

Corgi Foliage Painter



Corgi Foliage Painter Documentation

latest documentation available [here](#)

Thanks for purchasing Corgi Foliage Painter.
For help getting started, please keep reading.

Dependencies

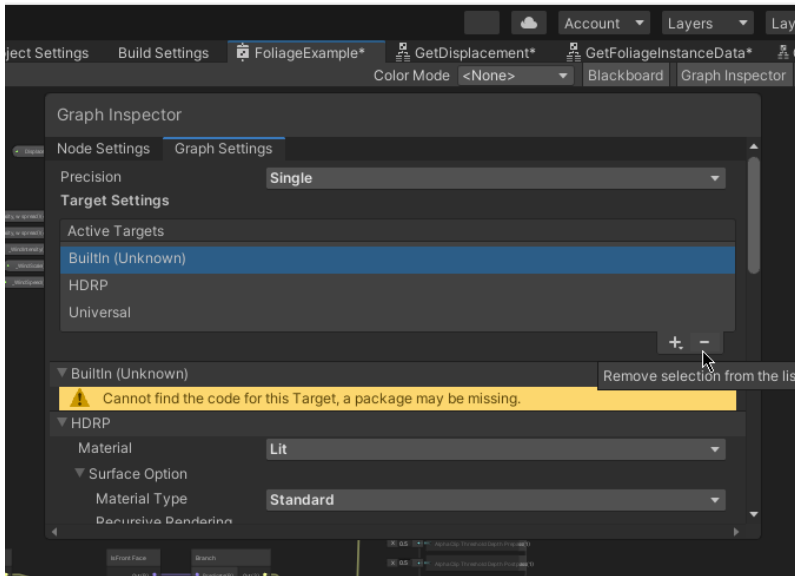
Please note, this plugin relies on a few things from the package manager. Here is the list of packages:

- Unity's Mathematics - `com.unity.mathematics`
- Unity's Burst - `com.unity.burst`
 - The plugin can function without Burst installed, as some functions will be considerably slower without it. It's highly recommended you install Burst.
- If you are using the included mesh assets, ShaderGraph - `com.unity.shadergraph`
 - Note: the included foliage mesh and demo assets rely on ShaderGraph for any of URP, HDRP, or BuiltIn to function. If ShaderGraph is not installed, the materials may be pink. Also, it will show a warning in the foliage painter window. You can permanently ignore this warning, if you do not need ShaderGraph.
 - The Editor gizmos are also using ShaderGraph shaders, so if you do not want ShaderGraph, you'll need to replace these with your own. `Invalid` is a simple unlit shader, `Sphere` is a depth-intersection shader which results in a subtle ring around where your mouse is.
- If you have not already done so, navigate to your project's player settings and enable `Allow 'unsafe' Code`
 - If you do want unsafe code in your project, you may disable this setting both in your project's player settings and in the assembly definition file for Corgi Foliage, and then navigate to **CorgiE.cs** and remove the line: `#define CORGI_ALLOW_UNSAFE_CODE` - this will make corgi take (slower) 'safe' code paths.

If these packages are not automatically pulled in, you may need to manually download them from the package manager.

Corgi Foliage Painter works with the built in render pipeline, URP, and HDRP.

Note for Unity versions at or older than 2020.3:



The ShaderGraph version shipped with **Unity 2020.3 and below does NOT support the built in render pipeline.** This is not the fault of the foliage plugin, but with Unity not supporting older unity versions. Either upgrade Unity, or go into the provided ShaderGraph example shaders and remove the built in shader pipeline to make this plugin work with URP or HDRP (it will show an error otherwise).

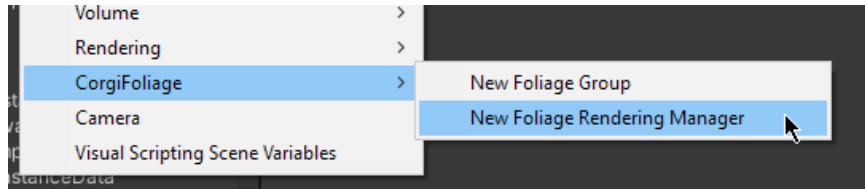
Also, Unity 2020.3 seems to have issues with instancing in ShaderGraph, where colors may not display properly. There is

no known work around, unfortunately. If you run into this issue, you may have to bite the bullet and upgrade to Unity 2021+.

Getting Started

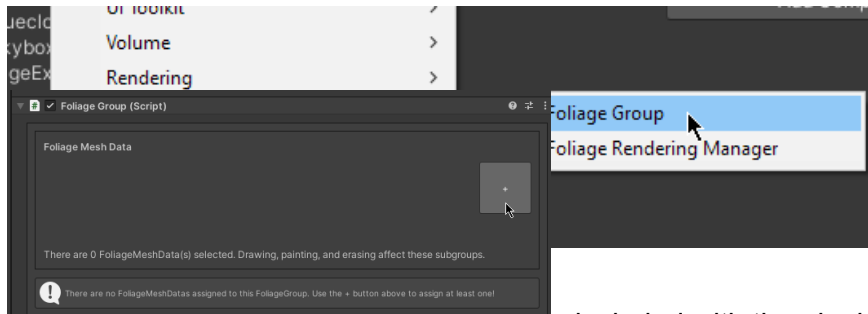
Check out the included demo scenes if you want to get painting immediately. For a fresh start, the very first thing you'll need is a **FoliageRenderingManager**. This manager needs to exist in any scene you want to have foliage in. You **MUST** have one and only one in the scene.

To create one: right click in the Hierarchy and select `CorgiFoliage` -> New Foliage Rendering Manager



Once this object exists, any Foliage Groups added to the scene will automatically register themselves to it, allowing them to render.

To actually create foliage, you must create a Foliage Group, right click in the Hierarchy and select `CorgiFoliage` -> New Foliage Group. Foliage Groups contain all data related to foliage rendering. You can have as many groups in the scene as you want, but remember that



each group is at least 1 draw call.

