MicroVerse

Documentation

All support is handled via discord, and access to the GitHub repositories can be acquired by registering MicroVerse modules with our GitHub bot.

Discord Group

If you own the Core Collection, a demo package is also installed in your package area. If you have bought the MicroVerse, Splines, and Vegetation module independently, you can download that demo from here:

Demo Github Repository

Installing Splines on 2021

Splines are not available in 2021.3LTS by default, and are required for the MicroVerse-Spline module. To Install them, open the package manager and press the + button on the top left corner of the interface. Select “Add Package by name” and type com.unity.splines into the box. After installing, make sure the Splines package is version 2.0.0 or greater.

Intro

MicroVerse is a set of scene creation tools which operate entirely in real time. You won’t have to wait while things bake, or look through small preview windows to see your work. Every change happens as you make it, directly on the scene.

MicroVerse also embraces a non-destructive workflow. Want to move a mountain or road, just move it. Want to paint over an area with Unity’s paint tools to customize it? No problem. Want to move a whole town including its terrain and texturing? Yup. Want to prefab any aspect of a terrain as a biome and simply drag it onto a new terrain? Trivial.

Core Concepts

MicroVerse works off a stamp based system.

- Stamps can affect a specific area, or in some cases everywhere.
- A stamp might represent placing a mountain, putting textures on slopes, spawning trees, or preventing trees from spawning in an area.
- Stamps can be filtered by the results of stamps before them. For instance, a texturing stamp can filter based on the height, slope, angle, curvature, or flow of the height stamps computed before it.
- Stamps are applied in output order (all heights, then texturing, then tree's, then details), and then by their **order in the hierarchy**.

Setup

If you are trying to work with an existing terrain, please read the section on how to do that below before attempting it or you could erase your terrain data. If starting from scratch, follow these directions:

Add the MicroVerse script to an object in a new scene, or create it automatically from the GameObject/MicroVerse/Create MicroVerse menu item. Create new terrain(s), and parent them to the MicroVerse object. Only one MicroVerse can be added to a scene, and it only works with stamps and terrains below it in the hierarchy.

The Height Stamp

The height stamp moves mountains, literally. Once created, you can set a stamp image and it will deform the terrain based on its position, rotation and size. The height maps used should be saved as 16bit format files and set in the import settings as R16 files.

You can create a height stamp by right clicking in the hierarchy and selecting Create->MicroVerse->Create Height Stamp. Select a heightmap texture from the Examples/Common/HeightStamps folder and drag it into the Stamp slot on the component. You can move the stamp around and scale or rotate it to change the effect.

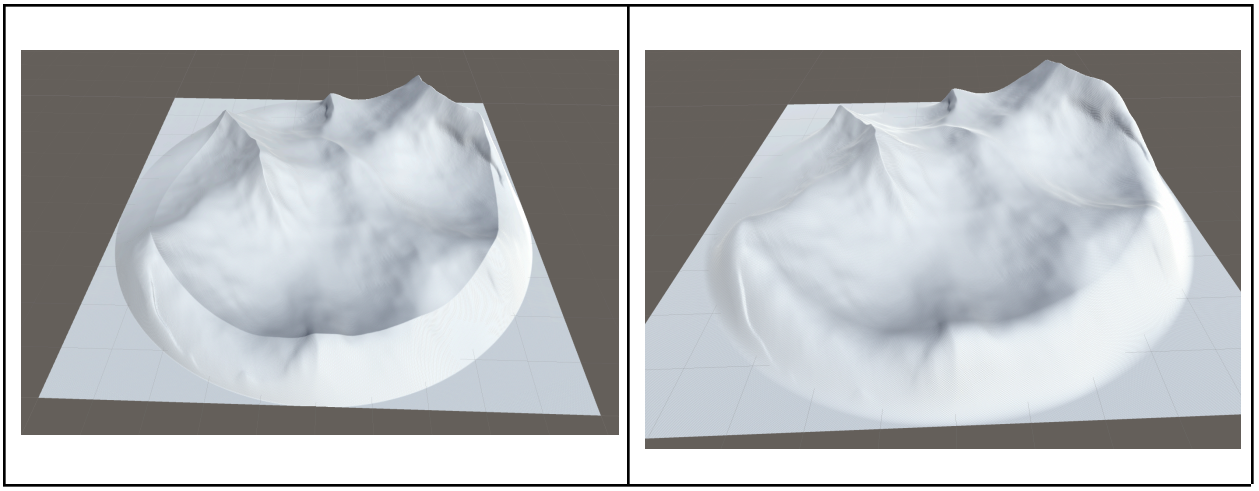
` Height stamps have a blend mode, which is “Max” by default. When set to Max mode, the maximum value of any stamps in an area is taken for the final height. Other modes include:

- Min
  - Takes the minimum value of the stamps in the are
- Add
  - Adds the current stamp value to the ones before it
- Subtract
  - Subtract the current stamp value from any ones before it
- Multiply
  - Multiply the stamp value with the previous stamps result
- Override
  - Set the value to the absolute value of the stamp
- Blend
  - Blend the current stamp towards the existing data, with a slider to control how much

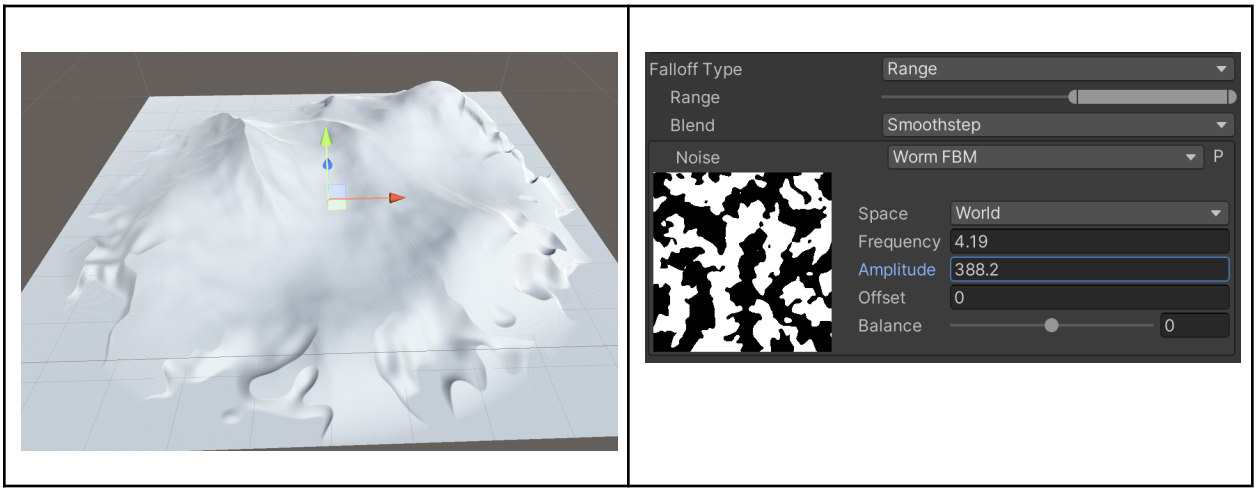
Understanding Falloff:

Most stamps have a falloff control. The falloff controls what happens at the edges of the stamp. Some stamps can work globally (like texturing), where others are clamped to the edge of the stamp’s bounding box.

When set to Range, a circular area with ranges ramps the effect down as it approaches the edge of the stamp.



On the left, the blend is set to linear, on the right, it's set to smoothstep, which puts a slight s-curve onto the interpolation, softening the edges of the falloff.

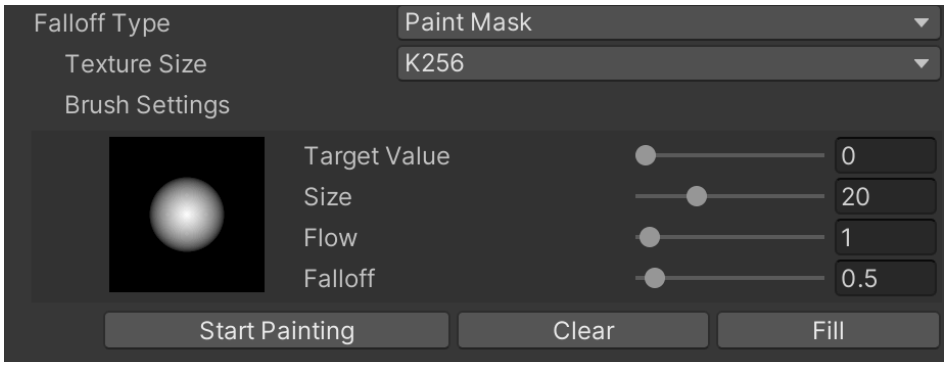


Falloff Noise is also available. Noise will be discussed more below, but this allows you to use texture based or procedural noise to break up the edge of the falloff area.

You can also use a texture to define the falloff, and when the Spline module is installed you can create arbitrary areas for stamp effects using splines, which will be discussed more later..

The falloff system makes biomes intrinsic to MicroVerse, since you can define the range of any stamp in the system.

Falloff Type : Paint Mask



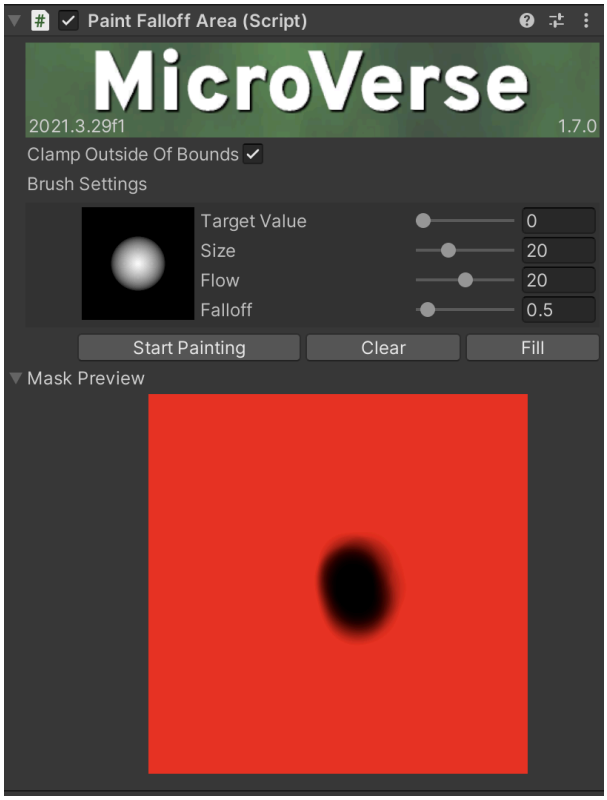
When Falloff Type is set to Paint Mask, a small painting window is available which allows you to paint the mask right in the scene view. A mask value of 1 has no effects, so painting with Target value of 0 will filter out the effect on the stamp in that area. You can control the size, flow, and falloff (shape) of the brush and see a preview of what the brush looks like. To start or stop painting, press the Start (or Stop) painting button. Clear will set the whole mask to 1, and fill will set it to the Target Value.

You can also control the size of the texture used to store this data. When resizing the texture, your existing data will be copied to the new texture.

Note that this painter is available for any type of stamp with falloffs, so you can paint the weight of tree's or textures in an area as well.

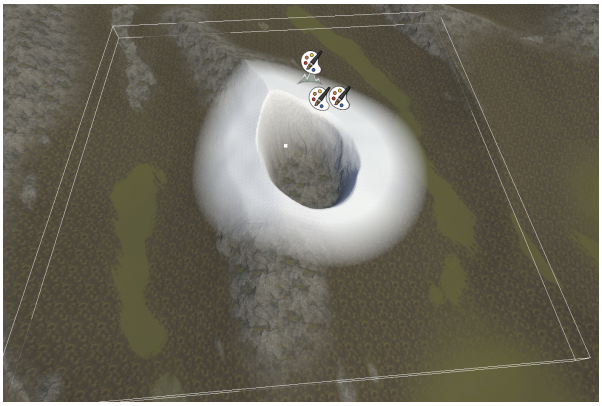
Falloff Paint Area

In addition to being able to paint the falloff for a single stamp, you can also paint the falloff for many stamps at once. This can be done through the Falloff Paint Area stamp. You can create and size one of these in the scene, paint on it, and have individual stamps or collections of stamps use the same Falloff Paint Area. Note that this falloff applies over the existing falloff's on the stamps rather than replacing them, which makes it easy to adjust a series of pre-created and blended stamps.

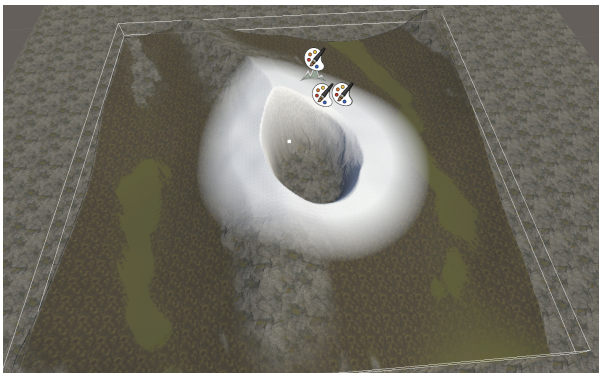


Here we see the same interface we have for painting individual stamp falloffs, but for a Paint Falloff Area.

Note the additional “Clamp Outside Of Bounds” option, this lets the stamp know how stamps using the area should behave when outside of the bounds of the Paint Falloff Area stamp.

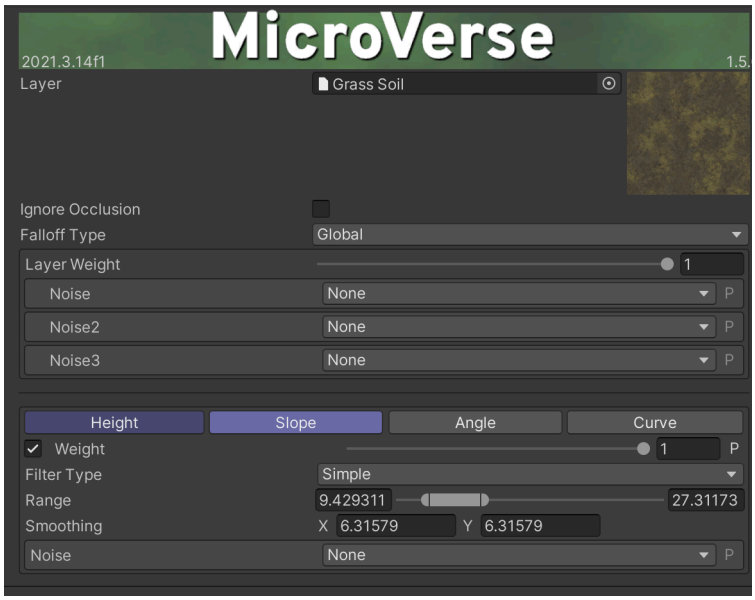


Here the paint mask is unclamped, with the stamps set to global, and we can see how the stamps effects spread outside of the area, but are effected by it within the area.



Here we see the mask is clamping, forcing the texture and height stamps to not have an effect outside of their area.

