Point Grass Renderer

Developed by Micah Wilder

Contents

Special Thanks

Ethan Dempsey at BitDepth - For creating the example ground textures
https://bitdepth.gumroad.com/l/

Compatibility

Render Pipelines

Pipeline	Compatible
Built-In	No
Universal	Yes
High-Definition	No

Tested Versions

Unity Version	URP Version
2021.2.2f1	12.1.1
2021.1.15f1	11.0.0
2020.3.23f1	10.7.0
2020.2.2f1	10.2.2

NOTE: When using a newer version of URP (12.0.0+), the *Render Face* setting of the *PointGrass_SHAD* shader will have to be updated to disable backface culling. It can be found within the shader graph asset at *Graph Inspector* > *Graph Settings* > *Universal* > *Render Face*

Mobile Support

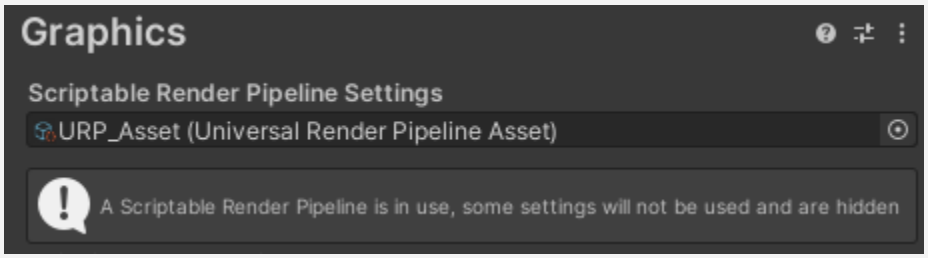
This package will run on mobile. However since this package is primarily designed to run on higher-end devices, performance is likely to be severely reduced. Tests on an Adreno 506 GPU with 5000 blades of grass ran at 9-12 FPS.

If you still wish to use mobile, I'd recommend using a low amount of blades and simplifying the grass shader.

Getting Started

Setup

Before using the point grass renderer, make sure your project is using the Universal Render Pipeline. You can go to *Edit* > *Project Settings* > *Graphics* and check if a pipeline asset is assigned.



If there is no pipeline asset assigned, refer to the URP Documentation for installation instructions

Demo Assets

This package comes with a few example assets and a scene that showcases the renderer component in action.

NOTE: The demo assets expect a *linear* rendering colour space. When set to *gamma* they will look much brighter than expected. If you're using *gamma*, you can either change to *linear (Edit > Project Settings > Player > Other Settings > Color Space)*, or darken the materials and lighting into acceptable ranges

The demo assets with different colour spaces - **Left:** Gamma **Right:** Linear

These assets are found at *Assets/Packages/Point-Grass-Renderer/Runtime/Demo.*

Point Grass Renderer

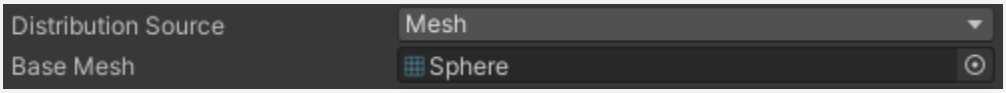
The Point Grass Renderer is the main component of this package. It is responsible for distributing and rendering grass blades, and has a variety of parameters to customise the look of the grass.

Distribution Parameters

The distribution parameters determine what the grass is distributed along and how the distribution is performed.

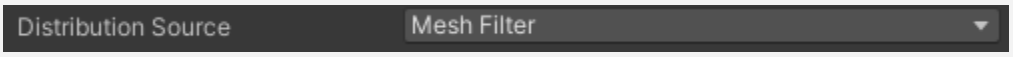
There are 4 types of distribution: **Mesh**, **Mesh Filter**, **Terrain Data** and **Scene Filters**.

