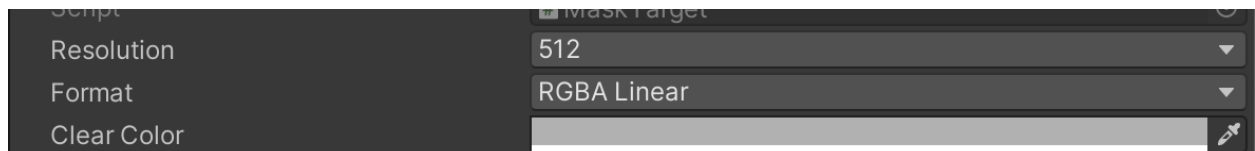


MicroVerse - Masks module

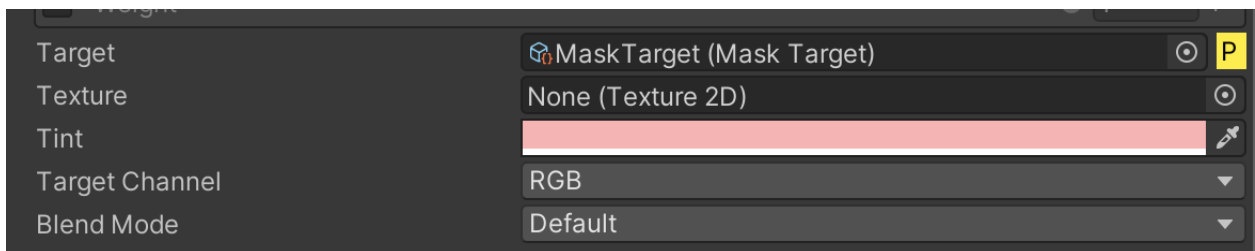
With the MicroVerse masks module installed, you can use MicroVerse to generate additional textures you might need. As an example, MicroSplat's shader supports things like global tint textures, masks to control where snow, wetness, puddles, streams, or laval appear, and other such effects. Systems which can be used to generate terrain stamps might output some of these textures as well, allowing you to use that data to compose a combined image, like any other stamp data in the system. Further, you can use existing MicroVerse filters, like slope or curvature maps, to filter these stamps as well.



To use the masking system, you first create a Mask Target scriptable object from the right click context menu (MicroVerse/Mask Texture). Once created, this will act as the container for all of the generated textures, which will be named based on the name of the terrain and mask. You can have as many Mask Targets as you need.

The mask target also lets you set the parameters for your masks. Resolution, texture format (RGBA in linear or sRGB color, or an R8 texture). You can also specify a clear color for the initial color of the texture. You can even use the SDFMask texture type to have your data turned into a signed distance field, giving you a rapid way to determine if something is near a feature in your game.

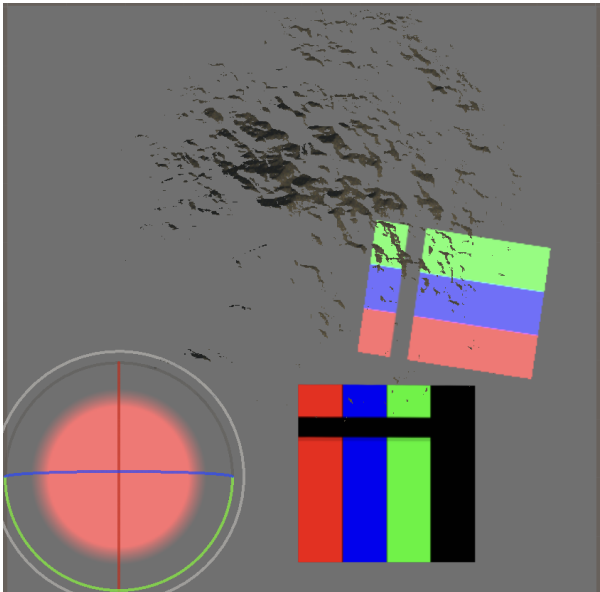
Next you can create a Mask Stamp from the MicroVerse stamp menu.