Now that a foliage group has been added, it needs at least one

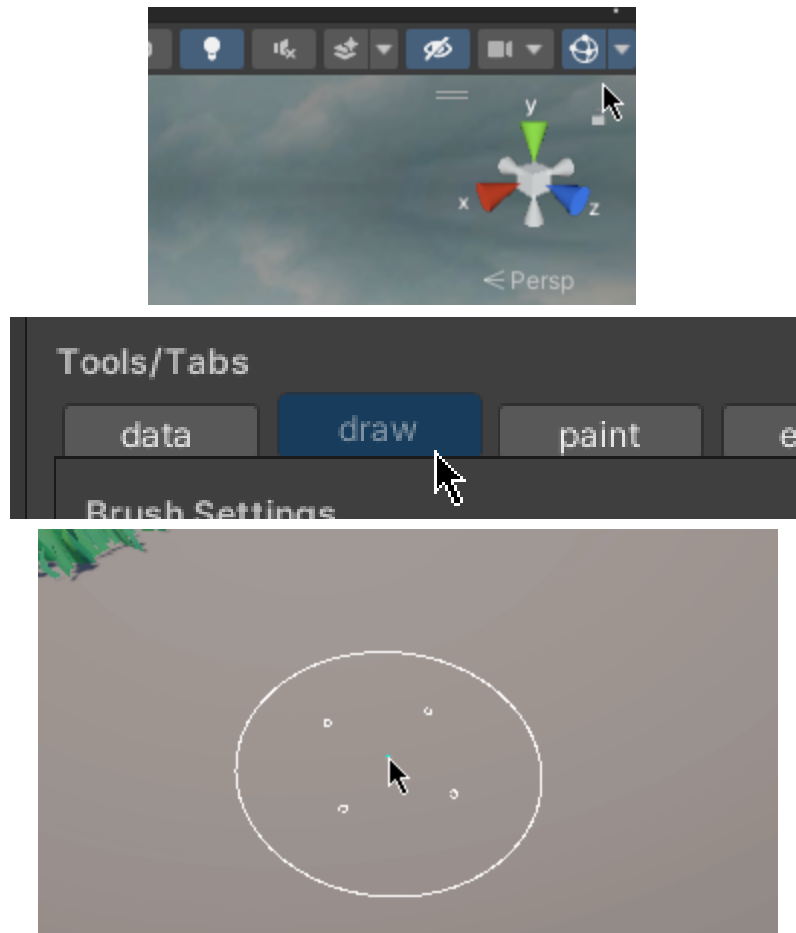
`FoliageMeshData` assigned. There are a few grass `FoliageMeshDatas`



included with the plugin. Hit the big + button to add one. If you imported the demo assets, you'll see a few in the object selection window that appears. Double click to add it to this `FoliageGroup`. Once it has been added to the group, you can click the image to allow drawing, painting, and erasing these meshes.

If you would like to customize these, click the “...” to open a context menu, which allows you to view the data file, remove it, and more.

Once at least one `FoliageMeshData` has been added into the `FoliageGroup`, and you have selected it (indicated with a blue outline), you can select the `draw` tab in the toolbar and start drawing in the `Scene View`! Note: to draw, you **MUST** have Gizmos enabled!



FoliageCameraHelper

Also! Your main game **Camera** MUST have a **FoliageCameraHelper** attached to it, or else foliage will not render in builds. You'll likely always see your foliage in the editor, because the scene camera will automatically register itself to the foliage system. But if you do not add this component to your game camera, builds will have no camera registered for foliage rendering, so no foliage will be visible.

Note: If your camera persists between scenes, but your foliage manager does not, *you may need to re-register your foliage camera helper to the foliage manager*. You can do this by simply toggling the enabled state of the foliage camera helper at runtime. For example: after a scene load.

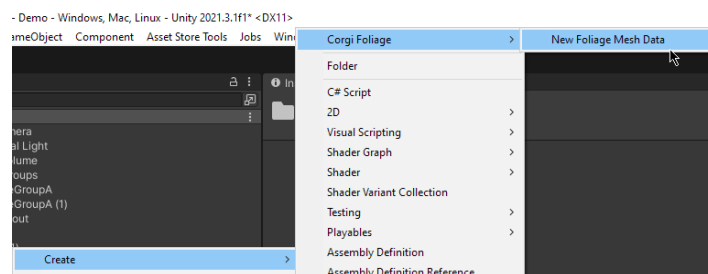
Foliage Mesh Data

`FoliageMeshData` objects contain rendering information for foliage instances.

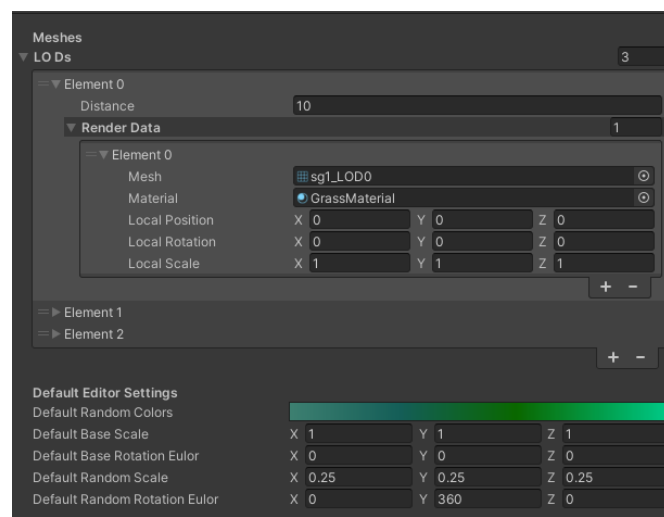
`FoliageGroups` can contain multiple references to these data objects, which are used by the `FoliageRenderingManager` to actually render them.

Think of `FoliageMeshData` objects as **Prefabs** for different types of foliage you want to paint.

If you imported the demo folder, you'll see a folder inside of there called `FoliageMeshDatas`. This folder contains some premade grass foliage assets for you to use!



If you want to create new ones, you can either simply control+d (duplicate) the existing ones, or use the right click menu in your project folder (create -> corgi foliage -> new foliage mesh data).



LODs

The `LODs` section allows you to drop in as many lod groups as you want, but you will typically only need 1 to 3.

The lowest indexed lod is the highest quality mesh, so descend this list by quality. For example, LOD0 (element 0) should be the highest quality mesh. LOD1 could be medium quality. LOD2 can be the lowest quality.