Mesh



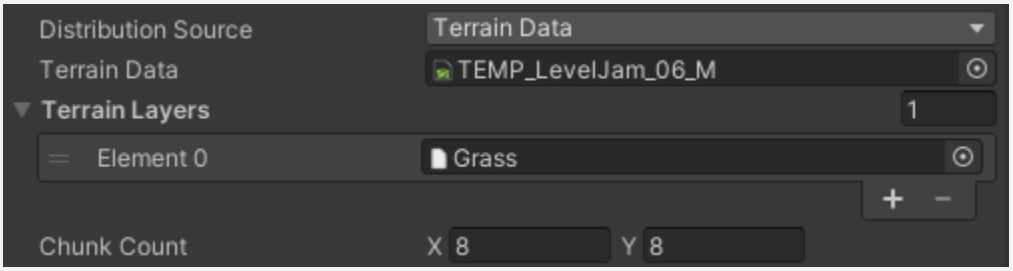
The renderer will use the assigned base mesh as the distribution source.

Mesh Filter



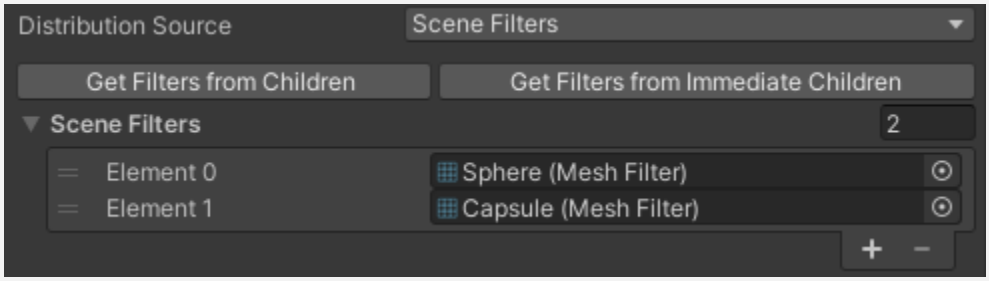
The renderer will use the mesh of a mesh filter attached to the game object. This can be useful if a mesh filter is using a mesh that doesn't exist as an asset.

Terrain Data



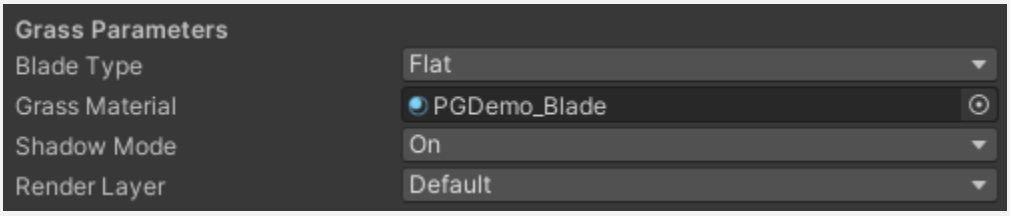
The renderer will use terrain data to distribute the points. The terrain layers are used to calculate the density of the points, and the chunk count determines the number of chunks the points will be split into. The blades will also receive colour data from the terrain that can be used to tint each blade (See *Use Blade Colour* in the shader parameters).

Scene Filters



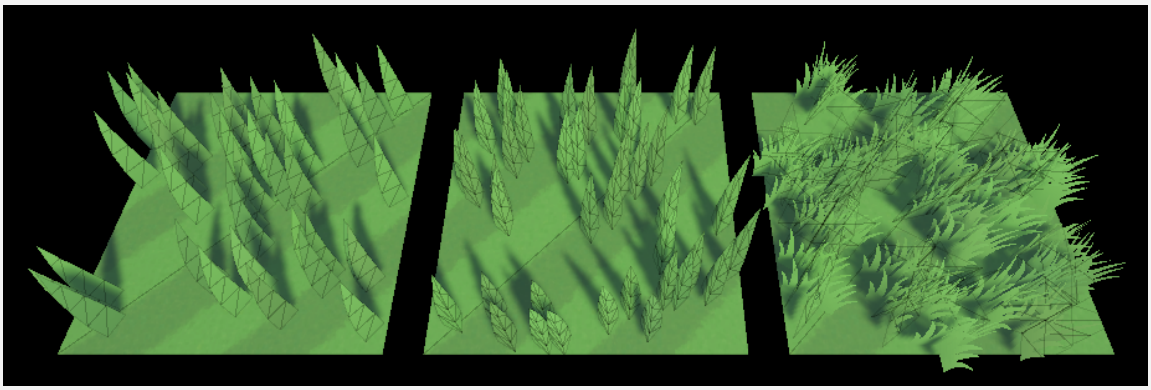
The renderer will use multiple mesh filters as the distribution source. The two buttons can be used to quickly retrieve all the child mesh filters. “Get Filters from Children” will use every child of the object, whereas “Get Filters from Immediate Children” will only use the object’s immediate children.

