Procedural Grass GPU Shader

& Terrain Tessellation Tools

Table of contents:

1: Introduction

This package includes two different modules:

- The first one is Terrain Tessellation Tools, which provides tools to easily create and organize an efficient terrain for your game. Its main purpose is to create, and slice simple meshes from heightmaps, to use for CPU-side purposes. (Like for example, collisions). And also use them as base for tessellation, which is the final result displayed on screen.
- The second one is Procedural Grass GPU Shader, which apart from being a procedural shader, it is also an implementation of a possible behavior of Terrain Tile from Terrain Tessellation Tools.

Basically, every tile has “Terrain Decorations” and each one with one Procedural Renderer assigned to it. This renderer uses Vector3 points to render the decoration in screen.

This is a Procedural GPU Grass Shader based on a Heightmap:

- Procedural: Instead of an animation shared to everything, each grass blade has random properties, that contribute to its animation and look differently.
- GPU: This is a GPU Instancing shader, meaning that the grass meshes only exist in the GPU, for much more efficient drawing. (Only at the cost that you can’t interact with it normally through CPU/Scripts/Game objects, with some workarounds.)
- Shader: Program in hlsl (usually) that draws colors on screen, among other things.
- Heightmap: square textures in grayscale, that are designed to depict in a value from black to white, the height of a certain point in the texture. Black is low, and white is high.

2: Terrain Tessellation Tools

Documentation

Terrain Tessellation Tools is another module in the package. If you want to improve your terrain, I recommend reading its documentation to understand it, located in its folder.

With it, you can create, slice, and tessellate meshes with a heightmap.

If you only want to use the grass shader, you can just read BirdViewUVCalculator.

Usage in Grass Shader

Even if you don't want to use Terrain Tessellation Tools for your terrain, if you want to use the Procedural Grass, you will at least need BirdViewUVCalculator and a heightmap to use it. (This script gives the height, for the points and grass blades. refer to Terrain Tessellation Tools Documentation).

If unsure how to set up the renderer, just look at a tile prefab or an example scene.

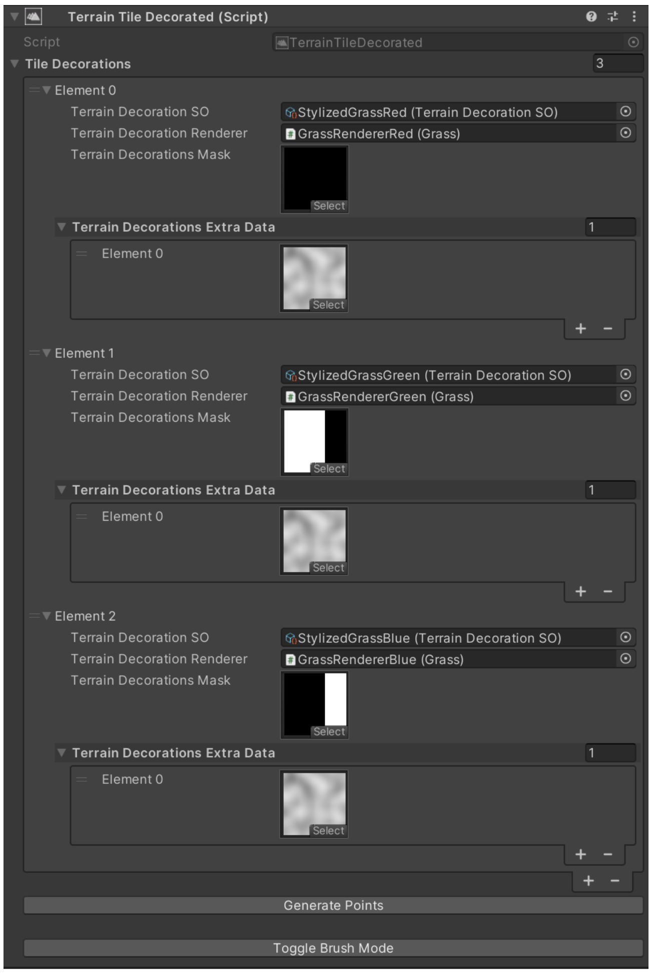
It's possible to make a workaround to give points to the grass renderer without a heightmap, but I still recommend you use heightmaps if your game will have a lot of terrain, as it will run smoother.

Also, the grass renderers benefit from being sliced in tiles, since renderers of a tile that isn't in view will turn off, releasing RAM buffer space.