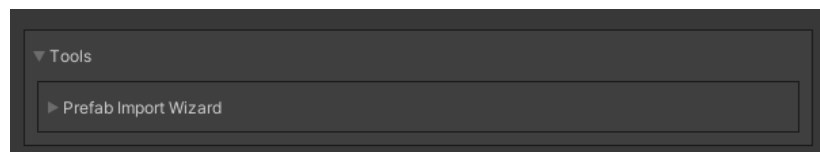
The distance variable in each lod is used to calculate which of the `LODs` to actually use. To be specific, the distance is calculated as the magnitude of the vector from the whole group's bounding box to the camera's position. If the distance between the bounding box and the camera is smaller than the distance of a particular element in `LODs`, that element will be used. If you need an example, please check out the Demo scene. The `RenderData` inside each `LOD` is a list of `Mesh`, `Material`, and local TRS offsets for that `LOD`.

Renderer Settings

These settings are used for actually rendering the foliage. Materials **MUST** have instancing enabled.

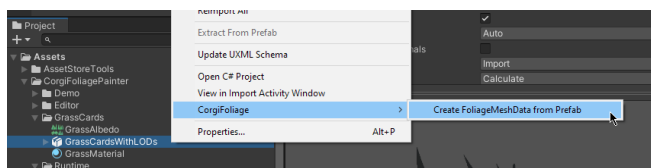
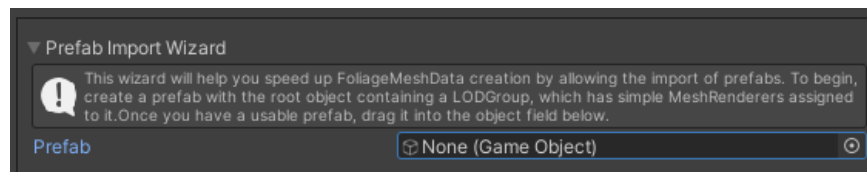
Tools

The Tools section in the inspector for FoliageMeshData is where you'll find any quality of life tools for managing FoliageMeshDatas. At the moment, there is only one tool: the Prefab Import Wizard.



Prefab Import Wizard

The Prefab Import Wizard is a tool which allows you to drop in a **prefab** which gets imported to populate the FoliageMeshData's LODs array for you. Follow the instructions in the import wizard to use it. The requirements for a **prefab** that can be imported are simple: any **prefab** which contains a LODGroup on its root. Only MeshRenderers will be imported.



Note: The Prefab Import Wizard can also be accessed by right clicking one or many prefabs in your Project view. This will use the default settings of the wizard and automatically create FoliageMeshDatas

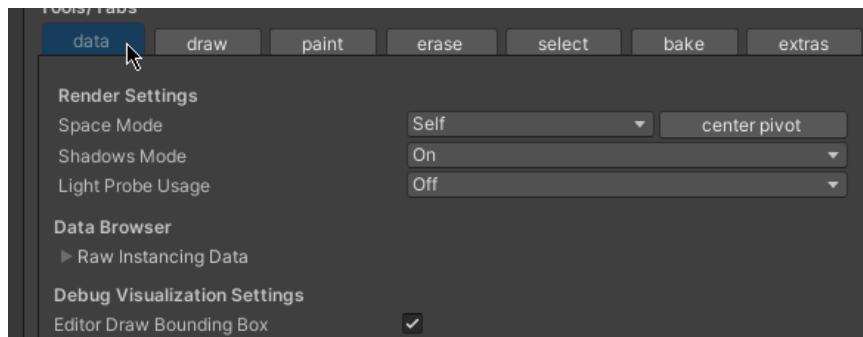
for each selected GameObject for you. They'll be placed inside the folder defined by FoliageResources's ProceduralMeshAssetPath.

Foliage Group

After creating any `FoliageGroup`, check the inspector. Notice that there are a few buttons at the top. These buttons will become available to you once at least `FoliageMeshData` assigned to this `FoliageGroup`. These tabs are your toolbar. Clicking draw will allow drawing in the scene view and also act as a tab in this foliage group's inspector.

Data Tab

The data tab contains the basics for configuring what you need to use the foliage group. It also contains a raw data browser, for debugging or fine tuning foliage groups.



Space Settings

Sets the space mode for the group. If the `SpaceMode` is set to `Self`, moving the Transform of the group will also move the foliage instances. If the `SpaceMode` is set to `World`, moving the Transform will have no effect on the foliage instances of that group.

Data Browser

The data browser is a tool for browsing, modifying, and deleting raw instance data of the foliage group.

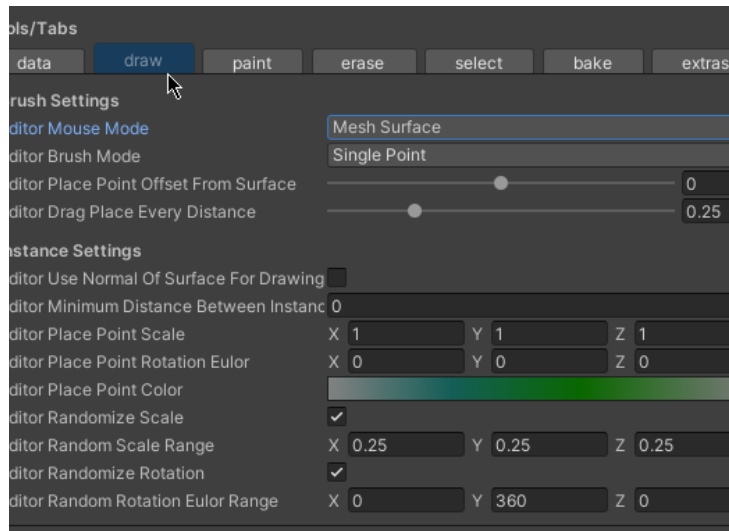
Debug Visualization Settings

When `EditorDrawIndividualBoundingBoxes` is true, draw the bounding boxes around foliage instances when the `FoliageGroup` is selected.

When `EditorDrawBoundingBox` is true, draw the bounding box for the whole foliage group when the `FoliageGroup` is selected.