Grass Parameters



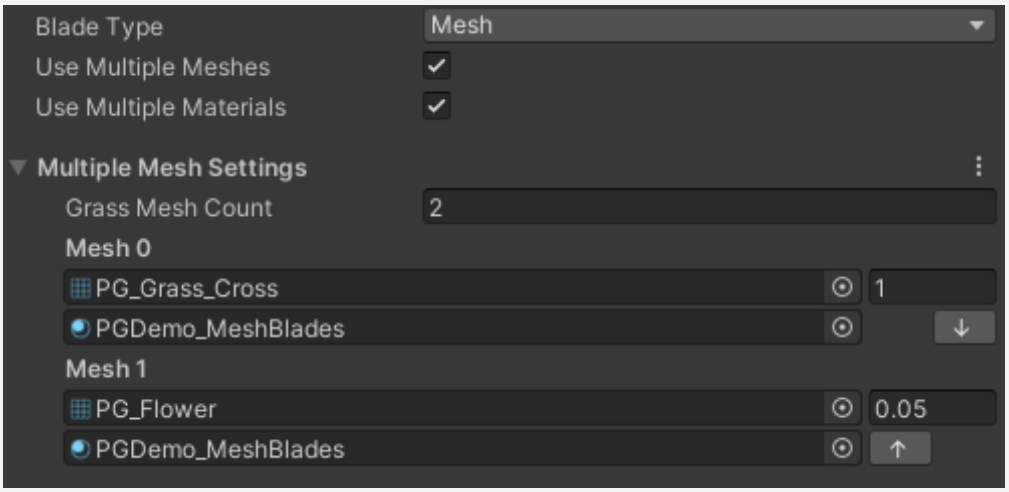
The grass parameters determine how the points are rendered, how they should cast shadows and the layer they will render to. The blade type controls what mesh will be used when rendering the blades (See below for more information), and the grass material is used for rendering the blades.

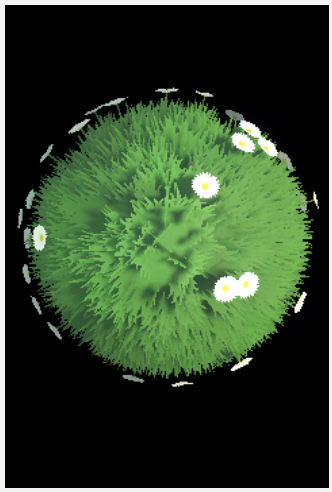
WARNING: When assigning the grass material, it is vital that you use a material that supports procedural instancing. If the material doesn’t support it, Unity will draw the blades without instancing, potentially creating thousands of draw calls.



There are 3 blade types: **Flat**, **Cylindrical** and **Mesh**.

Flat and cylindrical blades have the same basic shape, however cylindrical blades add two extra sides to help preserve their shape and shadows when viewed at extreme angles. This also means using cylindrical blades will draw 3 times as many polygons.



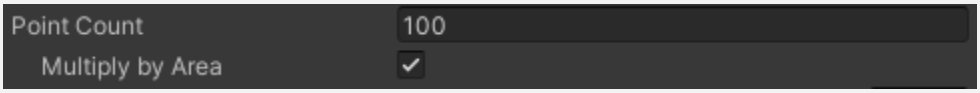


Mesh blades are the most versatile, allowing the use of multiple meshes and materials at the same time. When using multiple meshes, you can also adjust the proportional densities of each blade. The example above uses this to distribute grass and flowers on a single component.

Point Parameters

The point parameters determine the total number of points and how the points are generated.