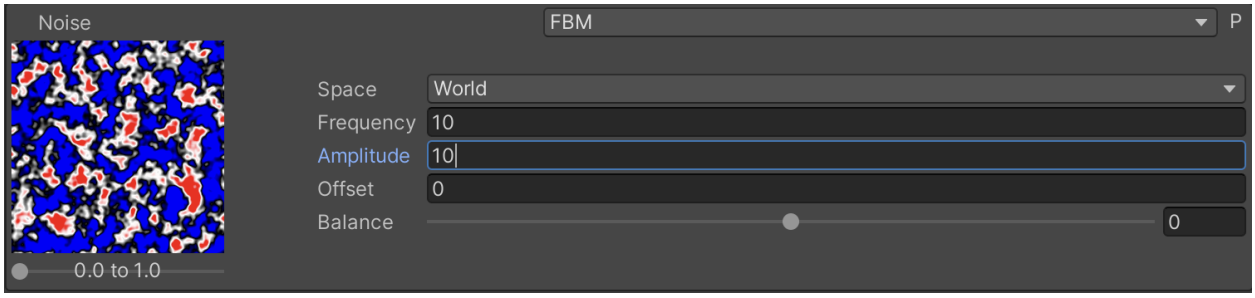
Texture Stamp



The texture stamp applies textures to your terrains. Textures are applied in hierarchy order, with stamps lower in the hierarchy being applied over previous stamps. You can assign a terrain layer file to the layer property to select a texture, and MicroVerse will automatically add only the textures needed for a specific terrain to the terrains that use it.

The weight of each layer is determined by a number of controls. First is the overall layer weight, which can also have up to 3 noise functions applied to it.

Understanding Noise:



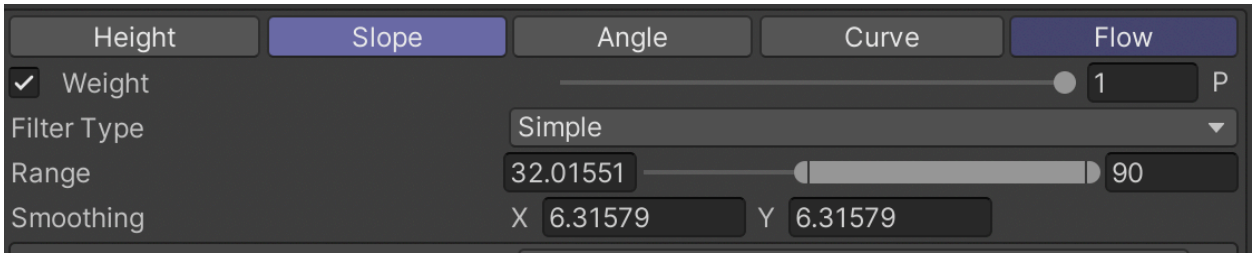
Noise options appear in many places in MicroVerse. You can select the type of noise - Simple, FBM, Worley, Worm, WormFBM, and Texture. Next to the noise type is a “P” preview button, which will show you what the noise looks like on the terrain.

The inline preview in the editor shows you a sample of the noise, with blue coloration for values which are less than 0 and red coloration for values over 1. You can use the slider below the preview to adjust the range of the visualization, such that you can determine the range of the values.

Noise can be in World or Stamp space. In world space, the noise is based on the world position. In stamp space, it is based on the position of the stamp. Another way to think about this is when you move a stamp in world space, it moves through a consistent noise across the terrain. When in stamp space, the noise moves and rotates with the stamp.

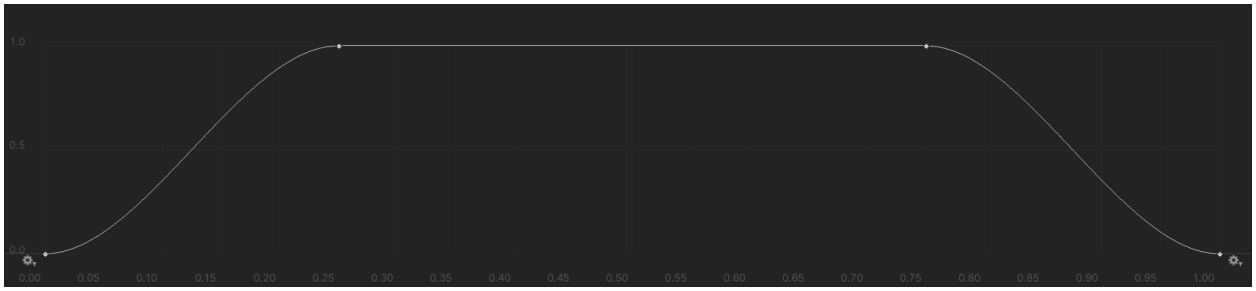
You can use the frequency to change the size of the noise pattern, amplitude to control the contrast, offset to reposition the noise, and balance to change the center point of the noise, adjusting it towards white or black. Note that a negative amplitude will invert the noise, and that noise values can become greater than 1 or less than 0. When set to texture mode, you can supply your own texture and set rotation, scale and offset as well.

Filters

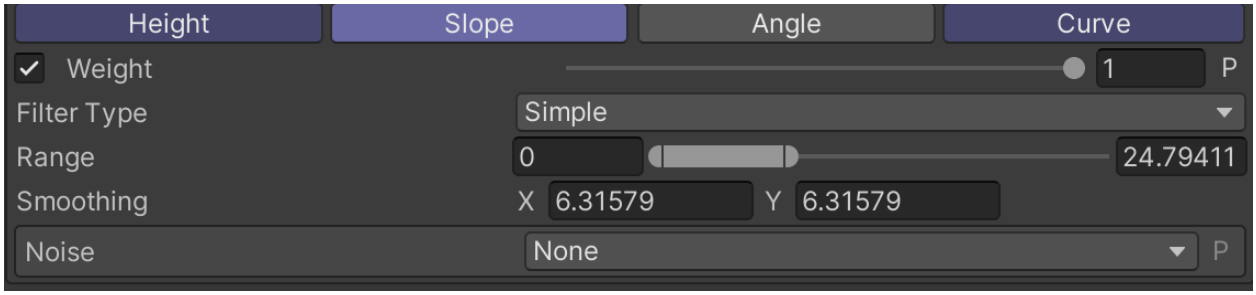


Filters are a great way to place content contextually based on the height, slope, angle, curvature, or flow of the terrain. Like noise, there is a preview button to show the effect on that particular filter. When set to type Simple, each filter has range and smoothing parameters. For instance, in the example above the texture is going to show from 30 units above 0 until 60 units above 0 at full opacity, and will fade in and out over 20 units on either side. So at 10 units this texture will not show up and will fade in fully by 30 units, then begin to fade out at 60 units until finally being completely faded at 80 meters.

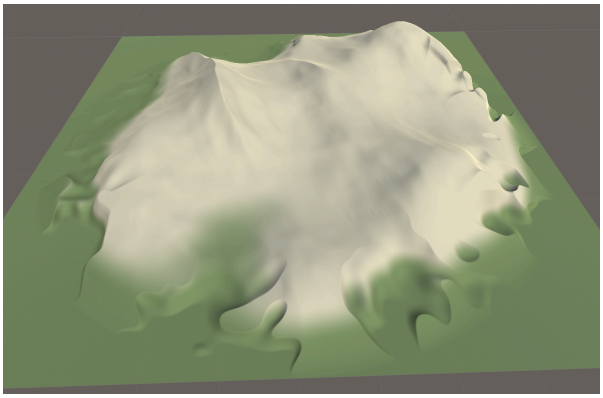
In most cases, simple filters are enough and very easy to work with. However, alternatively you can set Filter Type to Curve, which will give you an animation curve to describe the filter. This allows for much more complex filters, but can be harder to understand.



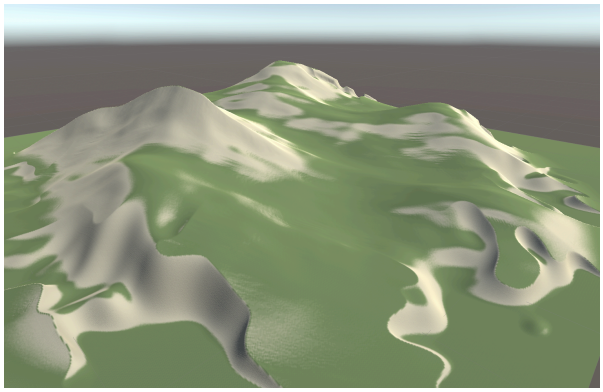
In curve mode, clicking on the curve will open up Unity's curve editing window, where you can edit the curve. Curves are all in 0-1 space, so in this case of a height filter, the Y axis represents the weight of the texture, and the X axis represents the height field (again, 0-1). If your terrain is 600 meters in height, then this curve will ramp the texture weight from 0-1 over the first 150 meters, then remain at full value until 450 meters, then ramp back down to 0.



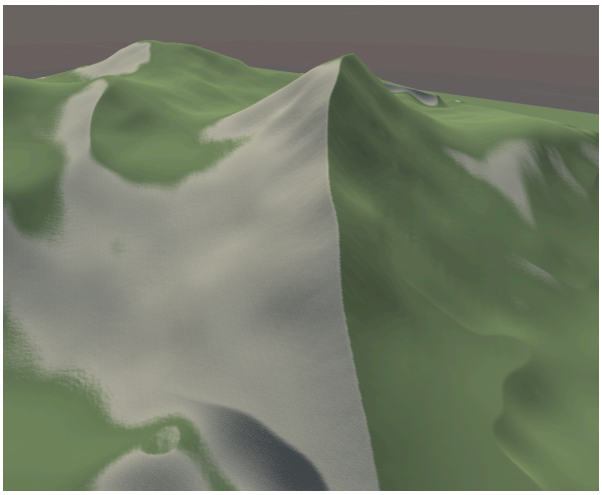
Slope Filtering allows you to texture based on the slope of the terrain, while angle is based on the facing. Curvature is based on the concavity or convexity of the terrain. Curvature values go from 0-1, with the lower half of the range (0-0.5) representing concave areas and (0.5-1) representing convex areas. Flow runs a simulation of water across the surface and generates areas where the deposits of matter would happen.



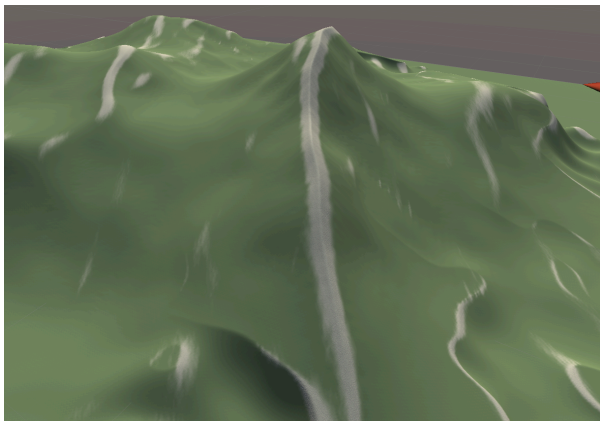
Height Filtering



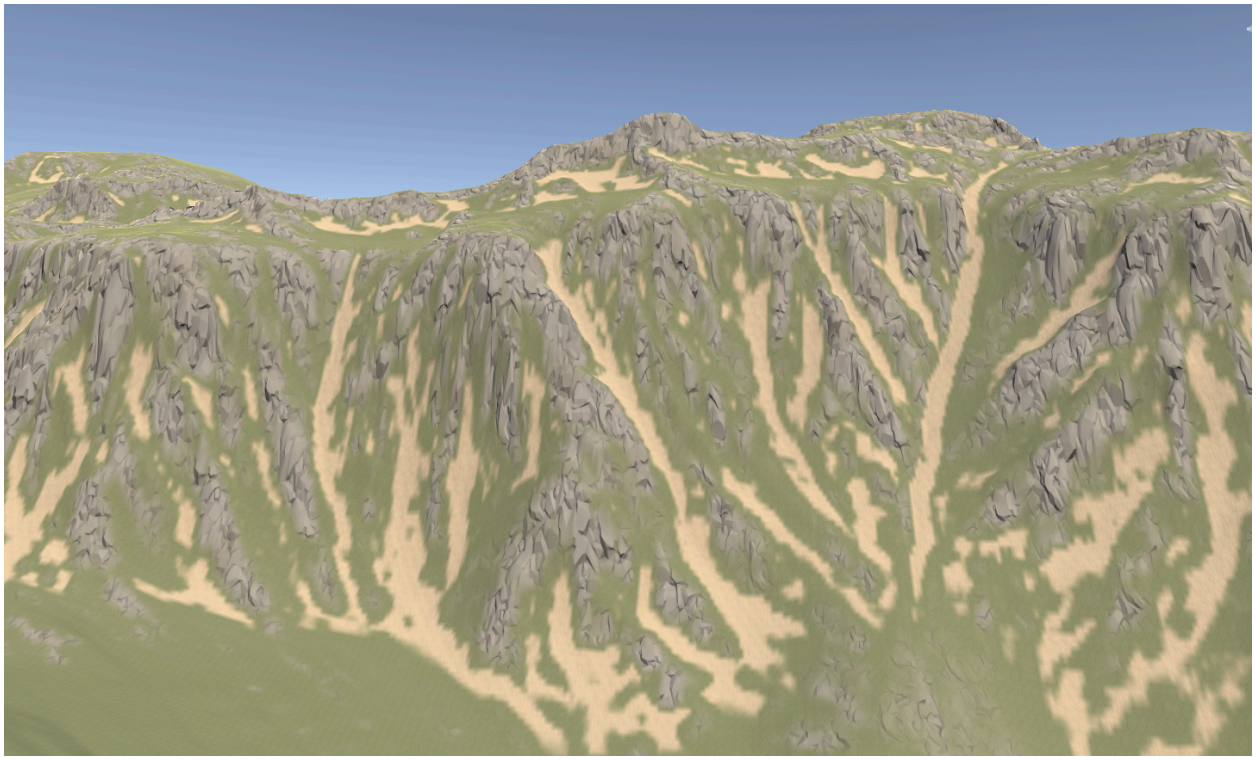
Slope Filtering



Angle Filtering



Curvature Filter
(Heavy Convexity)



This example was done entirely with the flow filter. Low values are filtered for where the rock is (low traffic areas for water), and high values for where the sand is (high traffic areas for water).

Finally, a noise is also available on each filter as well, modifying the input to the filter.