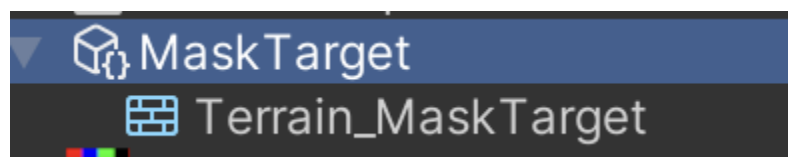


The stamp needs to target a specific Mask Target so it knows where to write textures. You can assign a texture for the data it will project into the target texture, which can also be

tinted. The Target Channel lets you decide which channels to write data into- for instance, an external system might have multiple things packed into one texture. For instance, MicroSplat's Puddles, Wetness, Streams and Lava module packs a mask for each effect into the RGBA channels of a linear texture, and by targeting the stamp at each channel, you can generate the masks for each effect.

Finally, you can set the blend mode for how this stamp should be blended with the existing stamp data.

	<p>In many cases, the data you might be rendering is not directly visible on the terrain. Note the small P button next to the target - if this is pressed, the resulting data will be rendered on the terrain using the preview system.</p> <p>This is an example of several mask stamps on a color texture.</p>
--	--

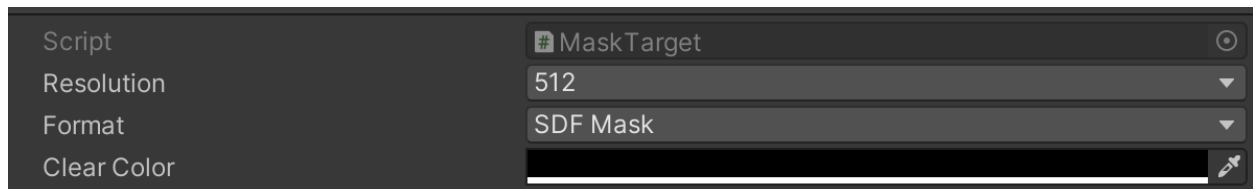


After saving the scene, a texture will appear under the mask target scriptable object. This can be assigned for use elsewhere in your project, and will be updated when your stamps change. Note however if you change the format of the mask texture it will replace this texture with a new one, breaking any references to it.

Creating an Water Area for sound effects

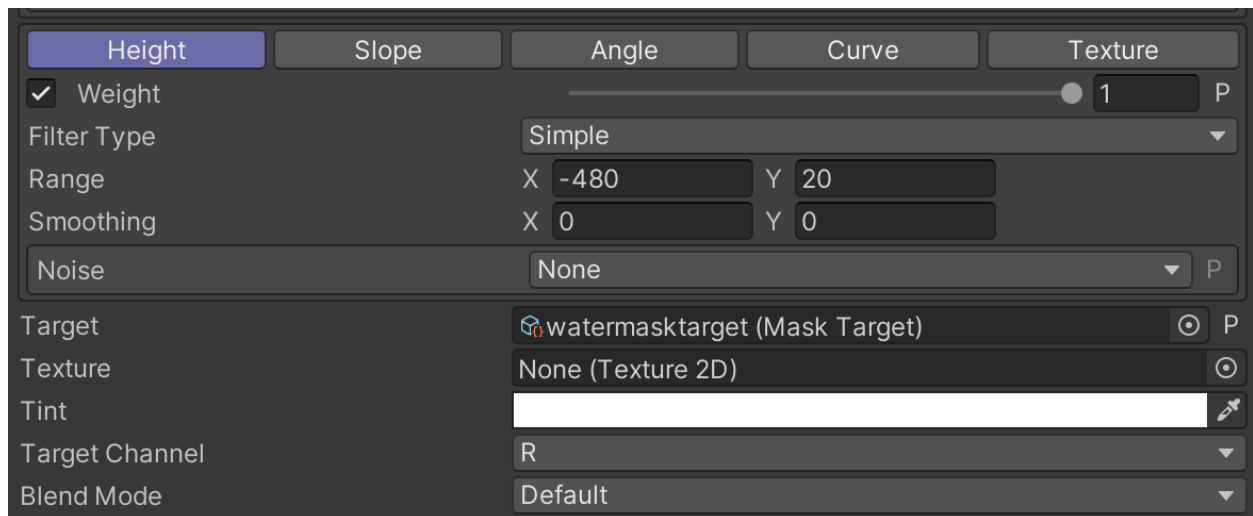
MicroVerse's ambient sound system can use basic shapes and splines to determine where sounds go - but this means creating a spline around your lake, which is inconvenient if you need to change the area of the lake. Using the MicroVerse mask system, you can generate a signed distance field texture for the ambient sound system to use to adjust sound levels as you approach water, automatically updating to wherever that water shows in your level.