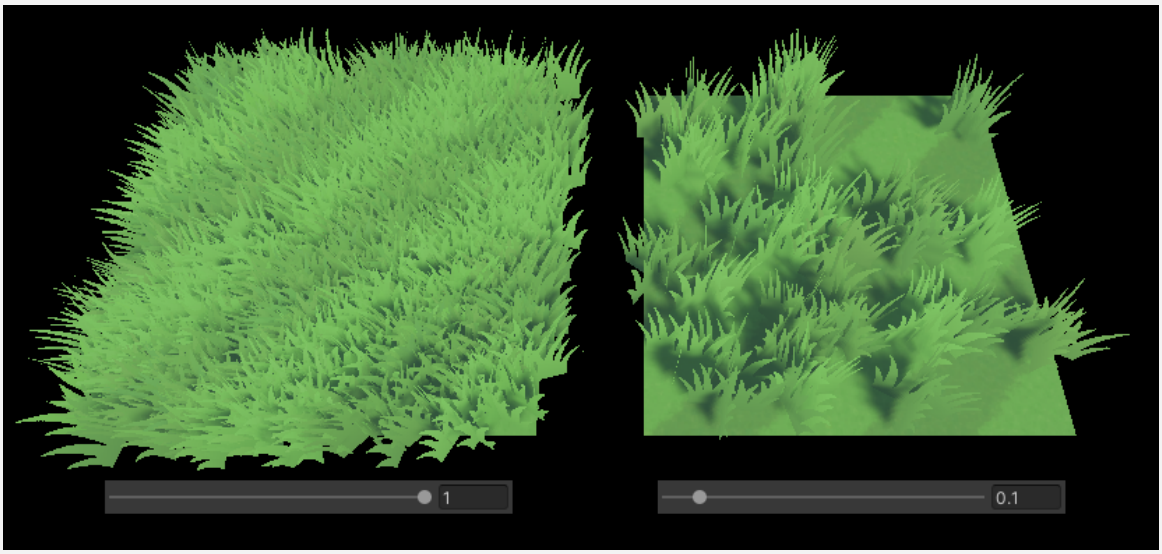
Point Count



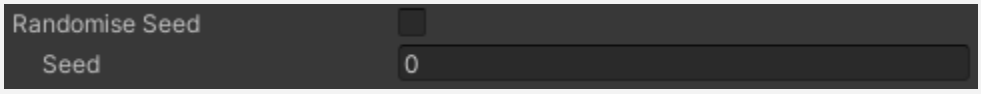
The point count determines the total number of points that are distributed. By enabling “Multiply by Area” it will scale with the distribution source’s total area. This setting is useful when using multiple renderers with different sizes for calculating similar densities.

LOD Factor

The LOD factor can be used to quickly change the number of points that are drawn. Note that this is only visual, and will not remove the distributed points from memory. This can be useful for reducing the number of blades drawn as they fill less screen space.

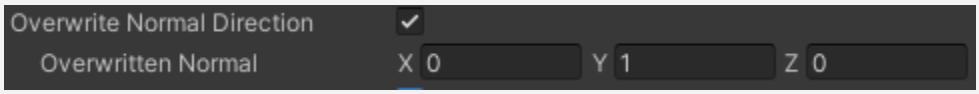


Seed



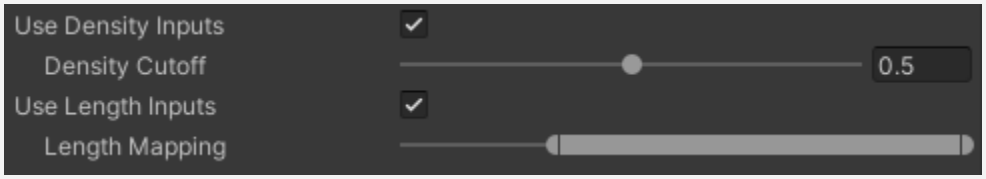
By default the renderer will randomise its seed when distributing points to generate different patterns. You can disable “Randomise Seed” and manually input a seed for a consistent pattern.

Normal Directions



If desired, you can enable “Overwrite Normal Direction” to supply a custom normal instead of using the normals of the distribution source. The normal will be normalized once the points are generated.

Vertex Inputs



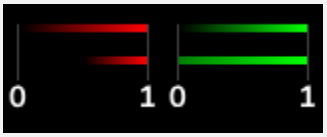
Density and length inputs can be used to change the distribution and size of the grass blades. The density cutoff changes the threshold value where vertices are considered as having no density. The length mapping defines a range that the length inputs will be remapped onto.

For meshes and mesh filters:

- Density is retrieved from the **RED** vertex colour channel.
- Length is retrieved from the **GREEN** vertex colour channel.