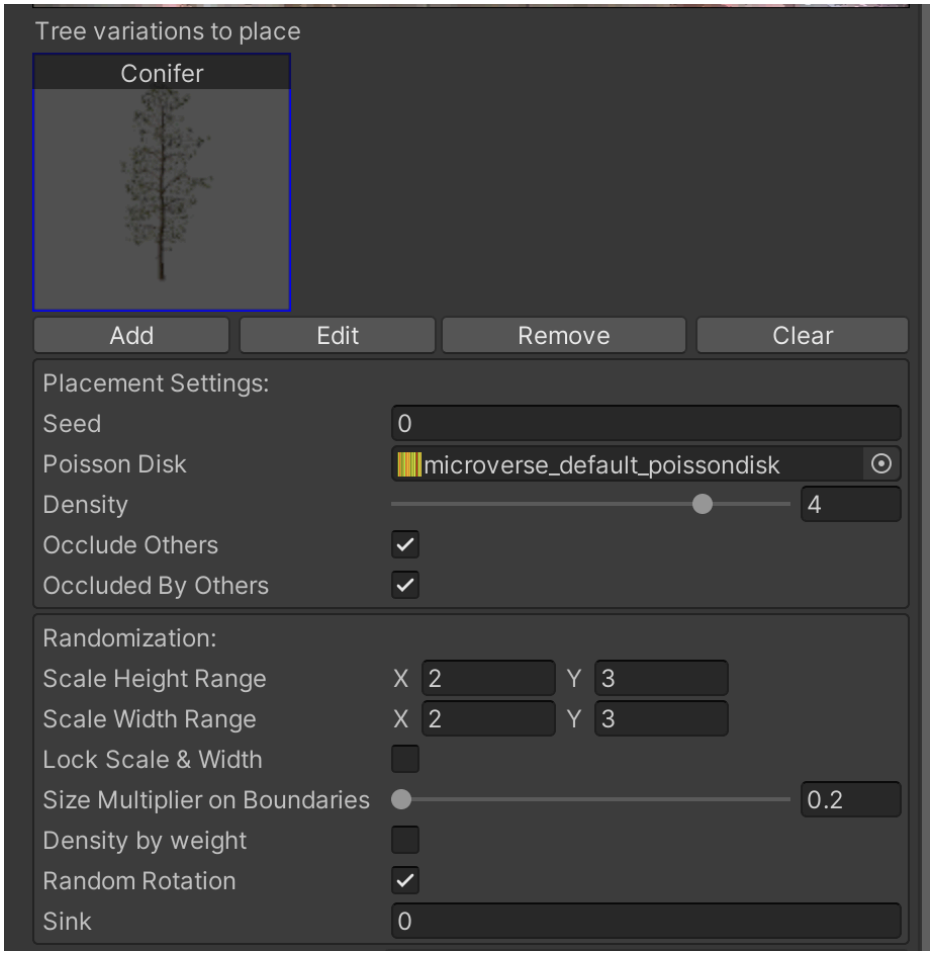
A good video on filters and noise is available here

Tree Stamps

(Requires Vegetation Module to be installed)

Tree Stamps allow you to place vegetation across the terrain. Note that you do not need to setup trees from Unity's terrain interface, you can add them directly to the tree stamp and it will add them to the terrains which use them automatically. Within a given tree stamp, you can

have multiple vegetation prefabs. Think of each terrain stamp as one “system” with variations within it.



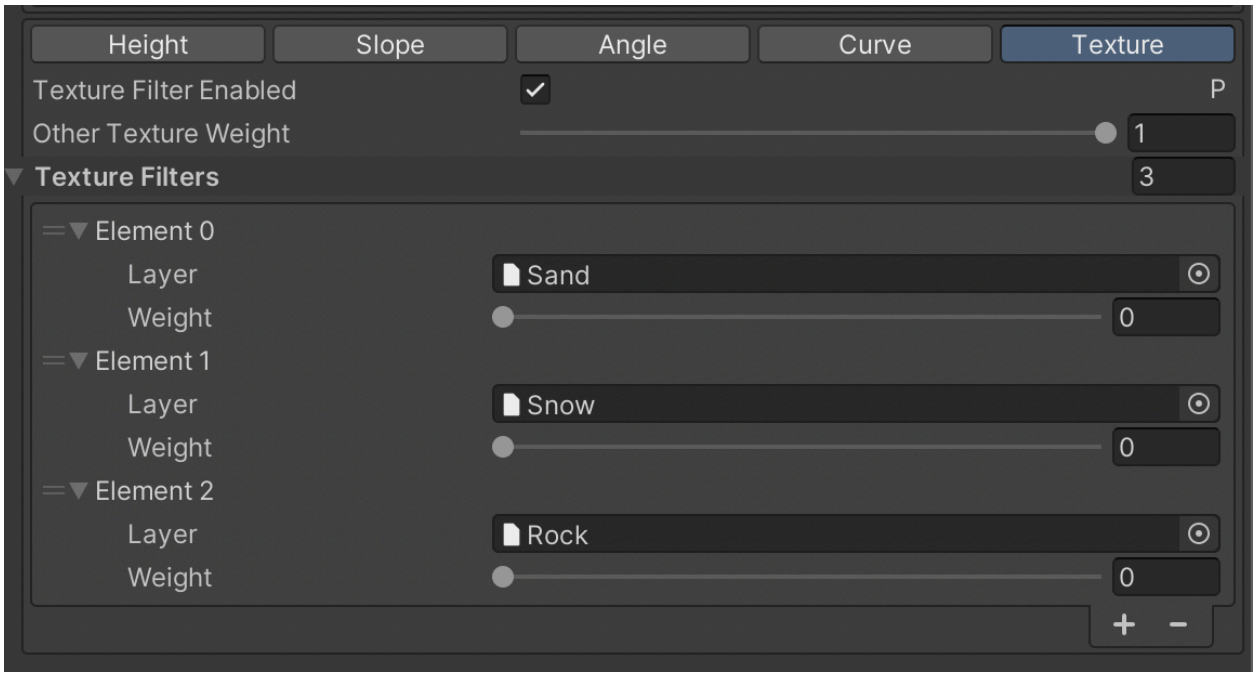
A seed is available which will change the randomization seed for everything within the stamp. A poisson disk image is used to vary the placement of objects without letting them overlap, but it’s unlikely you’ll ever need to change this image. The density slider controls how many vegetation objects should be placed.

MicroVerse has a primitive occlusion system for object placement. You can have these objects write into that buffer, or be occluded by things in it. These settings are all global to the stamp.

More information about how to use the settings in the Tree Stamp are discussed in the Tips and Tricks section.

Below these settings are per-object settings for each prefab in the system. These let you vary the scale of instances, or have the weight of the stamp affect the size of the objects or density as you approach the fading edge of an area.

Finally, all of the falloff and filters described earlier are available, along with a new filter type which allows you to filter based on the textures on the terrain.



The Texture filter allows you to filter things based on which textures are on the terrain. In this example, the other texture weight is set to 1, and the listed layers are set to 0. This means that tree's will appear on any terrain layer except sand, snow, and rock. If you wanted the tree's to only appear on Sand, you would set the Other Texture Weight to 0, and set the sand to 1.

Details Stamp

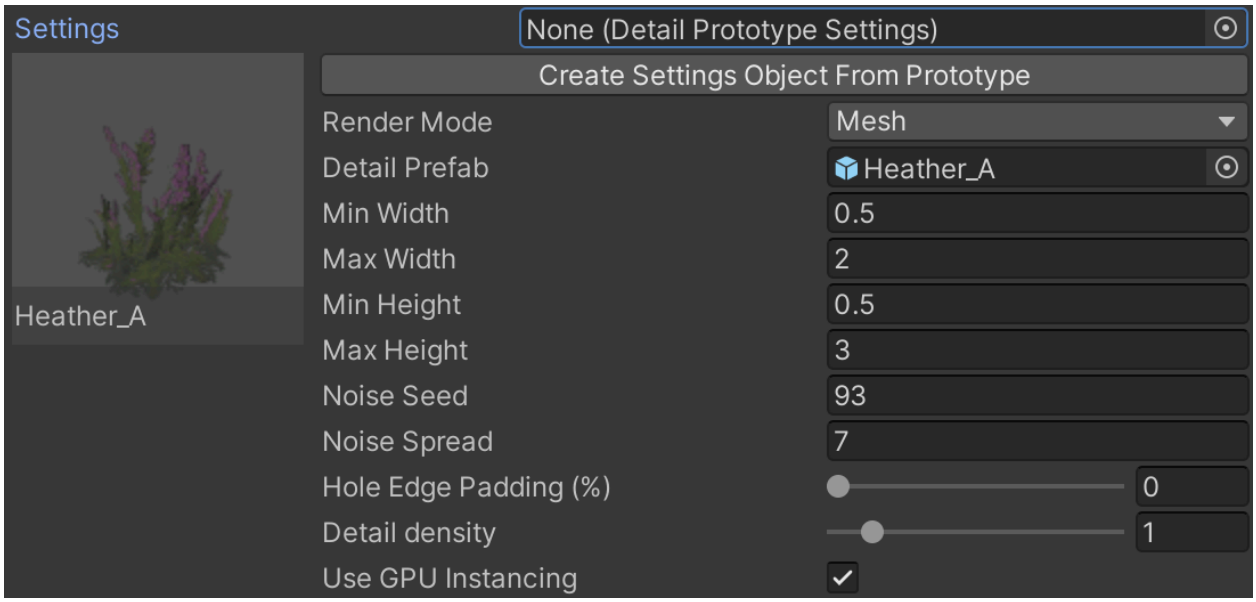
(Requires Vegetation Module to be installed)

The details stamp is similar to the tree stamp, but for Unity's detail system. Due to the way Unity's detail system works, only a single detail object is used per stamp. The controls here

match Unity's detail system when adding new detail objects, and the rest of the controls will be familiar from the earlier documentation.

Note that Unity's detail system computes density in a number of ways - both from the size of the detail map (set on the terrain, or via the MicroVerse component), the detail objects density value, and the layers resulting weight.

Most of the confusion I see when using the details stamp is people not understanding how the detail rendering system works in Unity. It does not "place grass" the way that the tree system places trees. Rather, internally the Unity detail system stores a texture for each detail stamp, which is sized according to the detail resolution on the terrain. So if your terrain is 1024 meters in size and your detail resolution is 1024, you have 1 meter of resolution for each pixel on the terrain. That pixel specifies a density value for spawning your detail object - and Unity will spawn some number of detail objects in that area depending on that density value, jittered and rotated with the detail objects noise setting. These objects may overlap, and there is no specific control over how they are placed or rotated. And if you have multiple detail stamps writing to that area, they will all spawn whatever objects they create in that same pixel worth of area as well. Thus, the detail system in unity is designed for things which can freely overlap and look ok, such as grasses and flowers, which also come in high enough instance counts that storing a weight for each meter on the terrain is more efficient than storing actual placement data.

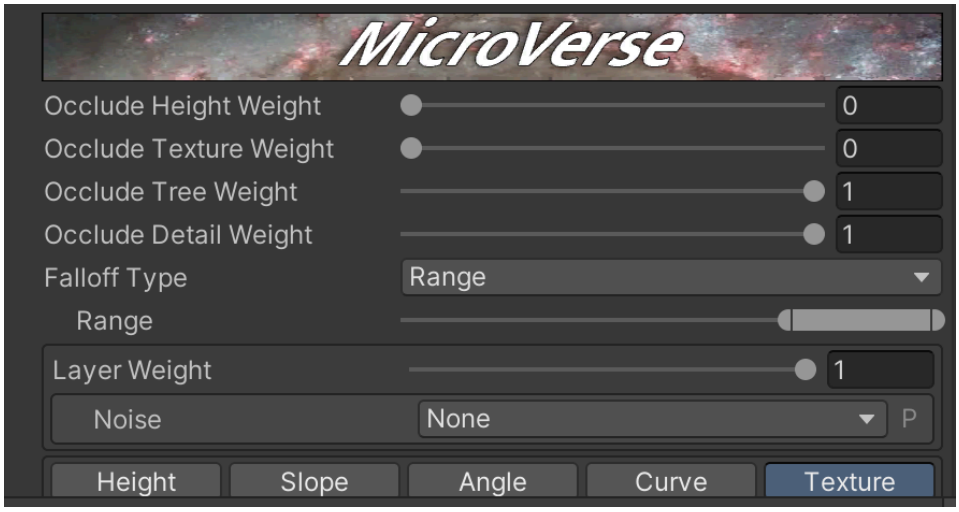


Note that if you have two detail stamps using slightly different prototype

settings(pictured above) then unity will treat these as two separate sets of detail objects, which will render separately and have their own data for density. To help prevent this, you can create a settings object from the prototype, which will copy all the settings into a scriptable object and store it in Assets/MicroVerse/DetailSettings. This can then be referenced from multiple detail stamps, ensuring that the settings can be adjusted in one place and remain the same.

Occlusion Stamp

The occlusion stamp can be used to prevent things from appearing in a given area. Essentially, it writes into the occlusion mask and blocks future stamps from adding their effect to an area.



You can set the amount of occlusion for each type of stamp effect - for instance, this stamp is preventing tree's and details from appearing in a circular area (falloff).

Clear Stamp

The clear stamp is kind of like the opposite of the occlusion stamp. Instead of preventing future stamps, it clears the data from previous stamps. This can be very handy - for instance, you might have a large forest biome and want to create a clearing in the middle of it. You can use the Clear Stamp to do just that- and since it works with noise and filters, you can

even partially clear the area. Want to clear everything under the sea? Just make a clear stamp with a height filter.

The clear stamp only works with trees and details, because textures and height's are intrinsically handled already (A new texture or height stamp can overwrite the previous data). Note there is currently a limit of 256 Clear Stamps per terrain, although this can be raised at the cost of some extra memory if anyone needs to go higher than this.

Hole Stamp

The hole stamp can be used to create holes in the terrain, using the falloff system and placement to specify where that hole should be.

Height Area Effect Stamp

The height area effect stamp allows you to "post processes" height data. It currently has several effects, which can be applied to any height stamp below the height area effect stamp in the hierarchy. The stamp works like other stamps, with falloff controls, but applies its effect to the existing data under it.

Terrace - Create a stair step like effect on the terrain

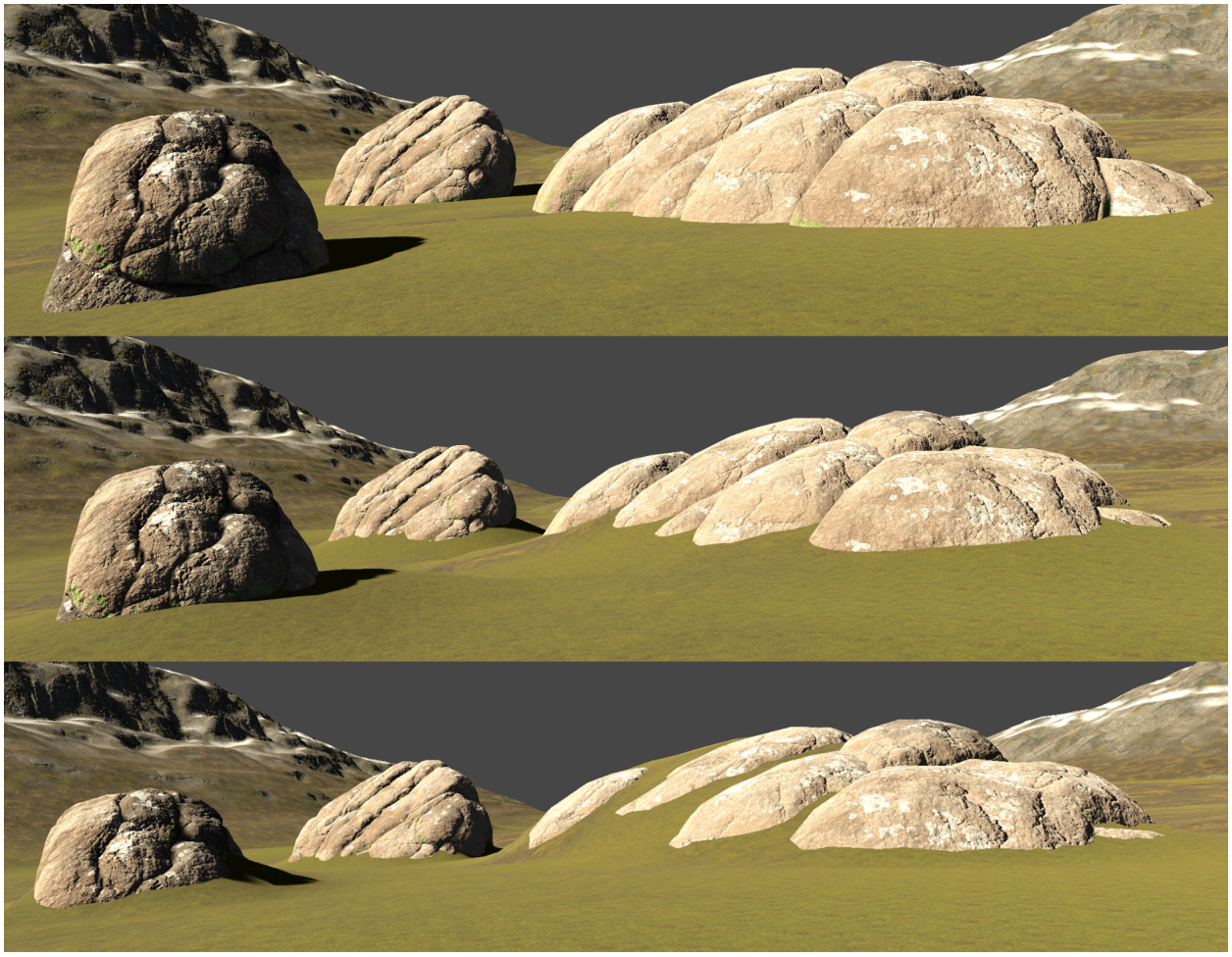
Beach - Pulls the heights closer to the world position of the stamp. This is very useful for beaches, which tend to have a low ramping angle around the shore line caused by the water erosion and deposits of sand in the area.