To set this up, create a new Mask Target (right click in the project browser and select Content->MicroVerse->Mask Target) and set its resolution to something reasonable like 512 or 256. SDF textures can be very low res and still provide good results. Set the target type to SDFMask.

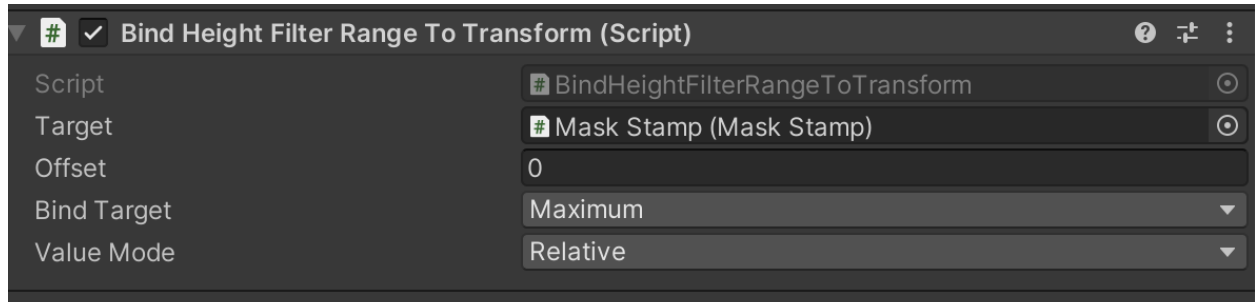


This will tell the rendering system that after rendering the mask it should convert the data into a signed distance field.

Create the mask stamp and set its mask target field to our new mask target. You can use whatever filters you need on your stamp, but since we're doing a water plane, let's turn on the height filter and set the smoothness of the filter to 0, 0. We can use the preview to see where it will appear.

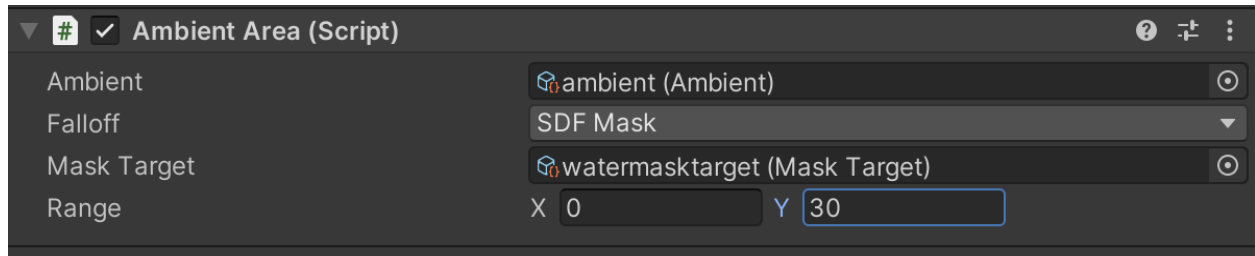


Finally, you can set the height of the height filter here, or use the BindHeightFilterRangeToTrasnform component to automatically set that based on the world height of the stamp, allowing it to update when you move the stamp.



Now create a new ambient scriptable object (again, from the right click menu in the project window) and set it up with a looping background water sound. See the main documentation for more on how ambient scriptable objects work.

Now create a new ambient area in the hierarchy (this is like a stamp for sound). Assign the ambient object to it. Change the ambient areas falloff to “SDFField”, and set the mask target to the one we created earlier. Set the falloff to something reasonable (say, 0 and 30).