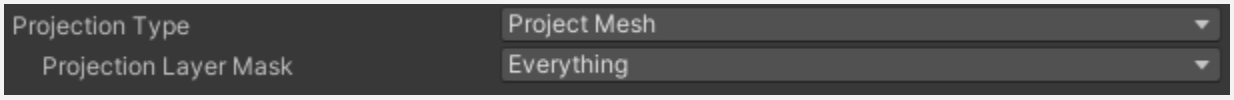
For terrain:

- Both the density and length is calculated based on the terrain's layers and the assigned terrain layers.



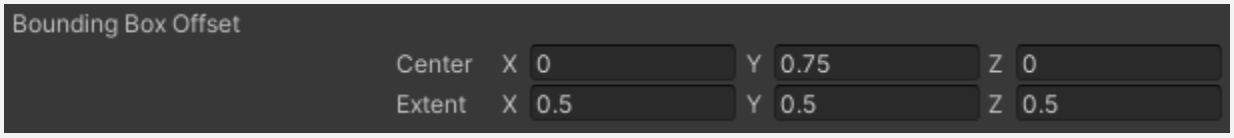
For a density cutoff of **0.5** and a length mapping of **0.5 - 1.0**, this image shows the final values for both with an input of **0.0 - 1.0**.

Projection Parameters



When enabled, the component will project the distribution source onto colliders in the scene. The *Projection Layer Mask* is used for masking out colliders on certain layers. By default it will project onto every collider in the scene.

Bounding Box Parameters

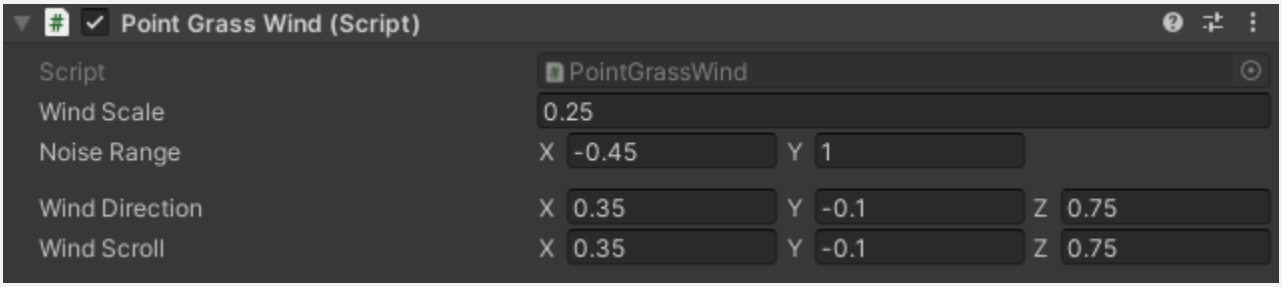


The *Bounding Box Offset* modifies the component's bounding box(es), allowing for more precise control. It can be seen in the scene view as a white box when the renderer is selected.

Point Grass Effects

In addition to the main point grass renderer, there are also a few effects for animating and interacting with the grass.

Point Grass Wind



The Point Grass Wind component sets wind parameters for the point grass shader. It's intended to be used as a singleton since it applies the parameters globally. To set it up, simply add the component to a game object in the scene.

Wind Scale

The world-space scale of the noise. Higher values result in more erratic bending behaviour.

Noise Range

Remaps the strength of the noise into this range. Useful for quickly adjusting the strength and range of the wind.

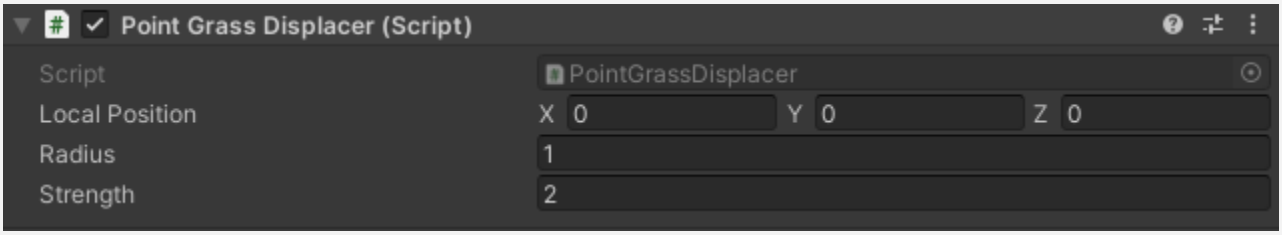
Wind Direction

The direction and magnitude the wind pushes the grass. This value is multiplied with the noise to calculate the final wind vector.