Remap Curve - Remaps all heights in the area based on the curve you provide

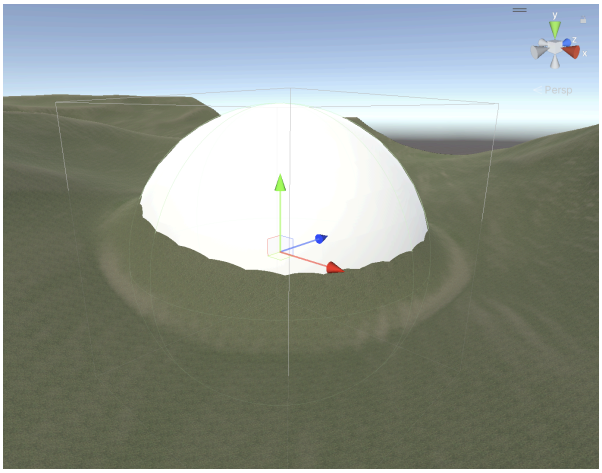
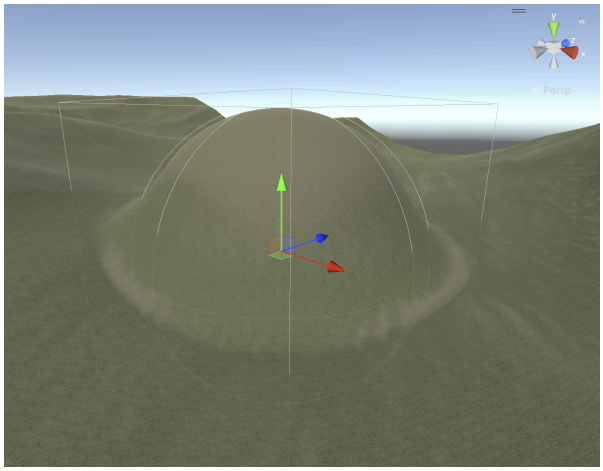
Noise - Applies noise to the heights in the area, combined based on a blending mode.x`

Mesh Stamp

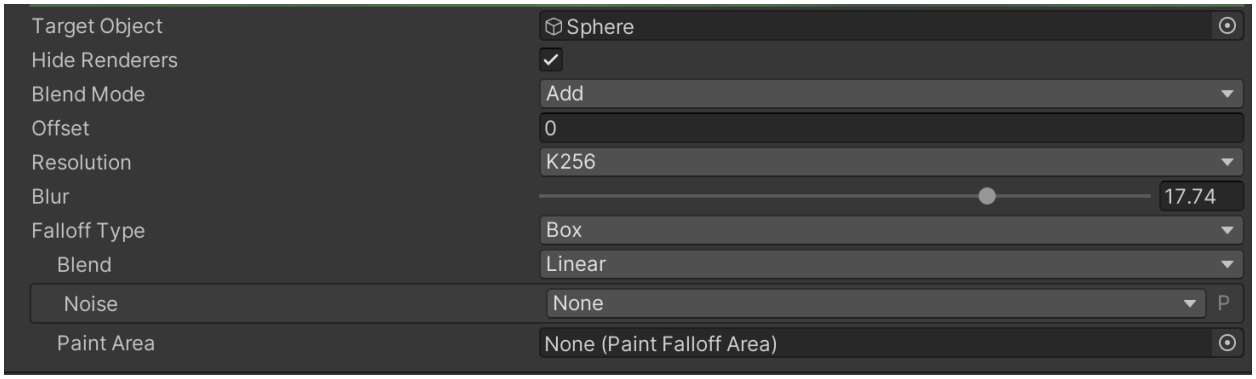
The mesh stamp lets you use a game object in your scene to modify the terrain. In add mode, terrain will be pulled up to conform to the top of the mesh. In subtract mode, it will be pushed down to conform to the bottom of the mesh. It includes blur and offset parameters, which allow you to smooth the result. This can be very useful when integrating the terrain into objects, such as making the terrain come up to join rocks, or carving out space for an object.



Here we see an example of how the effect can be used to better match the terrain to rocks placed in the scene.

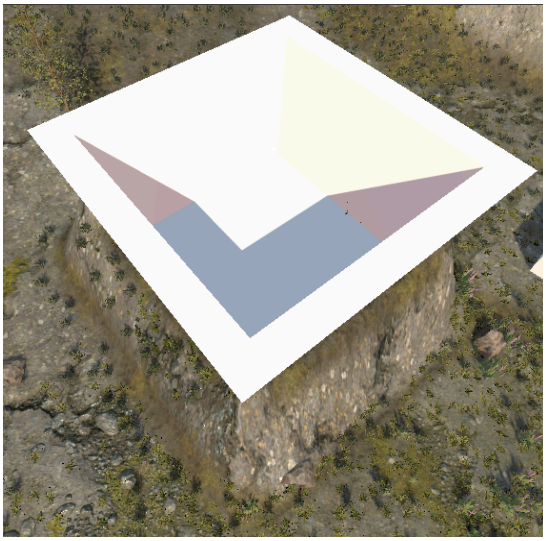
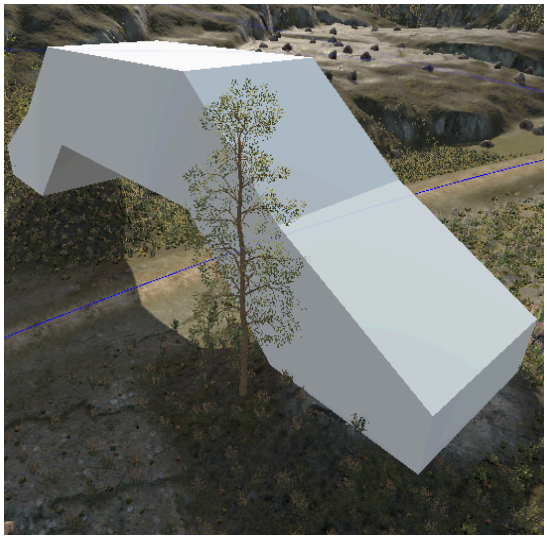
	
Here a sphere is used with Add mode and blur turned up. We see the terrain reaching up for the sphere.	Here we see it with the sphere hidden.

You can also use this feature to model terrain with existing meshes instead of using stamps.



Fillaround / Connect

Add and Subtract are self explanatory Blend Modes, Fillaround and Connect however are specific to the MeshStamp. Fillaround pulls terrain up around the edges of your model, useful if you have hollow objects that you want to blend with your terrain. Connect only pulls terrain up along the lowest parts of your Mesh.

	
Here a bowl shape using Fillaround, where the inside is not filled up with Terrain.	You can see a bridge with its ends connecting smoothly to the terrain using Connect.

The Mesh Stamp component can be added to an object in the scene, and the TargetObject set to that object. It will scan for any MeshFilters on that object and use them to render a depth map for the effect.

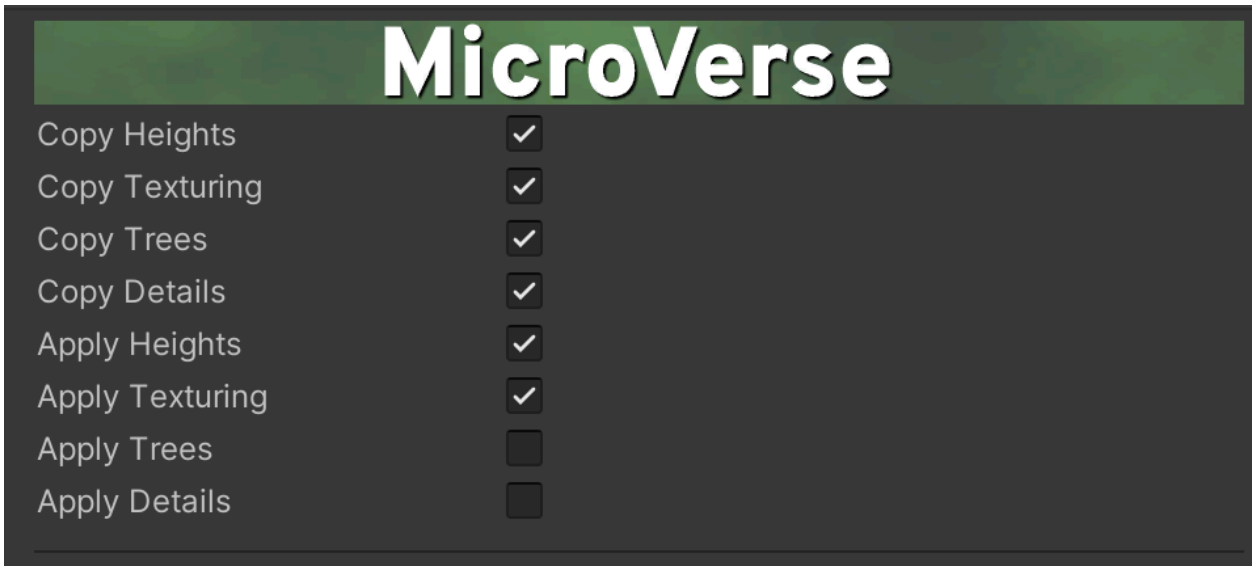
You can offset the effect along the vertical axis, choose the resolution of the depth map, and set the blur amount (which is based on pixels on the rendered depth map). The rest of the stamp should feel familiar.

Copy/Paste Stamp

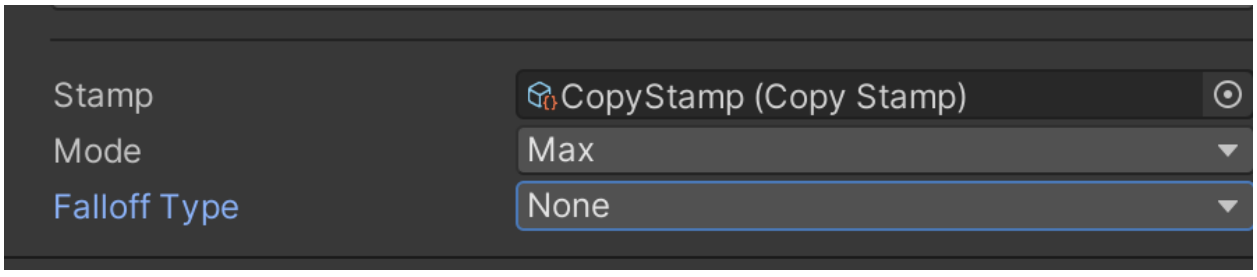
To understand how this stamp works, you'll need to understand a bit about how MicroVerse works internally. The stamps previously discussed start from a blank terrain and build everything up from scratch. But perhaps you want to customize an area with Unity's paint tools, or copy some area that you really like? The copy paste stamp can do this, and is an extremely powerful way to use traditional terrain tools in a non-destructive way. But if you just

open up the Unity paint tools and paint, when MicroVerse recomputes the world from scratch it will erase those changes as it will start from a blank terrain and rebuild its changes based on the stamps it has.

So what the Copy/Paste stamp does is allow you to create a stamp from whatever is on the terrain in its falloff area, and then re-apply that data to the terrain in the procedural process. You can then move that area around, place it multiple times, prefab it with any objects parented to it, and even apply additional stamps on top of it.



At the top are settings to set what you want to copy or apply, along with a Create Copy Object button. Once pressed, it will ask you to save a stamp object, which is where it will store the data.



Once created, you can set the height mode for the stamp and falloff settings. You can then freely move the stamp around and the data will follow.

Note that it's very easy to trigger a refresh of MicroVerse - enable/disable a stamp object, or otherwise changing a value in the system, will trigger it to rebuild the world from scratch. So if you are using the Copy/Paste stamp to do extensive painting/modification of the world manually, it is best to save the scene and disable the MicroVerse component while you do these changes, then make the copy and re-enable it. This will prevent an accidental refresh of MicroVerse from destroying your change.

There's also a pixel quantization option, which is true by default. When true, the copy paste stamp's scale is quantized so that the stamp lines up with the pixels of the underlying data of each channel. This guarantees a perfect copy. However, if you are scaling or rotating the stamp, you'll notice that it effectively point sampling the data. Turning off quantization will allow it to interpolate values between the pixels, which can provide a smoother (but not exact) past of the data.

Working with an existing terrain

MicroVerse rebuilds the world from the beginning each time a change is made. This is how it achieves its non destructive workflow. But that beginning is assumed to be a blank terrain. However, sometimes you might want to work with an existing terrain and add stamps on top. This can be achieved by use of the Copy/Paste stamp, but requires some care in its approach. To make this easier, there is a menu item which will perform this setup for you.

First, **back up your terrain**, as doing this wrong can easily white the data from the terrain data object.

Select the "GameObject/MicroVerse/Create MicroVerse for existing Terrains" option. It will warn you to backup your terrains first. After confirming, a MicroVerse game object will be created, and all terrains in the scene will be reparented to be under it. For each terrain, it will create a Copy/Paste stamp which is sized and placed exactly over the terrain, and use it to capture the state of the terrain as the starting point for all Microverse work. It will save the data about the stamp next to your terrain data on disk. This data allows MicroVerse to reconstruct

the terrain to this initial starting state at the beginning of its generation of the world, so do not delete it.

Splines

(Requires Spline Module to be installed)

If the Spline package is installed, several new features are available:

- The ability to create spline based paths, which affect the height of texturing of the terrain, and can clear tree's and details from their path.
- The ability to create areas with splines, which can be used with the falloff system. This lets you create arbitrarily, custom shaped biomes, and adjust their positioning and shape in realtime.