Draw Tab

The draw tab will put the scene view into an edit mode. When viewing this tab, clicking or dragging in the scene view will attempt to place foliage instances based on the settings in the inspector.



Brush Settings

`EditorMouseMode` controls how the mouse position in the editor scene view is transformed into a world position. Some MouseModes will have some additional options that become available once you select them. Hover them in the inspector to view information about them.

`EditorBrushMode` controls how instances are created at the mouse position. Some BrushModes will have

additional options once you select them. Hover them in the inspector to view information about them.

`EditorPlacePointOffsetFromSurface` offsets the mouse position from the surface of whatever it's hovering.

`EditorDragPlaceEveryDistance` - When dragging to draw instances, this value is used to determine how far you must drag for a new instance to be placed.

`EditorBrushDensity` - When dragging to draw instances, this value is used to determine how many instances to place.

Instance Settings

`EditorPlacePointScale` controls the scale of newly drawn instances.

`EditorPlacePointRotationEuler` controls the rotation of newly drawn instances. This rotation is relative to the normal of the surface you are drawing on.

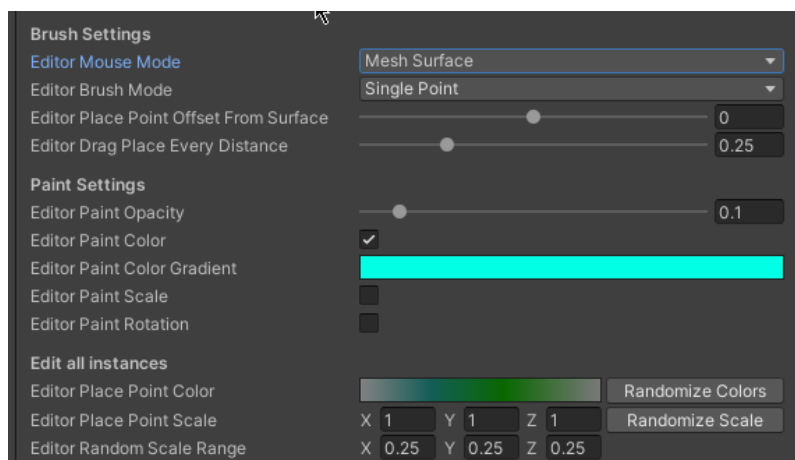
`EditorPlacePointColor` controls the color of newly drawn instances. This field is actually a `UnityEngine.Gradient`, which is randomly evaluated for each new instance created at the time you draw it.

When `EditorRandomizeScale` is true, `EditorRandomScaleRange` is used as a random delta on top of the drawing scale.

When `EditorRandomizeRotation` is true, `EditorRandomRotationEulorRange` is used as a random delta on top of the drawing rotation.

Paint Tab

The paint tab will put the scene view into an edit mode. When viewing this tab, clicking or dragging in the scene view will attempt to paint over existing foliage instances based on the settings in the inspector. Most of the brush settings are the same as the Draw tab.



Paint Settings

`EditorPaintOpacity` controls the intensity of paint brush.

`EditorPaintOpacityFeatherToPercent` - From this value to the outside of the brush, fade towards 0% opacity.


When `EditorPaintColor` is true, the paintbrush will affect color. The opacity is used to blend painted objects towards `EditorPaintColorGradient`.

When `EditorPaintScale` is true, the paintbrush will affect scale. The opacity is used to blend the painted objects towards `EditorPaintScaleTarget`.

When `EditorPaintRotation` is true, the paintbrush will affect rotation. The opacity is used to blend the painted objects towards `EditorPaintRotationTarget`.

Editing All Instances

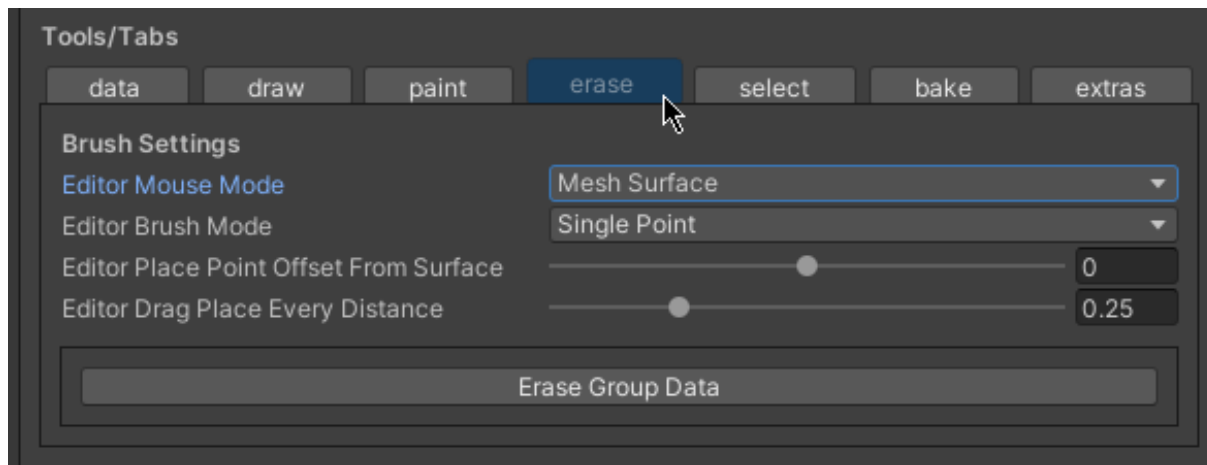
This section allows you to paint over all instances at once. It contains the fields from the drawing tab, but with some additional buttons for mass application of these settings.

Edit all instances							
Editor Place Point Color				<button>Randomize Colors</button>			
Editor Place Point Scale	X	<input type="text" value="1"/>	Y	<input type="text" value="1"/>	Z	<input type="text" value="1"/>	<button>Randomize Scale</button>
Editor Random Scale Range	X	<input type="text" value="0.1"/>	Y	<input type="text" value="1"/>	Z	<input type="text" value="0.1"/>	
Editor Place Point Rotation Euler	X	<input type="text" value="0"/>	Y	<input type="text" value="0"/>	Z	<input type="text" value="0"/>	<button>Randomize Rotation</button>
Editor Random Rotation Euler Range	X	<input type="text" value="0"/>	Y	<input type="text" value="0"/>	Z	<input type="text" value="360"/>	

Erase Tab

The erase tab will put the scene view into an edit mode. When viewing this tab, clicking or dragging in the scene view will attempt to erase existing foliage instances based on the settings in the inspector. Most of the brush settings are the same as the Draw tab.