Wind Scroll

The direction and speed the noise will move in over time. Generally this should be close to the wind direction.

Point Grass Displacer



The Point Grass Displacer is a simple way of adding a displacement effect onto the grass. Simply add the component to a game object and adjust the parameters to set it up.

Local Position

The origin of the displacer. This is transformed from the game object's local space into world space before it is passed onto the point grass shader.

Radius

The radius of the displacer.

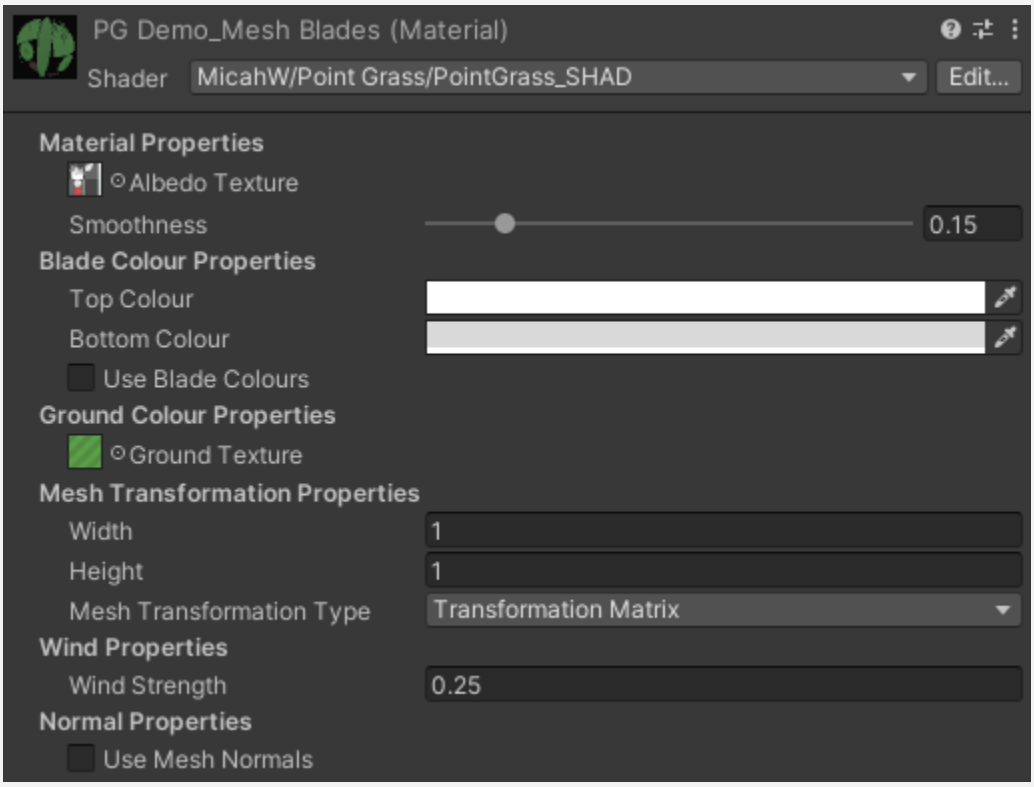
Strength

The strength of the displacement effect. Note that values above 1 can push the blades in strange directions

This component requires the use of a PointGrassDisplacementManager component, which is set up by adding the component to a game object in the scene. There can only be a single instance of the manager at all times, so any duplicate managers will automatically be destroyed.

NOTE: The upper limit on the number of displacers is 32. This can be modified within the *PointGrassDisplacementManager* script.

Point Grass Shader



The Point Grass Shader *(PointGrass_SHAD.shadergraph)* is the main shader used for rendering the grass blades. It has a variety of parameters for many use cases, including toggles to enable/disable various features. It has also been built using Unity's shader graph, which allows for easier modifications.

The shader uses the following vertex colour channels:

- **RED** - A mixing factor for the blade colour and ground texture.
  0 = no colour applied, 1 = full colour applied
- **ALPHA** - The bending strength of the blade & mixing factor for the top and bottom colours

Material Properties

The albedo texture is the base texture of the material. The smoothness parameter is uniformly applied to the whole material.