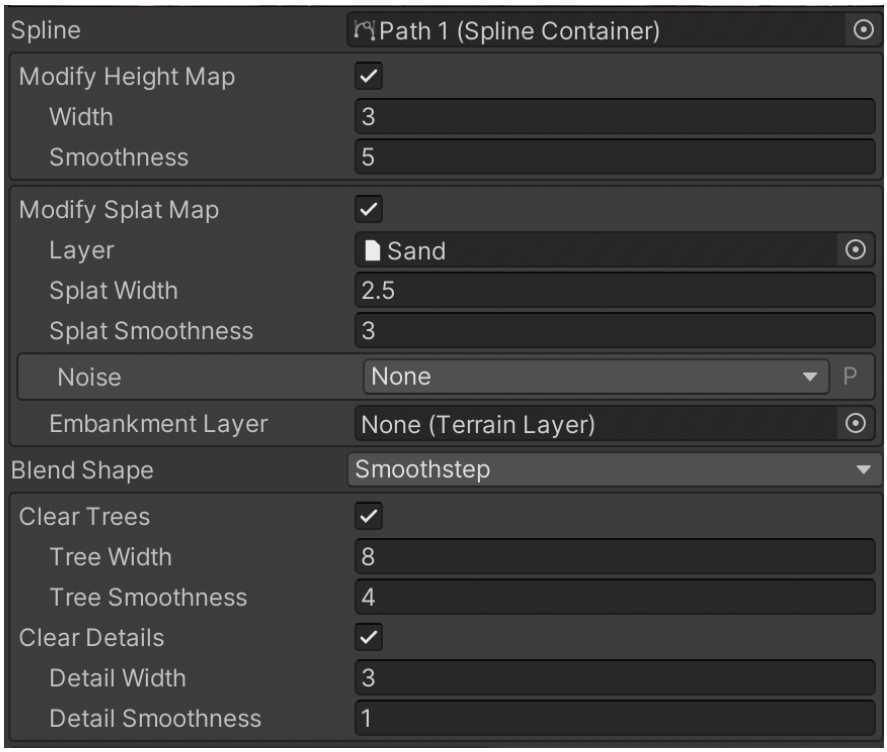
Installing the Spline package

If you are in Unity 2022 or greater, you can install the spline package from the package manager.

However, in Unity 2021 the spline package is hidden from users. To install it, open the package manager and press the + in the top left corner, and select "Add Package By Name". Type in com.unity.splines and press return.

Creating a spline path

The Spline module uses Unity's Spline system, which has additional features for placing objects along splines, generating meshes, etc. To create a path, create a new spline from the GameObject/Spline/Draw Spline Tool and create a path. Then add the SplinePath component to the spline, and make sure the game objects are under the MicroVerse game object.

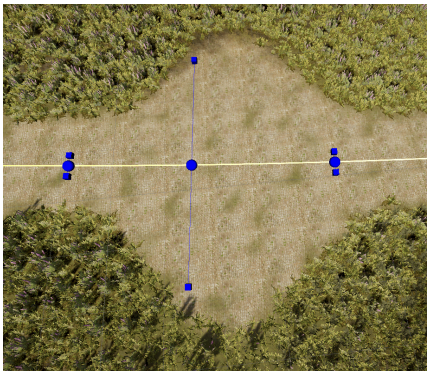


Set the spline reference to the spline you created. You can have the spline modify the heightmap or splat map, and control the area around the spline (width) and falloff (smoothness) of the effect. The texturing area lets you choose a layer to paint where the spline is with its own width/smoothness setting, and an embankment texture for the sides of the trail. You can also apply noise to the texturing to create a more broken look to the trail.

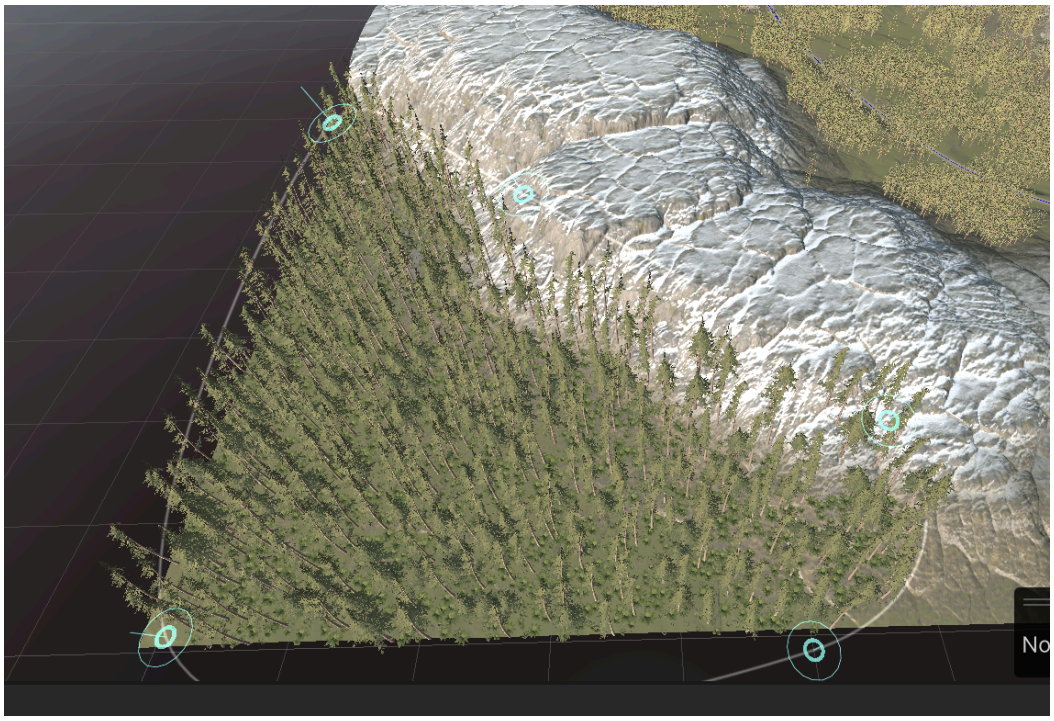
You can also specify width/smoothness for clearing trees and details from the path.

Spline Widths

Splines themselves can have width added to them, allowing you to adjust the width along points of the spline. However, though MicroVerse supports having multiple splines in one spline container correctly, Unity's tools for working with multiple splines are incomplete, and custom spline data (like width) cannot be added to a spline. This means width is only supported on the first spline.

	When a Spline is selected, you can use the first control in the overlay toolbar to select spline mode, and the last one to adjust the width of the spline. Once selected, you can double click anywhere in the spline to create a new width control point.		You can widen the spline by using the handles on either side of the width control points, or move the control points along the spline. Right clicking on a width control point removes the width control point. At the bottom of the SplinePath inspector, there's an option to change the interpolation on the spline widths. The image to the left is set to smoothstep.

Spline Areas

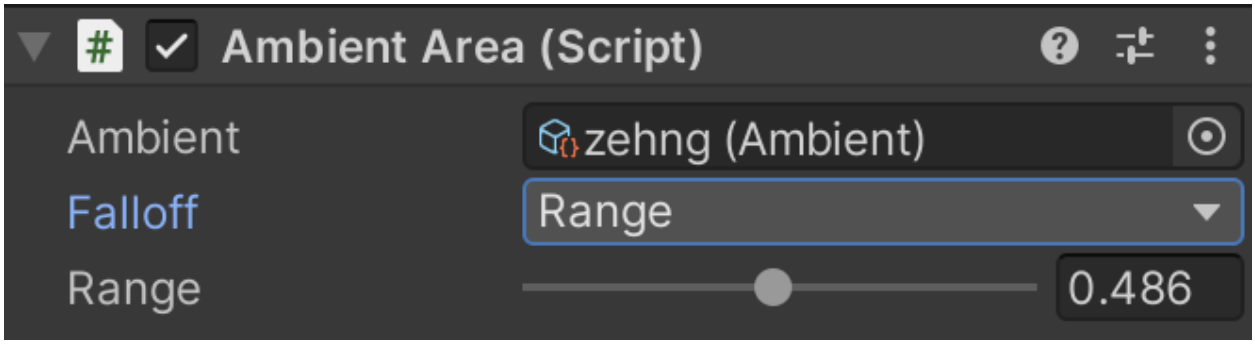


To create a spline area, create a new, closed spline around an area, and add the Spline Area component to it. Add any stamps below it and change their falloff to “Spline Area”, and assign the spline area to them. Rather than using the stamps falloff, any stamps assigned to the area will use the spline area instead, including height stamps. They will also have a falloff property that lets you control the weight around the edge of the spline. Note that when using a height stamp you still have to adjust the transform of the height stamp to scale/place it within the spline area.

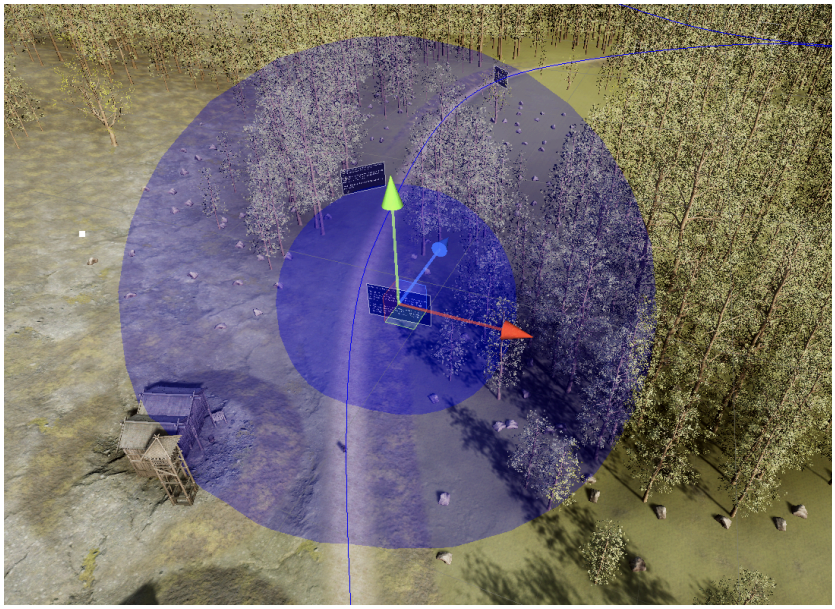
Ambient Area

If you have the MicroVerse - Ambiance module installed, you can place areas of ambient sound effects and music, and use similar falloff’s to constrain them to specific areas of the world.

Each ambient area has a reference to an Ambient scriptable object, which can be created from the right click menu.



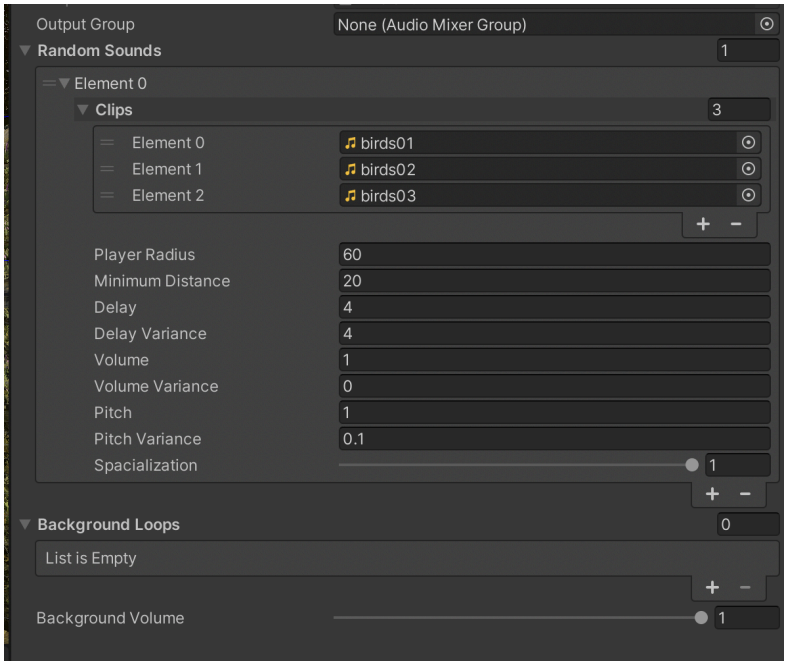
When set to box or range, the falloff is a single value that specifies where in the box to begin fading audio. This is visualized on the volume:



When a spline is used, the first value is the distance (in unity units) that the fadeout will begin from the spline, with the second value being the end distance of the fadeout. There is both a Spline option for paths, and a SplineArea option for areas.

Finally, you can use the MicroVerse Masks module to generate a texture based mask of where a sound will exist. This is extremely useful for things like water. If you check out the water in the demo scene, as you move the plane it clears trees and details under the water, and these stamps get updated via a binding component that maps their height filter range to the position of the plane. This same trick can be used with the mask stamp to keep your water area updated

as the plane is adjusted. See the mask module documentation for more information on how to setup a water plane with sound.



Here we see an example of the ambient scriptable object. Many different areas can reference a single ambient scriptable object, but only one of each instance will play regardless of how many areas use it. In this example, we will trigger one of three bird sounds every few seconds. They will be placed a minimum of 20 meters from the camera, and a maximum of 60, specializing them around the camera. Each time they play, they will have a small amount of pitch variance applied.

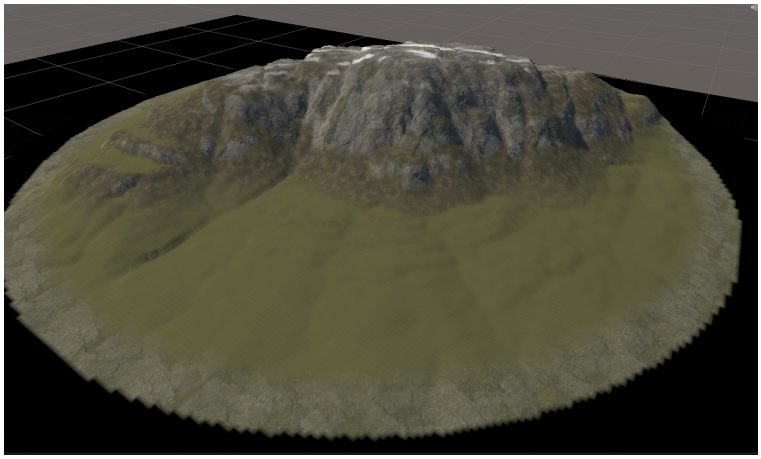
A looping background track can also be provided, used as a noise floor or musical soundtrack. Note that you can use as many areas to define your areas, but you can also have multiple areas overlapping each bringing in different ambient's to play.

The Content Browser

A video on the content browser.

MicroVerse has a content browser which you can open from the Windows/MicroVerse/Content Browser menu. From the browser you can access pre-existing

stamps designed to work with MicroVerse, as well as download new content packs which provide you with more content. These assets may combine multiple stamps to create more complex effects, and downloaded packages may install additional content in the browser to speed up your workflow. For instance, a single stamp might contain multiple texture layers, which texture your world in a specific style. Some example content has been included.



In this image I dragged a single height stamp and the 'Alpine' texture stamp into the scene. These can both be found in the MicroVerse-Examples packs. The alpine texturing stamp is a prefab that contains multiple texture layers, applying rock, grass, dirt, and snow to the terrain. By default, alpine is using a radial falloff, but you can easily change this to work globally across your terrain or to use a spline based falloff, for instance.