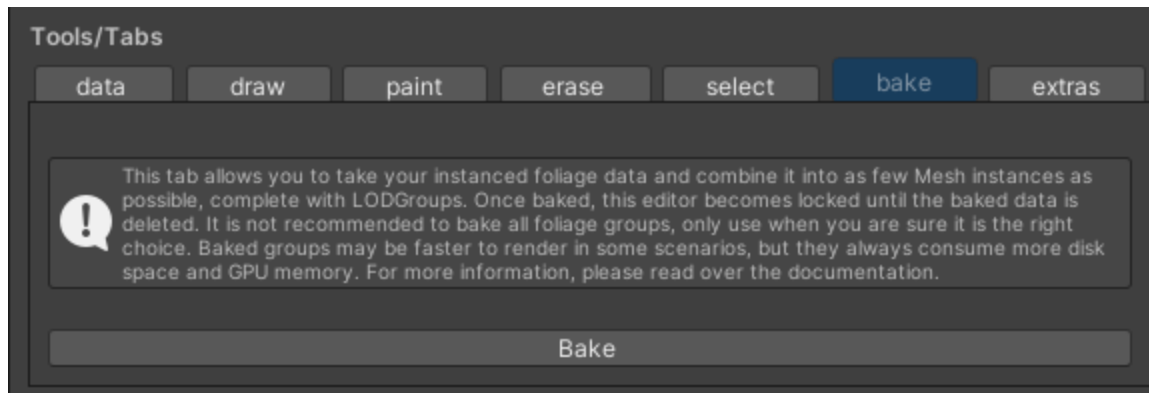
There is a big “Erase Group Data” button in this tab. When selected, it will remove all instances from the group.



Selection Tab

The selection tab allows you to select individual foliage instances and edit them in the scene view. You can change the position, rotation, scale, and color. This tab also allows you to delete individual instances.

Bake Tab



This tab allows you to take your instanced foliage data and combine it into as few `Mesh` instances as possible, complete with `LODGroups`. Once baked, the `FoliageGroup` editor becomes locked until the baked data is deleted. It is not recommended to bake all foliage groups, only use this feature when you are sure it is the right choice. Baked groups may be faster to render in some scenarios, but they always consume more disk space and GPU memory.

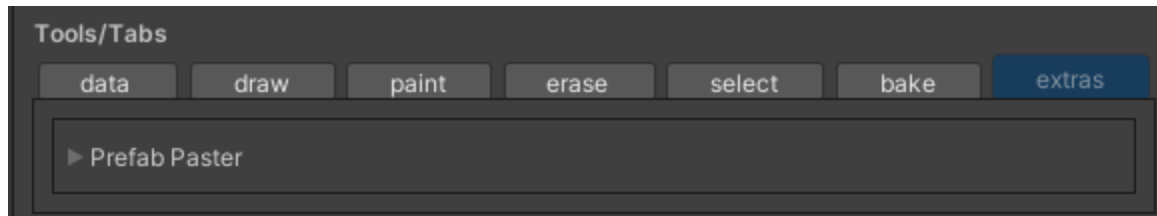
You may want to bake if your platform does not support instancing well, such as some mobile devices or possibly even on WebGL, if you want to support very old devices.

It may also be useful if you want baked lighting to take large foliage into account, although I do not recommend it as a general use case.

Only bake your `FoliageGroups` into `Meshes` when you are absolutely sure you need to, and you are absolutely sure you do not need to edit them again. If you change your mind and delete your baked data, the editor will allow you to edit foliage again, but you will lose all of your settings on the `LODGroup` and `Renderers` that existed as a result of the bake. Keep this in mind!

Baked meshes will be stored in the Assets folder of your Unity project. The exact location can be modified via the `FoliageResources` data file included with the plugin.

Extras Tab



Any extra tools that do not fall under data, brush, or bake categories will be placed into this tab. For now, there is a single tool: Prefab Placer.

Prefab Placer (Extra Tool)

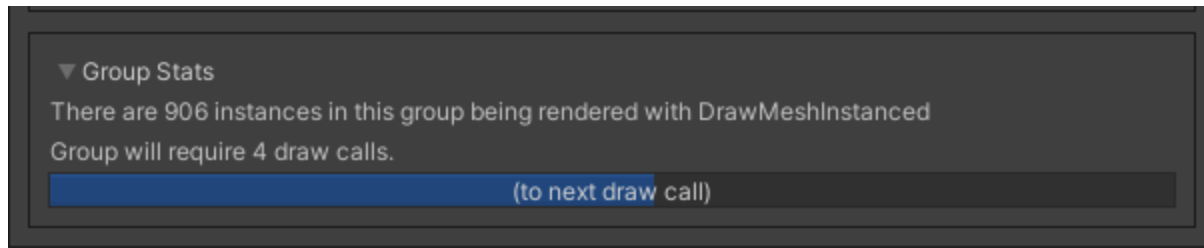


The Prefab Placer tool can be used to copy/paste a prefab onto every foliage instance. A common use case for this may be: a prefab with a trigger collider and script which reacts to players touching it.

If you have thousands and thousands of instances in your `FoliageGroup`, I do NOT recommend using this tool, as you can and probably will cause issues by generating too many `GameObjects` in your `Scene`. Use with caution! Settings for this tool are located on the `FoliageMeshData` objects.

Group Stats Section

This section is visible on all tabs. It shows some information on how the group will be drawn.



When the `FoliageRenderingManager`'s `RenderMode` is set to `DrawMeshInstanced`, this section will display the number of draw calls this group will require and a progress bar to the next draw call. To keep performance fast, you want to have a lower number of draw calls. Due to limitations of Unity's `DrawMeshInstanced`, you can only draw so many instances in a single batch at a time. The exact number varies by hardware, so this setting is stored in the `FoliageRenderingManager` and is reflected by this bar here. Each time the bar fills up, there will be a small performance overhead added on the CPU side of rendering.

