple

The situation with graphics API on OSX are a bit of a mess. For maximum compatibility, I recommend using OpenGL until these issues are sorted out by Apple and Unity. Note that as of Unity 5.6, Metal is the default API for builds in OSX, and from 2017.1 onwards, they expect to make Metal the default API for the editor. However, you can switch both the build and editor settings to use OpenGL.

Apple has decided to not support the use of Geometry shaders on Metal. This is because Geometry shaders are really slow, and should generally be avoided. MegaSplat uses geometry shaders for the “Low Poly Look” and “Preprocess in Shader” options, so these will not run on OSX when using the Metal API.

Tessellation is currently not available on Metal in Unity- so tessellation shaders will also not run when using the Metal API on OSX.

Some of MegaSplat’s features, like Snow and Wetness, use too many interpolators to run under shader model 3.5. When these features are enabled, MegaSplat will increase the shader model to shader model 4.0. However, since using shader model 4.0 also requires geometry shaders to be present, these features break when enabled while running under Metal. You can use the shader override level to bump the shader model to 4.5 and the shaders will run under Metal, but then the shaders will no longer run under OpenGL on OSX, which doesn’t support shader model 4.5.

So, like I said, it’s a mess, and I’d recommend disabling Metal in OSX for editor and build usage for maximum compatibility. However, if you want to use Metal in OSX, you can generate a shader fallback chain by creating multiple versions of each shader (4.5 for metal, 4.0 for openGL).

Troubleshooting and Faq

Q: The splat textures are all black in the example scene, what’s wrong?
A: Is graphics emulation on? Unity often switches my Mac to emulate openGLES2.0 devices, on these devices, the splat map data is always returned as black because texture arrays are not supported. I have also added debugging colors to unsupported shader types on device.

SHADER_API_D3D9 : returns green
SHADER_API_GLES (openGLES2.0) : returns yellow
SHADER_API_D3D11_9X : returns aqua blue

Unknown : Returns blue

Further, make sure your editor is not running in a DX9 or DX9_11 mode, which is listed at the top of the editor window.

Q: I can’t see the flow maps moving
A: Make sure Animated Materials is turned on, and that you have painted flow directions into the UV2.ZW channels. Make sure the flow speed and intensity are up, and then select the texture in the per-texture properties and make sure it’s speed is also turned up. Finally, if you are using the 2 layer shader, remember that the flow map is only applied on the top layer, so it must be visible to see the effect.

Q: The textures are being interpolated across the face - I see like 20 textures on one triangle.
A: You need to preprocess your mesh with the converter into the format used by MegaSplat. If you are procedurally generating geometry, then you will need to correctly mark the vertex colors so that each face has exactly one red, one green, and one blue value. If you are using the meshes, it’s also possible you accidently painted into the color channels- MegaSplat uses data in the color channels of the vertices to filter texture choices, so if that data gets accidentally changed, it can cause this problem.

Q: When I press play, I get an error that looks like this and the mesh looks like it’s not painted!

Not allowed to access colors on mesh 'Combined Mesh (root: scene) 2'
UnityEngine.Mesh:get_colors()
JBooth.VertexPainterPro.VertexInstanceStream:EnforceOriginalMeshHasColors(Mesh) (at Assets/MegaSplat/VertexPaint/VertexInstanceStream.cs:128)

A: The import settings on your mesh are not set to Read/Write, or you have set the mesh to be static. This means the CPU does not have access to the vertex data, and cannot apply the paint job. If you need your mesh to be statically batched, or need to turn off read/write for some reason, then you can bake out a copy of the finished mesh using the baking options in the Utility panel of the vertex tool.

Q: I painted the terrain, but when I reloaded my scene it was gone.
A: Unlike the vertex painter, it’s required for you to press save in the terrain painter, otherwise changes won’t be saved.

Q: My terrain/mesh is white
A: Assign the texture arrays to your materials in the SplatData section of the material editor

Q: The previews on my textures in the painting tools are very dark!

A: This is a bug in Unity's Editor code when dealing with linear space, and I have yet to find a good way around it.

Special Thanks:

- Henrik Holmdat for information on Minimal Vertex Coloring
- Seneral for his open source node editor framework (used by the Texture Graph). Get it here: https://github.com/Seneral/Node_Editor