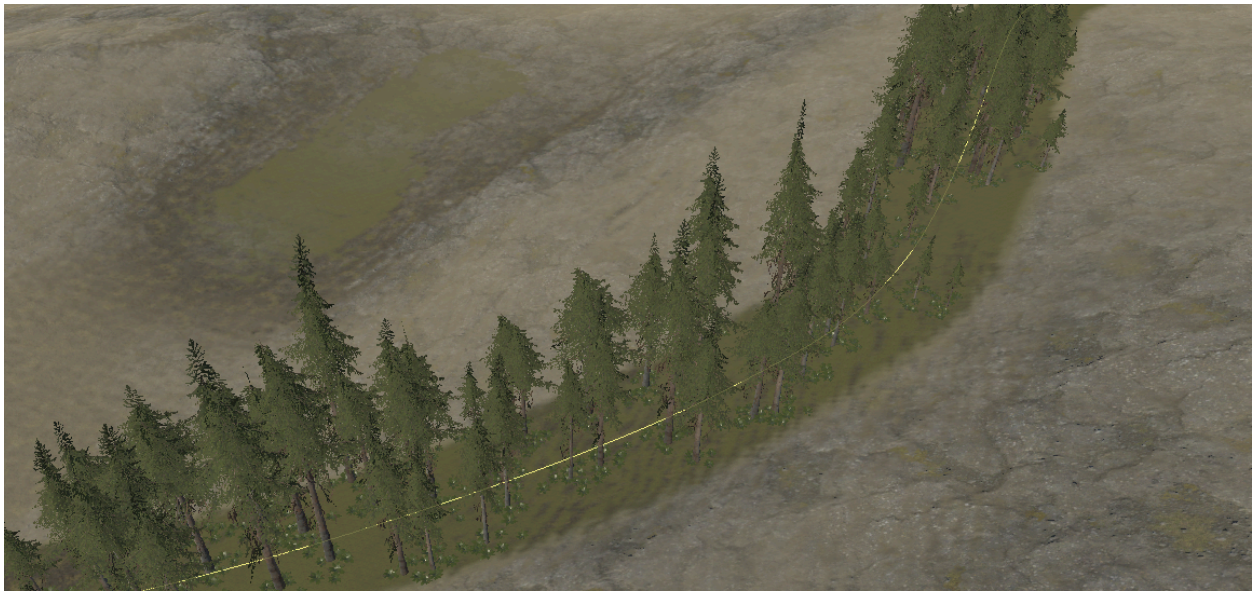
This is a far more convenient way to access content created for MicroVerse than the Unity Asset folder structure because all of the content gets organized by type in one place, instead of spread across a file system. And while this system is originally designed for asset store publishers to make it easier to use their content in MicroVerse, it can also be used by teams to organize common presets into quickly accessible stamps.

The browser also allows content publishers to link to external content and provide an image to advertise that content from within the browser. These show up as additional packs in the content browser with an image and download link instead of the actual content. If you are an asset store publisher and interested in providing this type of content for your users, please contact us so we can feature you right in the content browser.

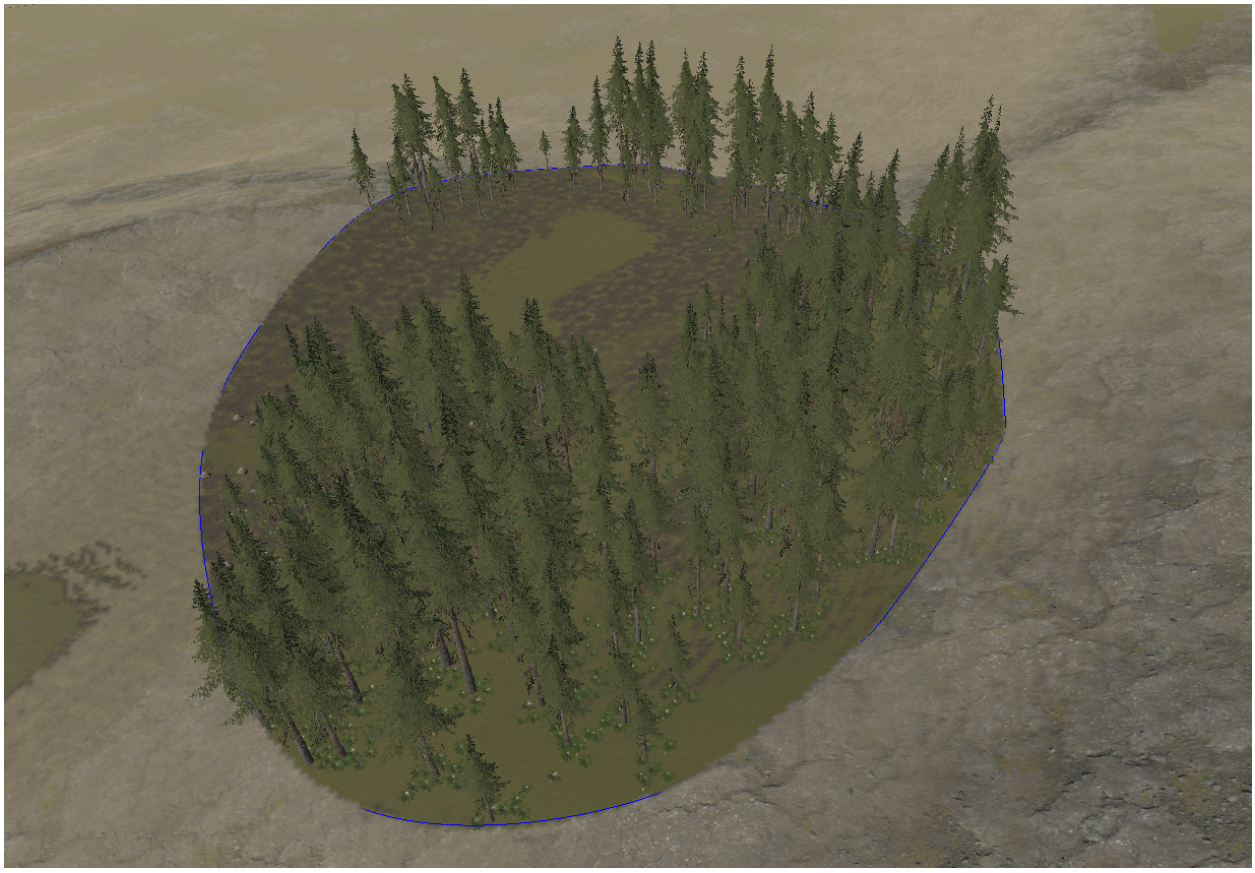
Content Browser Spline Painting Tools

At the top of the content browser, there are several modes. The default mode, represented by the arrow, allows you to drag content into the browser, with the content using whatever defaults for falloff's it was authored with.

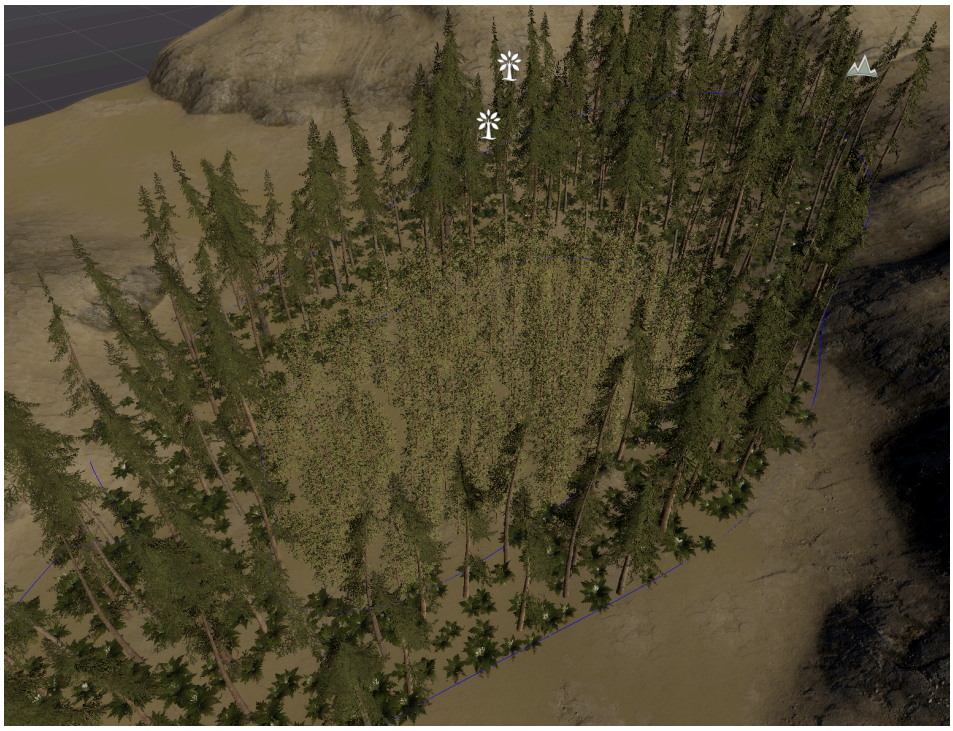
The second option is the paint Spline Path mode. This will allow you to paint a spline path over the terrain in the scene view.



The third mode is the Spline paint mode. This will create an open ended spline with a spline area assigned to it, and whatever content is selected in the content browser will be constrained to that spline. Above is an example of a biome constrained to a spline.



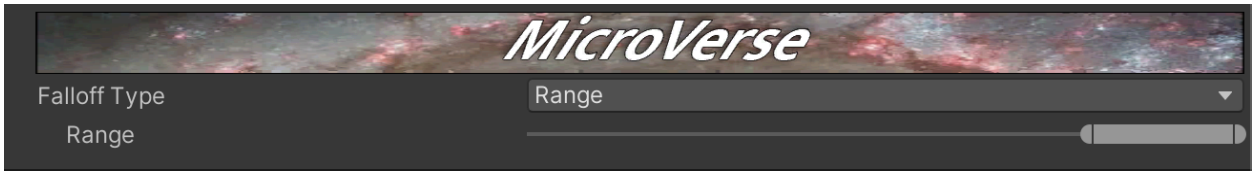
The final option creates a spline area and a closed spline, and constrains whatever content you have selected to the area. Above is an example of a biome constrained to the area of a closed spline.



If you press Shift while using one of the last two modes, a clear stamp will be added to the spawned content. This can allow you to easily remove the effect of the previous area. Notice the tree's spawned inside of the pine forest clear out the pine forest instead of overlapping it.

Falloff Override Component

The falloff Override component is a useful component when building collections of stamps for reuse that might want to have a single place to adjust their falloff. When added to the top of a hierarchy of stamps, all stamps below it will use these falloff settings.



MicroSplat Integration

(Requires MicroSplat to be installed)

MicroSplat, the best terrain shader for Unity, is well integrated into MicroVerse. By default MicroVerse manages what textures are on the terrain based on what stamps are in use, and will freely add or remove terrain layers as needed. And what textures are needed for each terrain can be different. However, this presents a problem for Texture Array based shaders like MicroSplat, which need all the textures in the same order and need to repack the arrays when textures are added or removed.

So when using MicroSplat, whenever new textures are added via a stamp, MicroVerse will show a warning and ask you if it can add the new texture layer and repack the arrays. It also has several controls on the main MicroVerse component to convert the terrain to MicroSplat, remove any unused textures from the arrays, or add any missing ones.

For other array based shaders, there is an option to keep all terrain layers the same across all terrains.

To convert your terrain's texturing to MicroSplat, you can simply press the button on the MicroVerse component, or assign the texture array config to it if you have gone through the traditional setup.

Always add your textures to a texture layer and then use it in MicroVerse and let MicroVerse update the texture arrays. The only time you should need to add textures to MicroSplat is if they are textures which are not available in Unity's Terrain Layer system, such as textures for Trax, clusters, etc.

MicroSplat 256 Texture Module support

Experimental support for MicroSplat's 256 Texture module is available. Note that the copy/paste stamp is currently not supported. This will allow you to work with up to 256 textures on a single terrain, using a single splat map texture as well.

To use it, convert your terrain to MicroSplat as per the instructions above. Set up your terrain for 256 texture mode as per the module documentation. Once in this mode, test that you can successfully paint on the terrain using the terrain painting mode. If this works, you should be able to use up to 256 textures on your terrain.

Note that Unity will still allocate splat maps for every 4 textures you use on your terrain. To prevent this wasted memory, set the texture sizes on the terrain to something very small. Note that in this mode, other tools or scripts which try to access the texture data will not work because the data in the Unity terrain does not exist - you will have to write code to sample the terrain data yourself from the resulting textures. The format is described in the MicroSplat 256 texture module documentation.

Demos

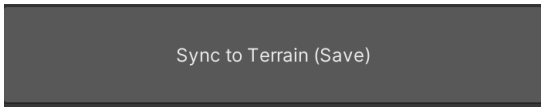
Each package contains its own demo scene, but if you own MicroVerse, MicroVerse-Vegetation, and MicroVerse-Splines, there's a demo with all three in action at:

https://github.com/JDB-Tech-INC/MicroVerse-Demo

This was excluded from the collection download to save space, as it adds another 500mb to the package size.

Common Issues

MicroVerse attempts to be very sneaky about when it sync's back to Unity terrain, as this can introduce random pauses into the workflow. However, Unity terrains uses some of the CPU side data to do things like set LOD levels, cull terrain, or move detail objects to the ground - and when things get far enough out of sync you might notice rendering errors and clipping of the terrain. However, you can fix these by pressing the "Sync Back To Terrain" button on the MicroVerse component, or by simply saving the scene with ctrl-s. Usually these issues are minor, and will fix themselves once MicroVerse finds a moment to sync back to the terrain.



Tips and Tricks

MicroVerse is built on a number of reusable concepts, designed to work across multiple terrains, and intrinsically handles complex problems like biome and terrain resource management for you. You can integrate in traditional workflows like the terrain painting system via the copy/paste stamp, while giving them new features like being able to move or replicate your work. Everything happens in real time, with no baking. With some understanding of the system, you get a completely non-destructive workflow that lets you rapidly take an environment from concept through production.

Prefabs:

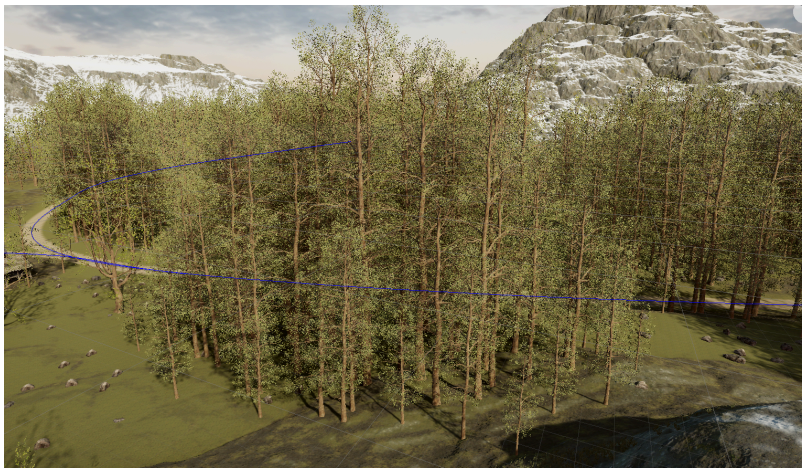
Stamps contain all the data they need and thus can be turned into prefabs and copied from one terrain to another. For instance, you could prefab a bunch of texture layers as a preset and drop it onto a terrain to quickly texture it. Or create a spline based area for it and drop it into that area to texture it. Or you could prefab a small village, complete with painted texturing, and drop it onto another terrain and set it into a mountain with the terrain adapting around it. Since everything the stamp needs is included with the stamp, it will carry the textures and vegetation with it, and any other objects you throw in there.

Note that some of Unity's terrain shaders have a hard limit of 8 textures per terrain, and that it's really easy to go over that. With MicroSplat, that limit is raised to 32 in all render pipelines.