Scripting API

The `FoliageGroup` class contains several useful functions, each of which has a small summary attached explaining what they do. The custom Editors were made using this public API, so hopefully anything the editors can do, you can also do with your own code, even at runtime.

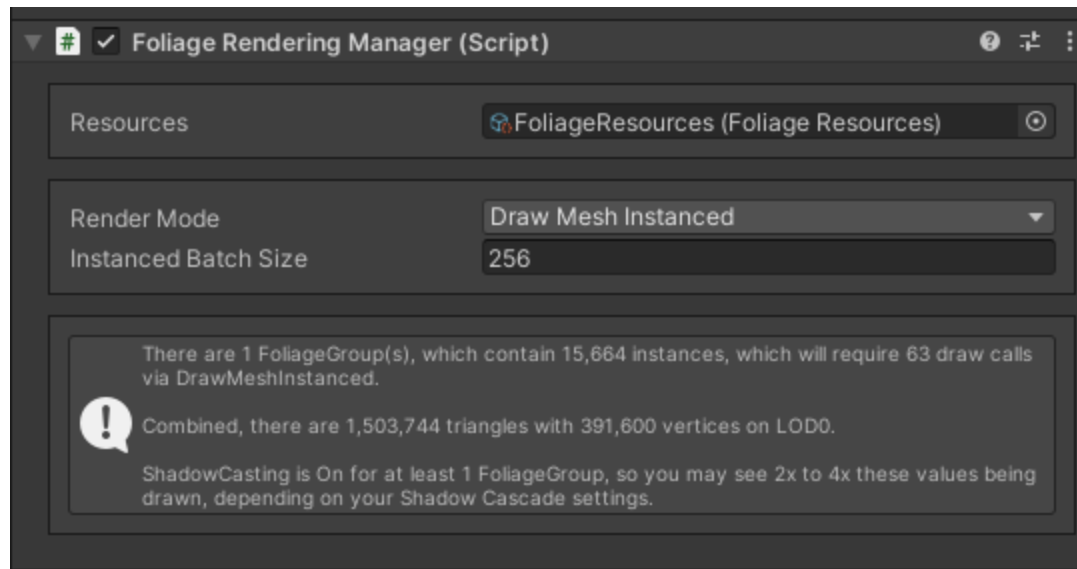
I've also included some helper functions which the editor does not use:

`GetAllFoliageData()` and `SetAllFoliageData()` - which are useful for saving and loading foliage data.

If anything is missing or even just confusing, please feel free to contact me! See the bottom of this document for more info.

Foliage Rendering Manager

This manager gathers the foliage groups in the scene and renders them.



Settings

`RenderMode` controls the method used for rendering. Due to a ShaderGraph bug in some versions of ShaderGraph, any ShaderGraph shaders may only be compatible with `DrawMeshInstanced` (sometimes referred to as direct rendering). `DrawMeshInstanced` has some limitations, such as `InstancedBatchSize` instances per draw call. It may require more CPU time, as groups will be automatically broken up into batches and their whole TRS matrix data will be submitted to the GPU every frame. I recommend using `DrawMeshInstancedIndirect` if possible for your project, but if it's not then do not worry about it. `DrawMeshInstanced` will allow you more control with Light Probe usage, as well.

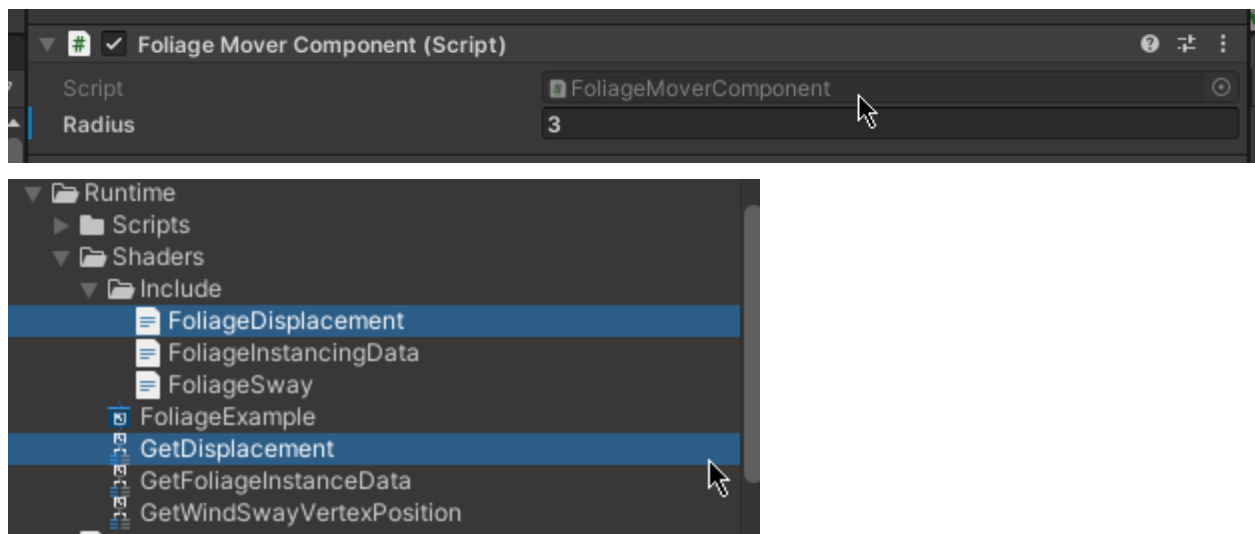
`InstancedBatchSize`- When using `DrawMeshInstanced` renderMode, batches within a single group will be split by this number. 256 is okay for PCs, but you may need smaller batch sizes for mobile devices. When using `DrawMeshInstancedIndirect`, this field will be hidden.

References

`Resources` is a global data block. If one does not exist, it will be created for you.