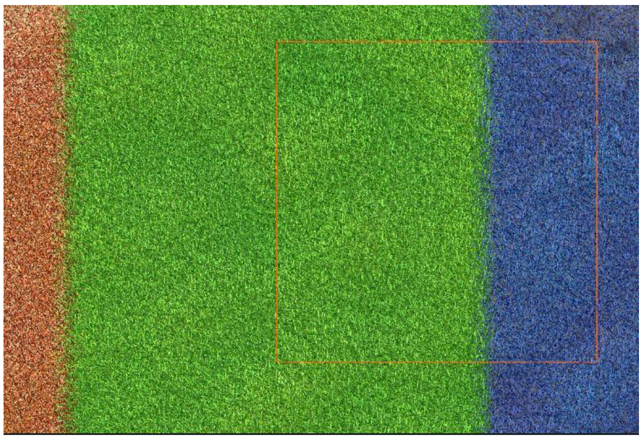
Modified scripts in this package

The Procedural Grass GPU Shader adds one script and modifies another of Terrain Tessellation Tools.

Terrain Tile Decorated



Terrain Tile Decorated (Script)
Script: TerrainTileDecorated
Tile Decorations: 3
Element 0
Terrain Decoration SO: StylizedGrassRed (Terrain Decoration SO)
Terrain Decoration Renderer: GrassRendererRed (Grass)
Terrain Decorations Mask: Select
Terrain Decorations Extra Data: 1
Element 0: Select
Element 1
Terrain Decoration SO: StylizedGrassGreen (Terrain Decoration SO)
Terrain Decoration Renderer: GrassRendererGreen (Grass)
Terrain Decorations Mask: Select
Terrain Decorations Extra Data: 1
Element 0: Select
Element 2
Terrain Decoration SO: StylizedGrassBlue (Terrain Decoration SO)
Terrain Decoration Renderer: GrassRendererBlue (Grass)
Terrain Decorations Mask: Select
Terrain Decorations Extra Data: 1
Element 0: Select
Generate Points
Toggle Brush Mode



This is a new class that inherits from the empty class Terrain Tile from Terrain Tessellation Tools.

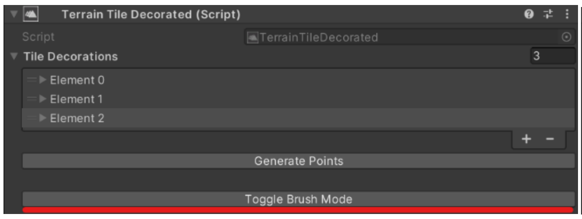
Its purpose is to hold all the information of all the decorations on the tile, and make use of a custom editor brush to easily modify the points of all renderers all in one place. Also, you can generate the points of all decorations with a simple click in the button bellow.

It holds a single list, with Tile Decorations. Each decoration is comprised of:

- Terrain Decoration SO: Scriptable object that defines each decoration. Refer to its page in Content.
- Terrain Decoration Renderer reference. (usually, a child in the same tile)
- Terrain Decoration Mask: a texture to know where and where not to create points. Black is none, White is 1 every spacing distance in the Terrain Decoration SO, and Grey (1/2) is half the points at that spacing, Etc.
- Terrain Decoration Extra Data: Depending on the renderer, you might need more data. (in Grass it's the clump ratio).

Terrain Tile Decorated (Brush Mode)

If you create your masks textures, but the result is not exactly how you wanted, or you simply prefer to draw in the terrain, you can modify the points with the Brush Mode.

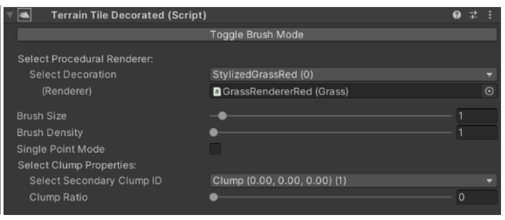


Terrain Tile Decorated (Script)

Script: TerrainTileDecorated

Tile Decorations: 3

- Element 0
- Element 1
- Element 2



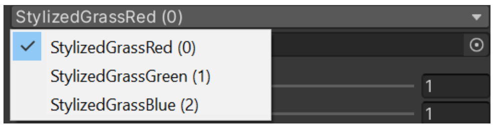
Terrain Tile Decorated (Script)

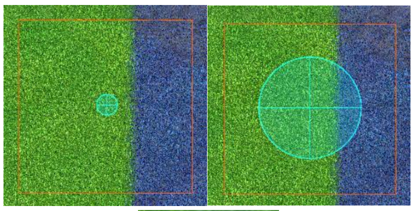
Select Procedural Renderer:
Select Decoration: StylizedGrassRed (0)
(Renderer): GrassRendererRed (Grass)
Brush Size: 1
Brush Density: 1
Single Point Mode: ☐
Select Clump Properties:
Select Secondary Clump ID: Clump (0.00, 0.00, 0.00) (1)
Clump Ratio: 0

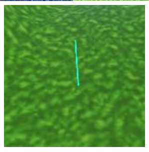
Then, if you have gizmos on, your handles will be disabled, and you will be able to Draw with the brush (I recommend enabling always refresh for this).