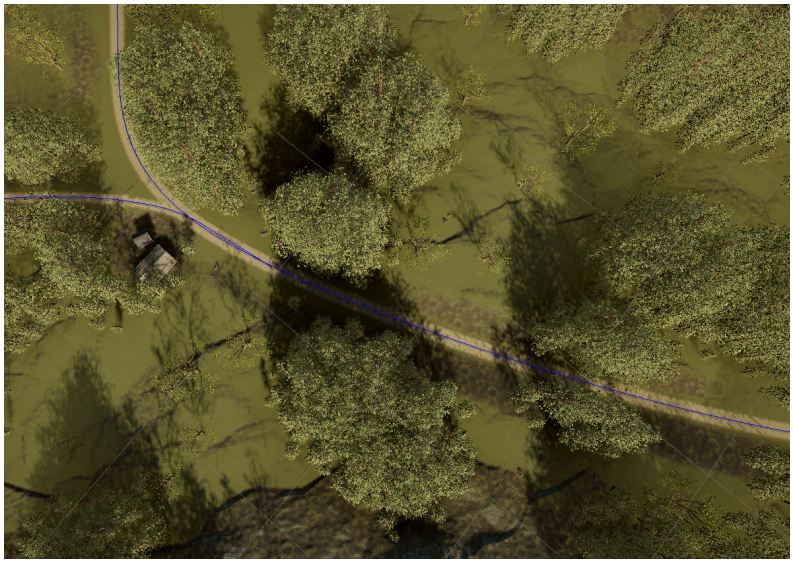
Trees

Things in nature are generally not just randomly scattered over an area, but rather contextually end up in certain places. Filtering is, in general, a powerful tool to create context in placement - using a texture filter, for instance, piggybacks off of the previous context aware patterns created in texturing.

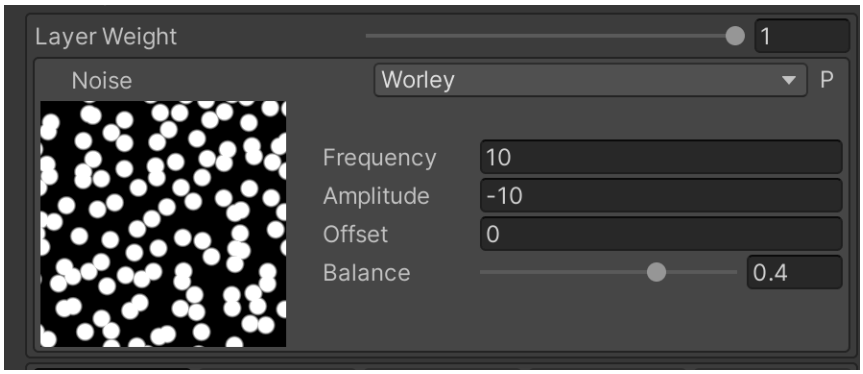
However, tree's in particular tend to grow in clusters and clumps until the forest grows in. This can be easily simulated in MicroVerse, and produces much more natural looking tree areas.



For instance, here we see a cluster of trees which look like they grew up around an older tree. Now look at it from the top:



Notice how multiple clumps create the effect of natural clusters forming, with meadows between. This creates a very compelling look which is fun to explore.



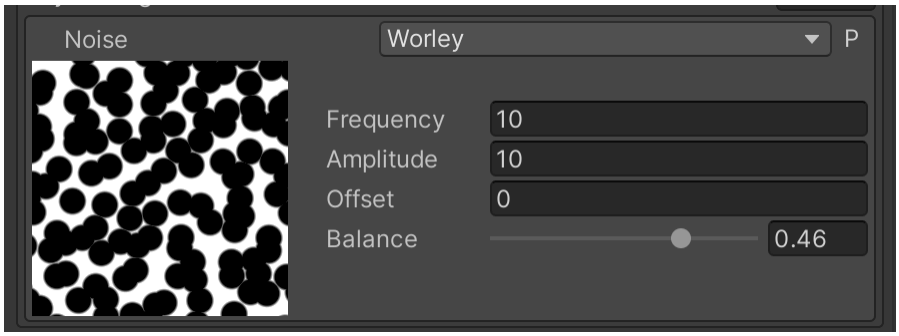
The way this effect is created is by creating a noise pattern for where tree's appear. In this case I'm using worley noise, which when pushed to high amplitudes and balance settings, creates a set of dots. Each dot represents our cluster of trees, with the areas between them being our meadows.

The other trick here is the use of the "Size Multiplier on boundaries" slider. At 0.2, this means that tree's will scale down 20% as they are weighted low (this translates to the black areas of the noise). Note that on tree's noise is unclamped, so that size multiplier will make areas of the noise that go above 1 produce even bigger trees.

With this combo, we can create clusters of individual trees that are tallest in the center and scale down as they get to the edge of the clusters.

There are also options to scale tree's randomly between a given range, based on the resulting weight of the filters, or based on the height filter result. This can be used to scale trees as they get higher in the scene (trees higher in altitude tend to grow smaller).

But what about the meadows? How do we populate them?



By simply inverting our noise amplitude, we create a filter for the meadows, allowing us to place separate objects in those areas. For instance, we could have flowers that only grow in the meadows.

Unity's detail system

Internally Unity stores detail rendering as a series of image maps controlling the density of how many details are in a given area.

The controls for density are kinda funky in Unity. First, you have a resolution for how large this texture should be. So if you set detail resolution to 512x512, and your terrain is 512 meters in size, then each meter on the terrain has a pixel saying how dense that area is. Unity has two other controls on the terrain as well - the patch resolution, which says how many objects to spawn per pixel if that pixel value is 1 (ie: 32 grasses in that meter area max), as well as an overall density amount that ramps up and down the density globally.

This is not the parameterization I would have chosen, as it becomes difficult to represent things like "I would like this flower to appear about once every 10 meters". But this system is kind of assuming you are painting these down, and as such would just hit each area with a light stroke of flowers. Further, when you adjust an area to low densities, it does not "thin out" the area as you would expect, rather a given cell in that grid either has 1 grass or no grass.

In Unity 2022, to address this and other issues, Unity added a new option to terrain called "Detail Scatter Mode", with virtually no documentation about what the difference is, or which mode is the old mode. Terrain created in 2022 defaults to using this new mode, while terrain in

2021 uses the old option by default. The old mode is called “InstanceCount”, and the new mode “CoverageMode”. From what I can tell, in coverage mode the object’s area is taken into account, and low density values result in detail objects being scattered far apart.

While it is not possible to make both these modes produce the same result, when set to Instance Count mode and density goes below 1, MicroVerse will generate a special noise to remove grass randomly, simulating the thinning you’d expect and want in the old renderer. Note, however, that a terrain can only be in one mode, so content created for one mode will have different results than content created for another mode. Note that some 3rd party vegetation renderers only support the old (Instance Count) mode, or may ignore the setting all together and just render the data as if it was in the old mode.

Clear Stamps:

Clear Stamps (and actually occlusion stamps as well, though not as intuitive), can be used to great effect when working with biomes. For instance, you might populate the world globally with a set of stamps to create a lush forest, then use a spline to have a path cut through that area. Internally, the spline path is writing into the occlusion system to prevent things from spawning over the path. You can do this as well by adding an occlusion stamp to something. However, the occlusion stamp prevents anything beyond it in the hierarchy from spawning, so is not as suitable for other uses.

A clear stamp works in the other direction by preventing anything before it in the hierarchy from spawning. So, if we have our forest with the spline going through it and want to create, say, an area of swamp inside of it, we can add a clear stamp to our swamp stamps to clear out the forest in that area. Then if we want a small grassy hill in the middle of that swamp, we could create another clear stamp after the swamp in the hierarchy to clear the swamp away. This lets us work in photoshop like layers with tree and detail stamps.

And note that clear stamps work with filters, splines, and noise to create any combination of shape or clearing amount - so if you just want to thin out an area, a clear stamp with a little noise will thin it out, rather than completely erase it.

Area pads:

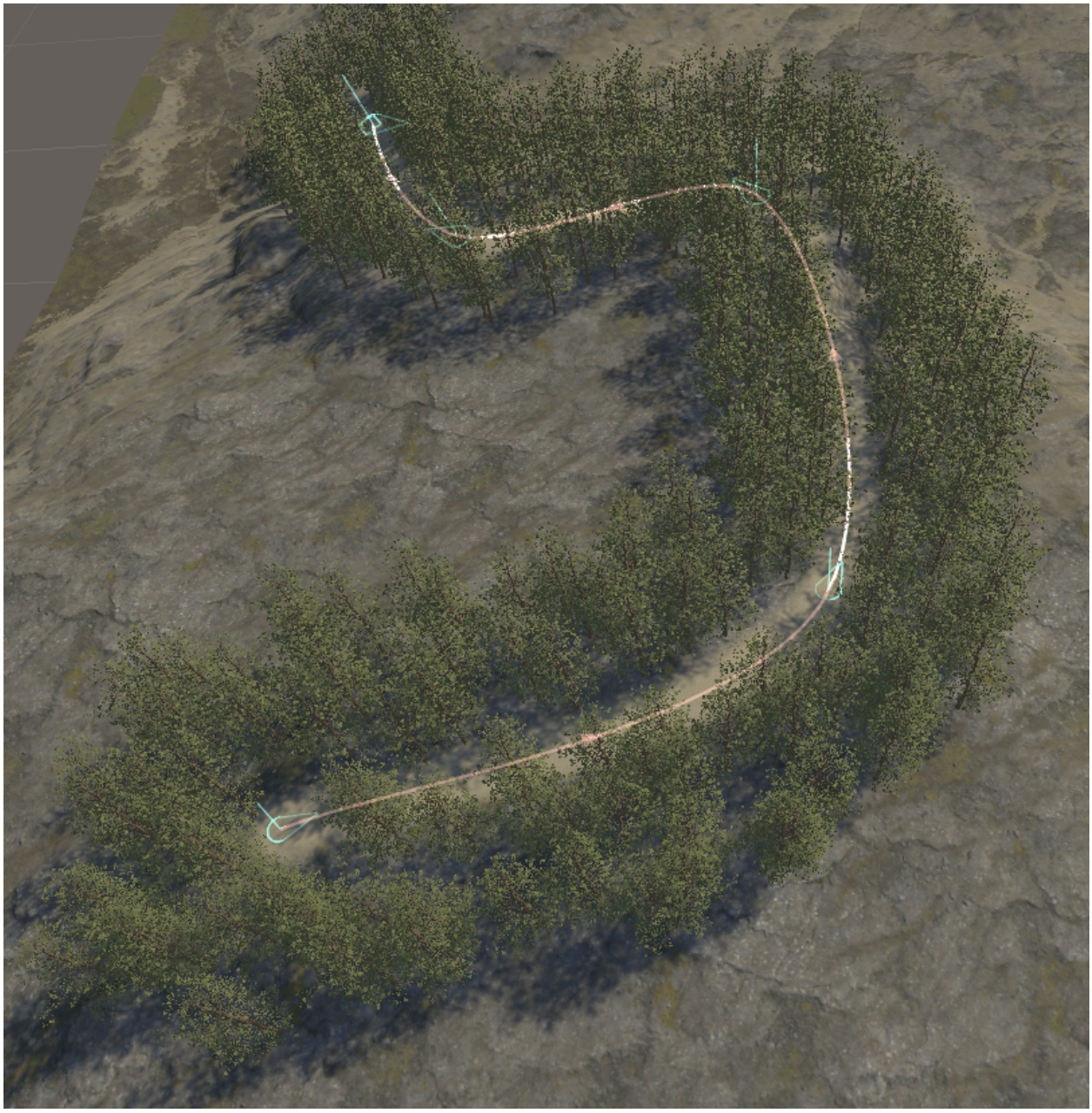
If you want to flatten an area of terrain around a house or a bunch of buildings, you can create a height stamp with no stamp assigned and parent your house to it - the terrain will rise up to meet this stamp, conforming around it based on its falloff. You can also use this to set a default height for your world.

You could also add an occlusion stamp to prevent tree's and details from happening in the area, or a clear stamp to clear what is already there. The terrain will conform to the shape and height of the stamp, acting as a pad to build your scene off of. You can then move that area around, and the terrain will modify itself to conform around it.

Trees along a spline

Spline areas are primarily used to shape areas, but what if you want to place things along a spline? You could use Unity's SplineInstantiate script to do that, but it lacks ground collision, and produces game objects instead of using the tree system. You could have a spline area around the spline path, but then editing is clunky.

However, you can have a non-closed spline work as a spline area too. Since the falloff goes into the spline area, it's always infinitely thin. But, on the Spline Area component there is a "Width Boost" value. Setting this to, say, 4, will make the spline 4 units thicker than the spline defines it as. You can then use the spline area falloff for any stamp to do things like spawn trees along a spline, or Objects with the Object module.



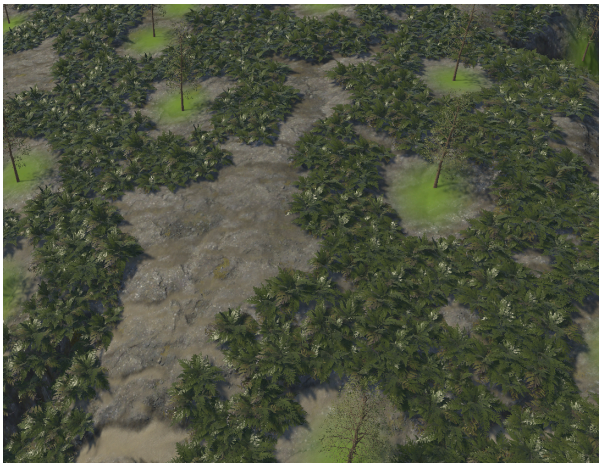
A Tree Stamp is constrained to a spline with a spline area falloff. The Spline path occludes the tree's in the path area. Note that you can easily paint areas like this using the Content Browser's spline painting tools, but understanding how spline areas can constrain any stamp is super useful.

Trees distance fields

Tree stamps can generate a distance field, allowing it to perform lots of different effects.

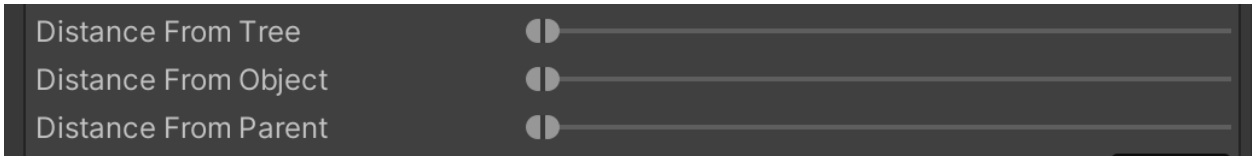
	For instance, when a tree gets large enough its roots push up the ground around it. The Height Mod Width and Amount properties can create this effect. You can also modify the texture around the tree to blend in variations of a texture.

These distance fields are then combined as each tree stamp is processed, allowing for future stamps to perform filtering based on how close they are to existing trees.

Here we are creating ferns with a tree stamp, with a minimum and maximum distance from the tree's which came before it. This creates a ring of ferns around each tree. You'll also notice that the ground texture is changed to green around each tree- an extreme example, but this can be used to great effect when more subtle.	

Understanding SDF Processing order

You can filter trees, details, and objects from spawning in areas based on the SDF's of previous spawning operations. Each of these stamps has a range slider:



When you raise the minimum value above 0, you filter things from spawning close to other things. When you raise the maximum value above the minimum value, you filter things from spawning far away from other things. You can filter vs. Tree's, game objects, or use the parent objects distance field to filter against.