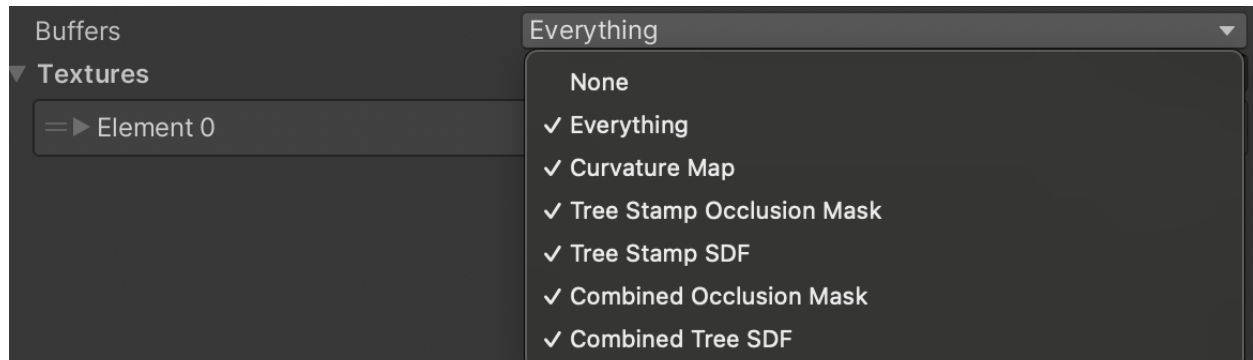


Now when you run the game, you will hear the sound of the water start fading in when you are 30 meters away from it. And if you adjust the water plane or create new ones in new areas, everything will just be adjusted automatically.

Extracting Internal Buffers

The masks module also allows you to extract various maps that MicroVerse internally generates, such as curvature maps for the terrain, the occlusion mask, or SDF data for tree stamps or all trees, which can be used for rapid lookups for gameplay operations.

To extract this type of data use the right click context menu to create a `BufferCaptureTarget` scriptable object. Then assign this object to the MicroVerse component.



Curvature Map - This is a single channel texture representing the curvature of the terrain, 0-1 from concave to convex. This is used by the curvature filter on stamps.

Tree Stamp Occlusion Mask - This is a single channel texture with a pixel set to the weight of the current tree, and black where there are no trees. The stamp will be named with the tree stamp's name on the end of it, so if you have non-uniquely named tree stamps only the last one will be exported.

Tree Stamp SDF - This is an SDF version of the Tree Stamp Occlusion Mask, where each pixel stores the distance to the nearest tree from that stamp. To get the distance in pixels, multiply the value by 256. To convert that into meters, you'll have to compute the pixel per meter of this map using the terrain's size and texture size. The stamp will be named with the tree stamp's name on the end of it, so if you have non-uniquely named tree stamps only the last one will be exported.

Combined Occlusion Mask - This is the occlusion mask for the whole terrain. **R** stores height stamp occlusion, **G** stores textures, **B** stores trees, and **A** stores details.

Combined Tree SDF - This is a combined SDF of all tree stamps.

Note that the first time MicroVerse updates all the textures for the output will be generated, which can take some time to do. After that, updates when making any edit are fast.

A sneakily powerful module

Looking at the masks module, only a few use cases might be immediately obvious. But it's quite powerful and applicable to all kinds of things in a game. For instance, using SDF generation, you can easily tell when certain types of features are nearby.

Perhaps you want to play sounds or particle effects when the user is in a snowy area? You can use the SDFMask feature to output a series of low resolution sdf masks for each terrain which say exactly how close to those areas you are, and trigger the appropriate effects accordingly.

Want to show some kind of alert when you are close to shrines, like in zelda? Have a mask stamp included with every shrine prefab, using it to generate SDF textures for how close you are to any shrine.

Check out the code in AmbientArea.cs GetFalloff() function to see how to use a world position to check the SDF field generated from a mask stamp.

Notes

MicroVerse stores the images in the scriptable objects you create as sub assets. This allows them to be stored and referenced by other assets in Unity, but avoids needing a lengthy import process that would make real-time updates of MicroVerse impossible. So while this has the upside of being automatically updated with any change while not breaking real-time updating, it has some downsides as well.

First, because the textures don't exist as a separate asset with a meta file, there's no way to change the format of the textures without breaking any references to the texture. In other words, if you change a MaskTarget's texture resolution or format, any material or script using that texture will have a missing reference and will need to have it reassigned to the new version. It is advisable, when possible, to have a script lookup the appropriate texture from the mask target object rather than linking it directly.

Second, because the textures are saved internally in uncompressed format, there's no easy way to easily compress them. That said, you can export the textures from these scriptable objects and have them written into the project as TGA files, which are imported like regular textures, compressed, etc. But these versions won't be automatically updated with every change.