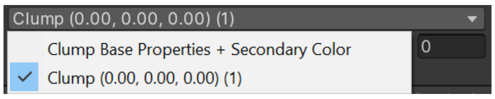
For drawing, you need to set up the brush:

- Select the decoration to draw: (These are from the Tile)
- Select brush size (diameter) and density (points added per frame) be careful with high density or holding the brush in the same spot. Or you will have much more points than needed in one place.
- If you want, you can add a single point with single point mode.
- If Grass, select Clump Point to define as Secondary, and its ratio. Refer to its page in Shader/SO. (Defaults to second, since first is base properties)





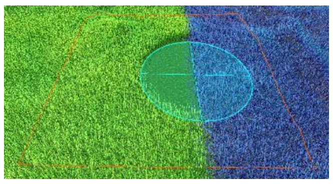




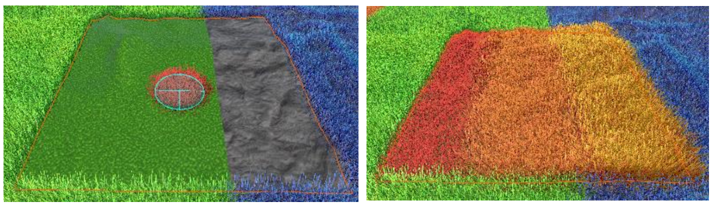
Clump (0.00, 0.00, 0.00) (1)
Clump Base Properties + Secondary Color
✓ Clump (0.00, 0.00, 0.00) (1)
0

To Draw, simply use left click. If you aren't using single point, you can also erase with right click.

Right click with Green selected (it only erases Green):



Holding left click with Red selected, 0 ratio:
& Drawing Red, with 0, 0.5, 1 Tall grass ratio:



Mesh Slicer Tiler (MeshToTerrainTiles)

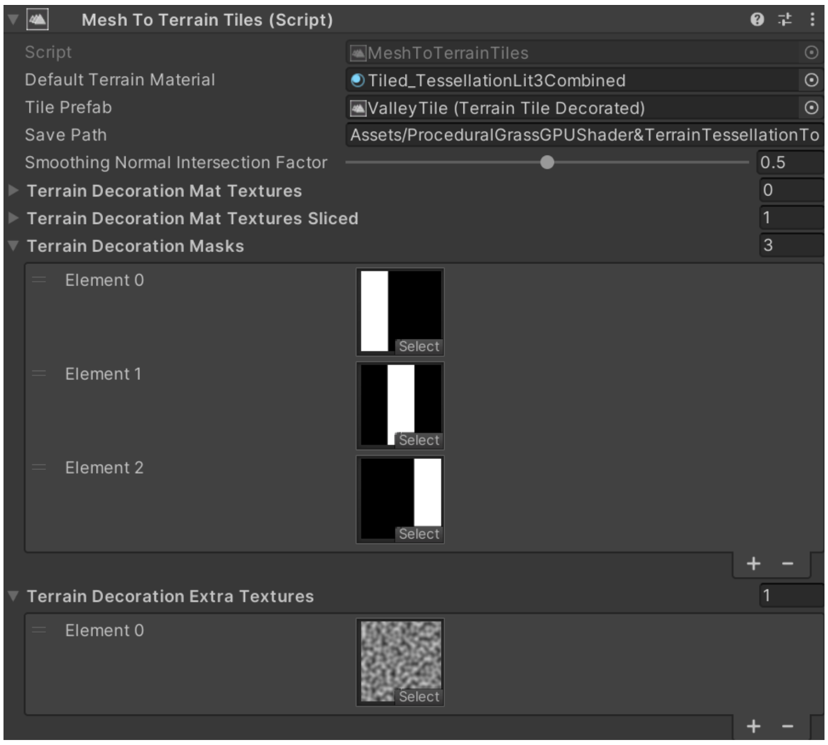
This is mostly the same file from Terrain Tessellation Tools, but has been modified to allow textures for Terrain Tile Decorated slicing, and generate the decoration points as soon as they are sliced.