Note that distance fields are built in evaluation order, so if you have:

- Tree (oak trees)
- Tree (birch)
- Tree (poplar)

- Bushes

If the bushes filter against tree's you'd get a combined SDF for all the tree stamps processed before. However, sometimes you want to filter against a specific item, and you could arrange the hierarchy like so:

- Tree (oak trees)
- Tree (birch)
    - Bushes
- Tree (poplar)

In this case, the Distance From Parent option can be used to just filter against the birch trees.

If you've noticed, I use the term filtering to describe what happens- using spawners in this way does not "spawn around the tree", rather, what happens under the hood is positions are chosen for entire terrain based on the density and poisson disk settings- once those positions are chosen, filters (height, slope, sdf, etc) are applied to reduce that positions weight - any tree/object/detail which doesn't have sufficient weight is rejected.

Tree Modifier tips

The tree modifiers are run after the initial passes on the terrain, allowing you to modify the height and texturing based on tree placement. However, this can make them interact with the existing terrain in undesirable ways. For instance, a path might get raised up due to a nearby tree, and it might be desirable to have tree's very close to the path. To fix this, there is an option to occlude height or texture modifications on the path component.

Another issue you might have is when placing textures under trees near cliffs, the leaf texture which looks great under the tree doesn't look good when it goes onto the side of the cliff. There is an option which will filter the tree texture modifier with the stamps filters to prevent this.

Texture based noise and falloffs

Most people will reach for procedural noises and falloffs, but using a texture to define these can provide a lot of benefits, by providing for different signatures of noise and rougher falloffs than simple procedural shapes.

Importing external Splat Maps

If you have a splat map you want to import and use, you can do this by assigning it as the falloff on a texture stamp. Each texture stamp can map one texture layer onto the terrain, so to do a 4 channel splat map you would need to create 4 texture stamps, each pulling from one channel and assigning it one terrain layer. If you still want to apply falloff, you can instead put the map in the weight noise and still be able to set falloff to range and adjust it.

You could pair this with a height stamp to stamp externally created content from programs like world creator onto your terrain.

BindHeightFilterRangeToTransform

This component can be very useful when creating height based falloffs. For instance, a water level that you might want to move up or down. It allows you to bind the Y transform of your stamp to the height filter's minimum or maximum range, as well as optionally make the size of the filter stay consistently the same size when moving the stamp up and down.

Runtime:

By default MicroVerse components are stripped from the build and play mode. This means that there is no runtime code or data, and no performance cost. However, you can delete the SceneBuildStripping.cs file and everything will get included into the build and continue to operate in play mode. Note however that this is a very expensive system to try to use at runtime, and IMO only viable as some kind of level builder. You will also want to manage things like sync back to the unity terrain, saving, etc. In general this is an unsupported workflow, and the system can consume a lot of GPU memory, but you could make a pretty amazing level editor in your game with it.

Editor Performance

Ironically, the largest cost in the system is usually not from building the entire world from scratch on every edit, it's from sending that data back to Unity's terrain through its API, or simply rendering a large dense terrain in the first place. MicroVerse jumps through hoops to avoid the Terrain API whenever it can, but ultimately has to sync data back to it eventually. MicroVerse automatically culls stamps which have no effect on a particular terrain, but rebuilds the full world every time it updates. As such, using more but smaller terrains will usually help editing speed. Lowering resolution of terrain height maps and alpha maps will also help. And if you ever want to up-res your work you can simply change the resolutions on the terrain and adjust a property to force MicroVerse to rebuild the world from scratch.

A note on large world sizes

Unity's terrain system is very old, and was never designed for what I see many people trying to use it for. I'm always amazed at how many people start a project by laying out hundreds of terrains to create their "world", without realizing that Unity's terrain system is really designed to have only one terrain. The system was designed back when Unity was used to make small indie games, and while various improvements and extensions have been created to make it seem modern, in terms of performance it's just not designed for large scale/streamed worlds.

As an example, let's consider what happens when you create a terrain with 4k height and alpha maps. Unity keeps this data on the CPU and GPU because it assumes everything is dynamic. Further, it doesn't compress any of the data, so a 4k height map takes 32 megs, which means your 16x16 block of 4k terrains takes 8gb of data. That's larger than most AAA games are in total size, and it's just for your height maps. Then Unity uses an uncompressed RGBA texture for every 4 textures you have on the terrain - at 4k that's 64 megs of data, and if you have 16 textures on your terrain, that means each terrain consumes 256 megs of data for the splat maps, or another 16gb. So for that setup you're already at 48gb of data, and that doesn't include the rest of the terrain data, a single tree, or anything else yet.

Now let's consider rendering this data- Unity doesn't generate mip maps for any of this data, so that terrain way off in the distance is not only loading those 4k textures, but sampling them at 4k, utterly destroying the cache on the GPU, which samples from all over this texture even when the resulting terrain is tiny.

So my first advice is to consider if you really need a world of this size - can you even fill it with interesting content? Is your gameplay already built and actually requiring this, or is it all theoretical at this point? Do you really want to manage this much data on every change and build of your project at this early stage?

If you're really going to do this, consider writing a custom renderer and streaming system for this data instead of using Unity's, because getting unity to scale well to these kinds of sizes is going to be harder than just writing a custom solution. That said, you can still use MicroVerse and Unity terrains to construct it, but use a different, more scalable solution to load and render it. One that loads this data in small chunks into some kind of virtual texture page, so that the total data used for all the terrains is a fixed size regardless of how many you store on disk, and distant areas use proper mip maps to render fast. And if that's out of your capabilities, so is making this work well with Unity's system.

But if you're not going to take that advice and absolutely insist on trying to get this to work,, here are ways you can scale MicroVerse to work with more terrains.

1. Terrain Culling system
    a. MicroVerse has a terrain culling system which can be enabled in the Options of the MicroVerse component. When enabled, MicroVerse will automatically disable rendering of terrains outside of the scene camera's frustum or beyond a certain distance. It also provides a distance for vegetation rendering, and an easy way to adjust the scene camera's far clip plane. There is also a setting to control the maximum number of 'sneaky savebacks' MV will perform per frame, as these take about 70ms per 1k terrain. For comparison, MicroVerse can update 64 1k terrains with dozens of stamps on them in 10-40ms.
2. Work in lower resolution
    a. MicroVerse is mostly resolution independent. You can easily work with low resolution terrains, then use the sizing options on the MicroVerse component to up the resolution and it will regenerate the world in higher resolution. Note that the copy paste stamp when using the copy function copies at the existing terrain resolution, so you'd need to switch to a higher resolution to do any copy operation, then can switch back once it's done.

3. Seriously, save the giant world building for later and figure out what you need first. A giant world that causes all kinds of issues and slows development down is likely not what you want to solve for right now. And if you get to the point in which content generation is your real concern, build it out as you need to- you'll find that populating a giant world with interesting content is a lot more time consuming and difficult than you might realize, and that a small, dense, rich world is more compelling that a giant empty one.

How MicroVerse internally processes data

It can be useful to understand how MicroVerse processes data. Essentially all height map data is generated for all the terrains, then texturing, then vegetation, then details. This is because each of these passes relies on the result of the previous data for filtering and occlusion, and operations often need to sample from neighboring terrains to make operations seamless. In each of those passes, the data is ordered based on the order of objects in the hierarchy. So if you want an occlusion stamp to prevent vegetation in an area, you should place it before the vegetation stamp in the hierarchy.

This data is generated entirely on the GPU, and only sync'd back to the Unity terrain when absolutely necessary, or when MicroVerse detects that it might be able to do it without your workflow being interrupted.

Stupid Unity stuff

Unity's terrain system is really old, with a terrible API that they don't seem to want to fix. It also lacks many practical features which would make using it easier. One such example is detail rendering, which has a distance setting on the terrain. At whatever distance this is set, whole blocks of details will pop in and out of the scene, which is incredibly distracting. So what most shader authors do is fade the detail objects in the shader before that distance is reached. However, this means you usually need to adjust the shader and the terrain's detail rendering distance to work in conjunction, which may or may not be exposed by the shader you are using in the material. Ideally, if we could fix the terrain system, this would be published into a global shader variable so the user doesn't have to align all of this data correctly - but if you want to

render details further than the default you will have to set those distances both on the terrain and every material you use for details.

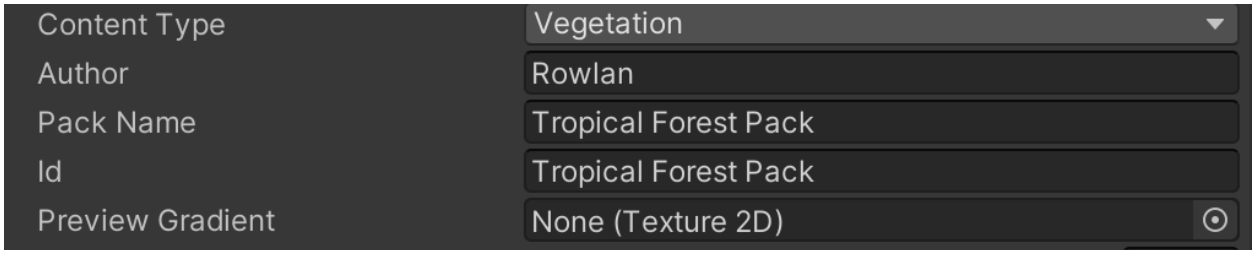
Also, Unity's detail system does not support LODs, while many asset store based vegetation rendering solutions do (such as Vegetation Studio and Nature Renderer).

Unity also emulates the available texture formats based on the platform you are on. This means that if you set the editor to iOS or Android, it will not be able to allocate an RGBAFloat texture (even though most of these devices now support this format). MicroVerse is then forced to use an RGBAHalf texture instead, even though it's an editor tool and the platform you're really on (Windows or OSX) fully supports this texture format (and ironically, so does the real target platform). This can cause some minor stair stepping of heights from spline paths, for instance. Thus for the absolute best quality, you want to do your final rendering of your terrain in windows or OSX (perhaps on a build machine).

Making your own content packs

Workflow for end users:

If you are not planning on distributing your content pack on the Asset Store, it's quite quick and easy to create your own content collections. To do so, first create a ContentPack scriptable object from the Create/MicroVerse menu in the project view somewhere in your project.



You will need to set the Content Type for this scriptable object, which controls which tab the content appears under in the browser. Give your pack an author name, pack name, and a unique ID.

Once this is done your collection should show up in the browser under the correct tab. You can then drag a hierarchy of stamps to the browser content window to add a new preset. Using the right click menu, you can rename the preset, or generate an icon for it which will be rendered from the current scene view.

Note you can also create new content collections directly from the content browser by selecting the tab and pressing the “Create Content Collection” button, and dragging existing content into the browser with your new collection selected.

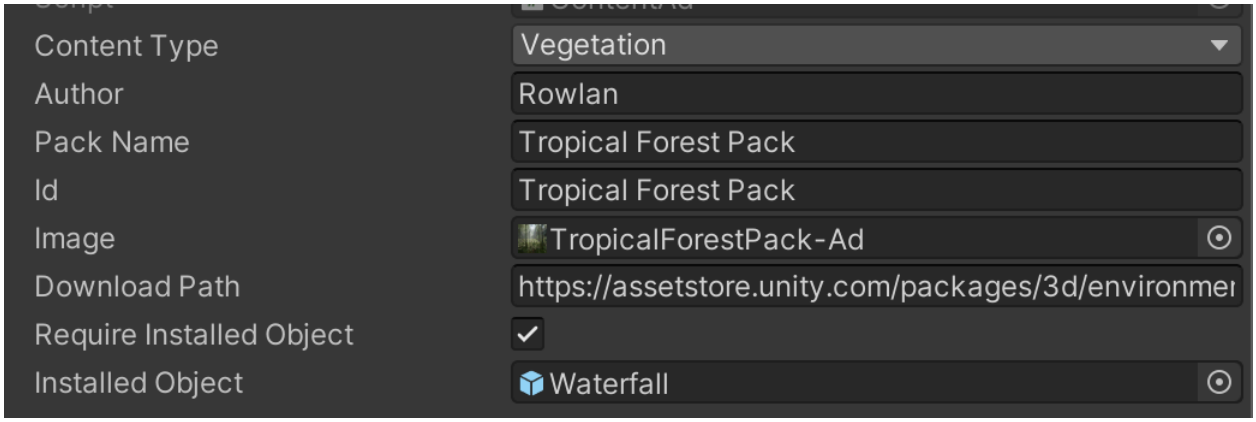
Workflow For distribution on the Unity Asset Store:

If your goal is to distribute this content on the Unity Asset Store, you will want to setup some additional details. If you are distributing height stamps, you should enter them into the content array on the content package manually instead of using prefabs, and you should assign the Preview Gradient texture, which allows the system to render them with a custom gradient in the browser. This gradient should go from left to right and represent the color from lowest terrain to highest - or just use one of the included gradients if you do not wish to provide a custom one.

For other types of stamps, you can use the quick workflow to create the prefabs as listed above, or fill out the details manually. There are also some other options in the content array you might want to use.

The Child Prefab can be used to get around a very specific use case. Let's say you are making a collection of presets for a content pack on the store, but you are not the author of that pack. Let's say you really want to have all the houses in that pack come with a clear stamp, but you can’t do that without modifying the original package. So what you can do is create a clear stamp prefab, set it as the childPrefab, and it will get instantiated as a child of the house which is set in the prefab slot.

Note that content packs get merged - so if you have multiple content packs with the same id and type both sets of content will show up in the browser. This makes it so we can ship some examples of presets with MicroVerse, then extend that library through other assets being installed.



A content ad is not needed unless you are distributing this to users who may not own the content. I include a number of these in MicroVerse, with the actual content packs shipping with the asset itself. If a content pack with the same ID is not found, then the ad will be displayed instead of the content, along with a download path which the user can click on. No affiliate ids are allowed in this url. Alternatively you can create content packs right in the project and use the "required installed object" for the check. When this is true, set the installed object reference to an object in the asset, and if it's not null then the ad will display. This variation lets us ship the content packs with separate assets or within MicroVerse and doesn't require the owner of the package to keep anything in their package.

Origin Shifting

While MicroVerse does not have a need for origin shifting at runtime, if your editor tooling is doing origin shifting, you can set the origin shift matrix with Shader.SetGlobalMatrix("_GlobalOriginMTX", originMatrix); and MicroVerse will offset world space noises match.

Scripting

Sometimes you want to script the editor in various ways. If you have changed something about the stamps, you need to call MicroVerse.instance.Invalidate() to cause MicroVerse to update. Note that Invalidate is a deferred operation, making sure that the actual modification is only performed once per frame. If you need to force MicroVerse to update right away, call

Modify directly - but note that changes coming from elsewhere (editor, other scripts) may trigger MicroVerse to update as well. If you need it to fully complete on a single frame, you call pass true, true to the function to force everything to update immediately.

Note that Invalidate takes a bounds, which can be used to cull the update area when working with large numbers of terrains. Note that the bounds of an edit should include both the new area of the stamp and the old area of the stamp. All bounds passed to Invalidate are combined to form a total bounds for all edits in a given frame (ie: When moving multiple stamps at once).

MicroVerse also has events for when the terrain begins being modified and is finished, though keep in mind that you also need to handle cancellation properly. Cancellation happens when a user adjusts a slider on a stamp, and the system can short circuit some operations because the data will be immediately invalidated anyway, speeding updates.

You can add your own stamp types as well without modifying MicroVerse, if you have very custom workflows that need to modify the terrain. These will need to inherit the interfaces for what they modify, for instance, IHeightModifier, ITextureModifier, etc. Looking at the existing stamps is the best way to understand the system.