Foliage Displacement System

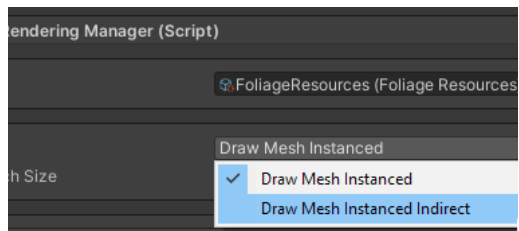
Corgi Foliage Painter comes with a simple and easy to use foliage displacement system, too! There is one helper component: `FoliageMoverComponent` - add this to any Transform to make it displace foliage. There is a ShaderGraph node `GetDisplacement` and also an HLSL include file `FoliageDisplacement.hlsl` you can use to access displacement data in shaders. Note: This feature is not supported on WebGL or any platforms that do not support Compute Shaders/Buffers.



Unlocking MORE Performance!

If you consider yourself a shader wizard, continue reading. If not, move along.

Corgi Foliage Painter is pretty fast. But, it could be faster. If you want to ensure `FoliageGroup`'s complexity is decoupled from (per-frame) CPU performance, you'll want to switch the `FoliageRenderManager`'s `RenderMode` over to `DrawMeshInstancedIndirect`.



Once you do this, depending on your Unity/ShaderGraph versions, you may notice that all of the foliage instances in the demo scene suddenly vanish. Before you start bashing your head against your keyboard, don't worry! I've gone through this suffering already and can spare you the pain.

ShaderGraph does NOT support procedural rendering through the `DrawMeshInstancedIndirect` graphics api, out of the box. However, I have managed to get it working for some versions of ShaderGraph. If it's not working for you on your version, you'll either need to upgrade Unity/ShaderGraph, or handle shaders entirely yourself.

If you really want some juicy juicy performance you're probably writing your own shaders from scratch anyways. So let's just get to it.

Custom HLSL Shaders

First off, `#include` these babies. You'll need to use a path relative to your Assets folder, so if you're an organized individual these include locations will need to be modified.

```
#include "CorgiFoliagePainter/Runtime/Shaders/Include/FoliageInstancingData.hlsl"
#include "CorgiFoliagePainter/Runtime/Shaders/Include/FoliageSway.hlsl"
#include "CorgiFoliagePainter/Runtime/Shaders/Include/FoliageDisplacement.hlsl"
```

Being as curious as you are, you'll definitely take a look in these includes and see what's up. There's a few comments and some helpful macros to make your life easier. You might even notice `FoliageInstancingData.hlsl`'s sneaky `foliageSetup()`. You're probably wondering what this does.

```
void foliageSetup() { }
```

Anyways, the first thing you're gonna want to drop in your custom shader after those includes is this pragma.

```
#pragma instancing_options procedural:foliageSetup
```

Then in your `cbuffer` block, which you're definitely using because as a performance-appreciating human you ARE writing SRP batcher compatible shaders, you'll want to use the macro `DEFINE_WIND_SWAY_PARAMS` - if you care about the wind sway I shipped in. If not, skip this part.

```
CBUFFER_START(UnityPerMaterial)
    // ~ your special snowflake properties here ~ //
    DEFINE_WIND_SWAY_PARAMS
    DEFINE_FOLIAGE_DISPLACEMENT_PARAMS
CBUFFER_END
```

Since we're dealing with instanced data, don't forget to include the unity macros for instancing shenanigans.

```
struct Attributes
{
    float4 position : POSITION;
    float3 normal : NORMAL;
    float4 tangent : TANGENT;

    UNITY_VERTEX_INPUT_INSTANCE_ID
}
```

```
};

struct Varyings
{
    float4 positionHCS : SV_POSITION;

    UNITY_VERTEX_INPUT_INSTANCE_ID
    UNITY_VERTEX_OUTPUT_STEREO
};
```

Once your attribute and varying are all good to go, you'll want to actually put some stuff in them. You'll need some more instancing macros smeared all over your once beautiful code. Once that's done, you'll rub in some of my tasty macros.

```
Varyings vert(Attributes i)
{
    Varyings o = (Varyings) 0;

    UNITY_SETUP_INSTANCE_ID(i);
    UNITY_INITIALIZE_VERTEX_OUTPUT_STEREO(o);

    float3 worldPosition;
    float3 worldNormal;
    float3 worldTangent;
    float3 objectCenter;
    TRANSFER_FOLIAGE_TRS(unity_InstanceID, i.positionOS.xyz, i.normal.xyz,
i.tangent, worldPosition, worldNormal, worldTangent, objectCenter);

    float offsetDistance;
    DO_VERTEX_DISPLACEMENT(objectCenter, i.positionOS.xyz, worldPosition,
worldPosition, offsetDistance);

    float3 windSwayWorldPos;
    DO_VERTEX_SWAY(i.positionOS.xyz, worldPosition, worldNormal, worldTangent,
windSwayWorldPos, worldNormal, worldTangent);

    // your other garbage
}
```

This `TRANSFER_FOLIAGE_TRS` is where the magic is. It's what you've been working up towards. You give it some floats and you get back some better floats. I think you get it.

```
TRANSFER_FOLIAGE_TRS(..)
```

If you wanna move around some grass and stuff, you better include this `DO_VERTEX_DISPLACEMENT` macro.

```
DO_VERTEX_DISPLACEMENT(objectCenter, i.positionOS.xyz, worldPosition,  
worldPosition, offsetDistance);
```

If you care about my wind sway stuff, you might want to include this `DO_VERTEX_SWAY` macro too. I recommend putting it in some kind of `#define`. I'm not your dad though, do what you want.

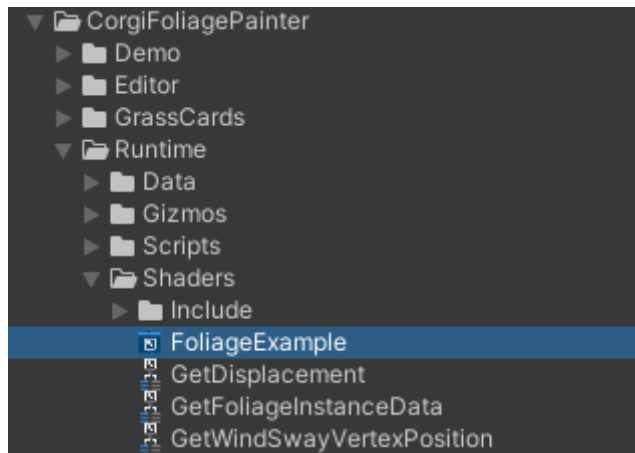
```
DO_VERTEX_SWAY(..);
```

After this you'll just need to set up the rest of the vert/frag stuff. I believe in you. You can do it. But if you can't, feel free to contact me with the info at the bottom of this document. I'll probably reply.