(The file with the same name has been removed from the Terrain Tessellation Tools folder, and moved to the Procedural Grass folder)

One is for the masks, and the other for the extra data. Both are sliced properly, and assigned to their proper tiles and decorations.

Masks are in order (meaning Element 1 will be for the first decoration, Element 2 will be for the second… etc.) You should use the same number of textures than the number of decorations on the tile prefab.

Extra textures are sliced, but these are shared to all decorations, since you might need more than one per decoration, and it might be useful for the decorations to share this data. (meaning Element 1 will be for the first extra data for all the decorations, Element 2 will be for the second extra data for all decorations… etc.)



Mesh To Terrain Tiles (Script)

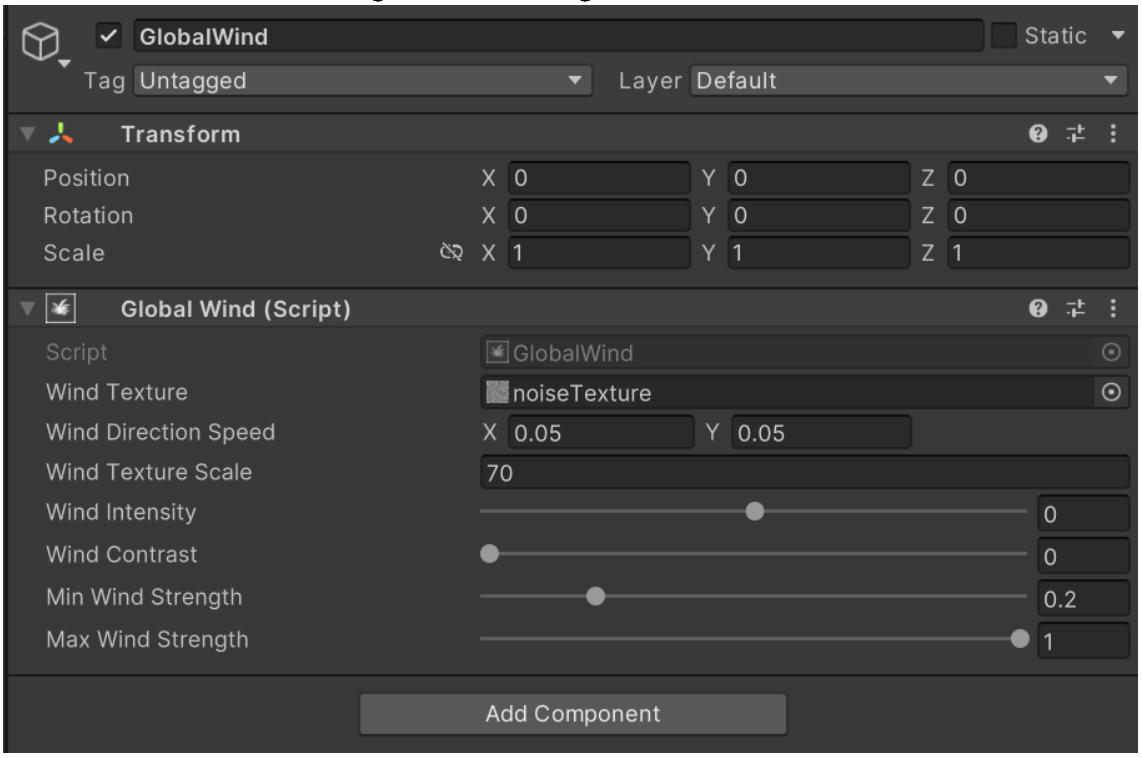
Script	MeshToTerrainTiles
Default Terrain Material	Tiled_TessellationLit3Combined
Tile Prefab	ValleyTile (Terrain Tile Decorated)
Save Path	Assets/ProceduralGrassGPUShader&TerrainTessellationTo
Smoothing Normal Intersection Factor	0.5
Terrain Decoration Mat Textures	0
Terrain Decoration Mat Textures Sliced	1
Terrain Decoration Masks	3
Element 0	Select
Element 1	Select
Element 2	Select
Terrain Decoration Extra Textures	1
Element 0	Select

(Example of use of modified script, result in Terrain Tile Decorated or in _TerrainTileDemo.unity)

3: Procedural Grass GPU Shader Content

Global Wind

The Procedural Grass Shader uses Wind to sway and move the Grass blades. This singleton manages it.