Blade Colour Properties

The top and bottom colours define a gradient tint applied to each grass blade. The blending is determined by the blade's alpha vertex colour. The *Top Highlight Amount* is multiplied into the top colour to brighten/darken the base colour. This parameter is particularly useful for helping the grass stand out from the underlying surface.

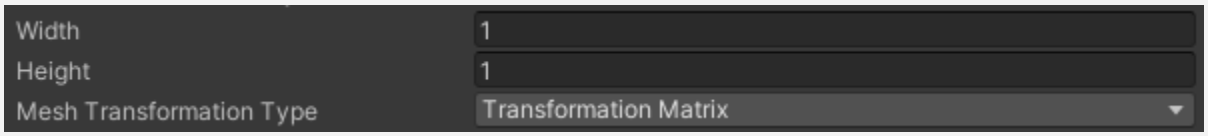
Different values of *Top Highlight Amount* - **Left:** 1.0 **Right:** 1.375

Use Blade Colour enables the use of sampled blade colours (Used with terrain distribution).

Ground Colour Properties

The ground texture is used to tint the grass blade. The UVs used to sample this texture are generated from the distribution source's UV map, or the world position from terrains.

Mesh Transformation Properties



The width and height control the scaling of the grass blades. This is more useful when using the flat and cylindrical blade types.

The mesh transformation type changes how the blade's vertices are transformed. *View Space Offset* aligns the blades to face the camera. Whereas *Transformation Matrix* calculates a matrix based on the blade's normal direction and uses that for transformation. This mode is ideal for meshes since the transformation is independent of the camera.

Wind Properties

This modifies the strength of wind effects. This is useful if different materials have an increased/decreased amount of stiffness.

Normal Properties

When using the *Transformation Matrix* mode, you can use the blade mesh's normals. This is particularly useful when scattering objects like pebbles or plants with large leaves.

FAQ + Troubleshooting

No points are being generated/rendered!

Start off by pressing “Refresh Points” at the bottom of the Point Grass Renderer component. Sometimes the renderer simply hasn’t had a chance to rebuild its point buffers and requires a manual refresh.

If this hasn’t worked it is likely one of the following issues:

- The component is disabled
- The distribution source is invalid/empty
- When using terrain and density inputs, the terrain may not have terrain layers matching the renderer’s layers
- The supplied blade mesh may be invalid/empty
- The supplied material may be invalid/empty
- The point count is too low
- The surface area of the distribution source is too low when using **Multiply by Area**
- The point LOD Factor may be too low
- The density cutoff is too high
- The distribution mesh’s **RED** or **GREEN** vertex colours may not have valid data when using density or length inputs
- The camera rendering the scene may not use the component’s render layer
- The bounding box may be too small

Why does the source mesh need to have read/write enabled?

In order to distribute points the mesh data has to be stored in CPU-addressable memory. If the mesh has read/write disabled, Unity will remove the mesh data from CPU-addressable memory after it has been uploaded into GPU-addressable memory.

Why don’t the points exactly match the terrain’s control/diffuse texture?

When generating points with terrain data, the renderer first has to convert the terrain’s data into an intermediary mesh that will be used for the final distribution. This uses the terrain’s heightmap resolution to determine the resolution of the resulting mesh.

Therefore, when the terrain’s control texture has a higher resolution than the heightmap, some of the data will be lost when sampling each vertex’s data. There’s a similar effect when the control texture has a lower resolution than the heightmap. This is because the terrain’s alpha maps cannot be sampled bilinearly easily, which results in a nearest-neighbour type of sampling.

This also means that fine details will be lost when sampling the terrain’s diffuse textures. Once again because the effective resolution is limited by the terrain’s heightmap resolution, and also because the textures used for sampling are downscaled to save processing time and memory usage.

Why do blades pointing straight down flicker erratically?

This is a complication when using *Transformation Matrix* as the blade transformation mode for the material. In short, the algorithm used to calculate the transformation matrix of each blade fails when the target vector (the blade's normal) is opposite the original vector (0, 1, 0). As a result blades pointing straight down can flicker from the vertices bending in incorrect ways.

The shader does attempt to mitigate this behaviour, but it cannot completely solve it.

What is the "Expected size in memory?"

It's the estimated size of the compute buffers based on the total point count and each point's size in bytes (56). It doesn't account for the renderer's parameters or any cached terrain data.