Good luck!

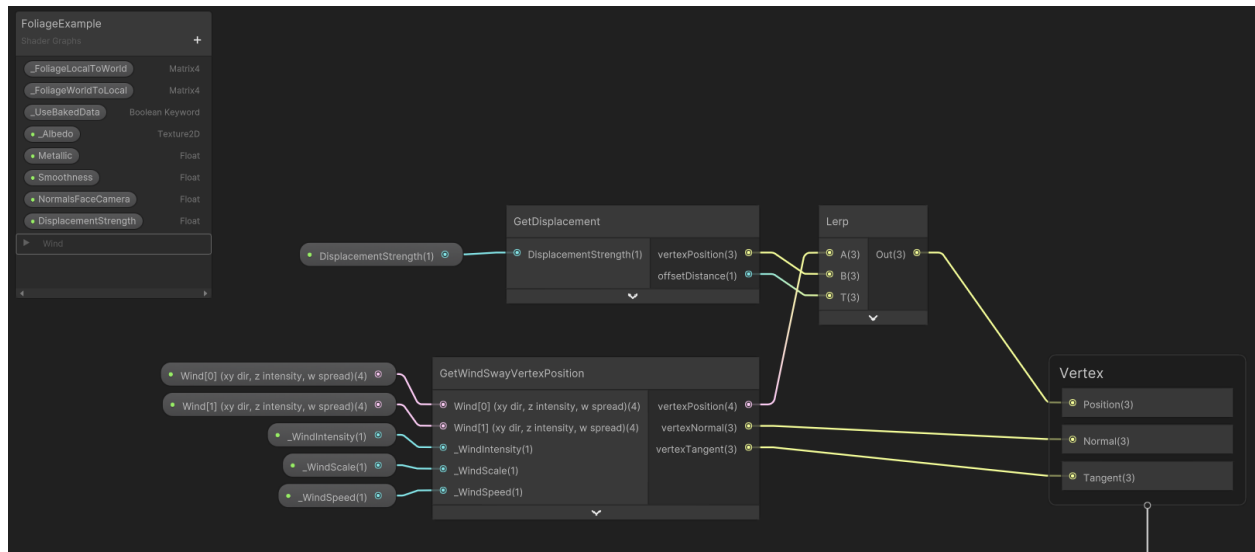
ShaderGraph Support

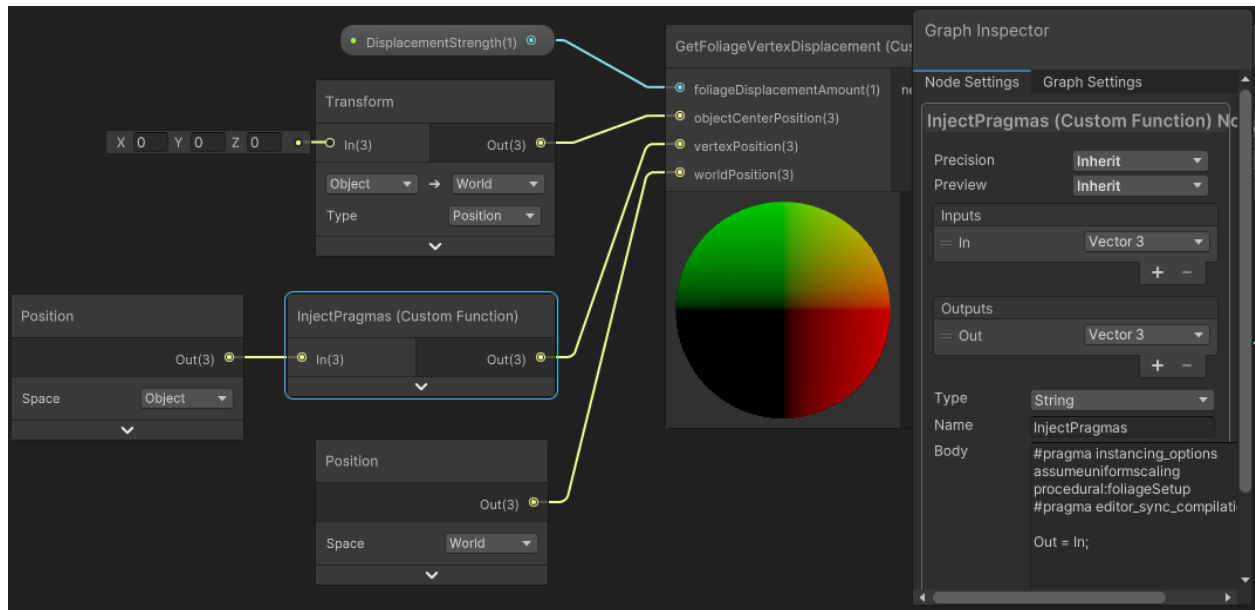
If you're on a later version of ShaderGraph, you may actually be able to get procedural rendering to work. There are some provided sub graphs you'll want to look at, and if you include them they handle pragma injection for you to get things working. I recommend just taking the provided example shader and modifying it for your needs!



As long as your ShaderGraph shaders include the following information, you can pretty much modify them however you want.

However, if you are writing your ShaderGraph shaders from scratch entirely, such as custom wind and displacement, you'll need to also handle the pragma injection stuff too.





The pragma injection secret is located in the `GetDisplacement` subgraph. Look at the `InjectPragmas` node. For procedural rendering to work, you **MUST** include a text only custom node like I've done here, to ensure the `#pragma` code ends up in the top level of the generated shader file, instead of inside of an `#include`. Here is the code, for easy copy/paste.

```
#pragma instantiation_options assumeuniformscaling
procedural:foliageSetup
#pragma editor_sync_compilation

Out = In;
```

Contact Me!

If you run into any issues or just need some help, feel free to reach out and contact me.

- You can email me at coty@wanderingcorgi.com
- Or you can hit me up on Discord (Coty#2845).
- There's also a support discord here: <https://discord.gg/n23MtuE>