GlobalWind Static

Tag Untagged Layer Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

Global Wind (Script)

Script	GlobalWind
Wind Texture	noiseTexture
Wind Direction Speed	X 0.05 Y 0.05
Wind Texture Scale	70
Wind Intensity	0
Wind Contrast	0
Min Wind Strength	0.2
Max Wind Strength	1

Instead of only hardcoding the wind directly into the shader, I made a global wind script.

Its purpose is to scroll a Texture in one direction, and modify its values, just like in the shader.

The Shader is in the GPU, so if you need the same wind on the CPU, like moving a game object with the wind, get it from this script.

This script is also necessary to initialize the wind in the shader (takes the buffer from here at enable), so you will need it anyways. (Unless you want static wind)

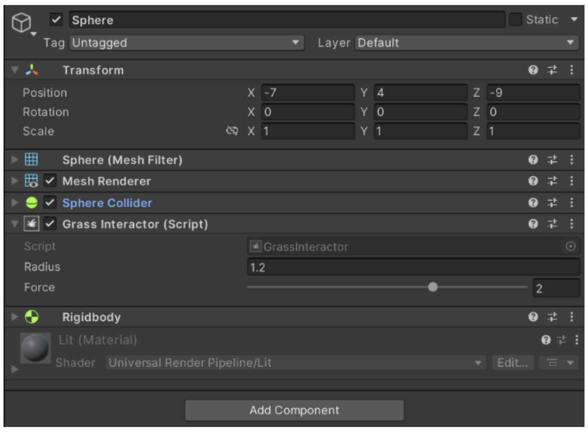
Please note that the script and shader are mostly independent, so changing the behavior in the script alone won't make it behave differently in the grass. Also, the texture is needed for the grass to work.

- Wind Texture: The texture to scroll, should be greyscale.
- Direction Speed: both direction and speed of the scrolling.
- Texture Scale: 1 scale is 1 unit until the texture repeats. The larger, more distance until the texture repeats.
- Wind Intensity: -1 to 1 value. 0 is neutral. Added to the texture and clamped. At -1, there is no wind. At 1, the wind is at max strength everywhere. At 0.2, the wind is 0.2 stronger than the default.
- Wind Contrast: 0 to 0.5 value. It makes that values less different to the min or max value get rounded to it. At 0.2 values from 0 to 0.2 are 0, and values from 1 to 0.8 are 1.
- Min/Max Wind Strength: Simple remap to those values. Min value is the new 0, and max value is the new 1. (Min should be lower than Max) Useful if you always want your grass to move a bit, or want to limit the max intensity so another windy area really shows the difference.

To get the wind in a position, Use public float GetWindAt (float posX, float posZ); of the singleton. It returns a float ranging from 0, to 1.

Thanks to that and public Vector2 GetWindDirectionSpeed (); You can get the wind direction, and strength, making it possible to make you own behaviors with it.

Grass Interactor


```
Sphere                                   Static
Tag Untagged              Layer Default
Transform
Position     X -7    Y 4    Z -9
Rotation     X 0     Y 0    Z 0
Scale        X 1     Y 1    Z 1
Sphere (Mesh Filter)
Mesh Renderer
Sphere Collider
Grass Interactor (Script)
Script       GrassInteractor
Radius       1.2
Force        2
Rigidbody
Lit (Material)
Shader  Universal Render Pipeline/Lit    Edit...
```

What is something without interaction in a videogame?

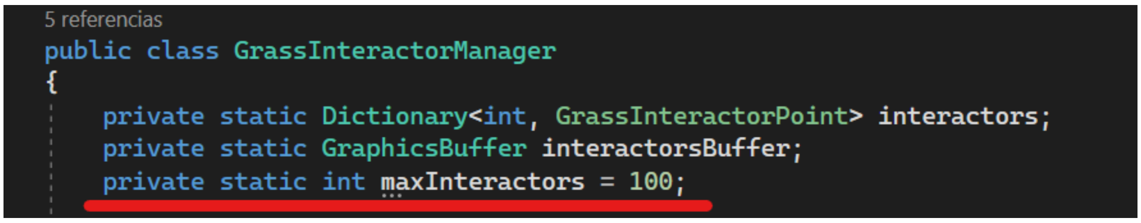
Even something in the GPU should at least be somewhat interactable.

What this Script does is very simple. It makes so that this Game Object, has a Radius where it can tilt grass downwards. If you want to keep it simple, add one to your character. If you want to make it fancier, add one in each foot.

- Radius: Units that this game Object has effect on.

- Force: Since the tilt difference is linear from the edge to the center, you might want to reach maximum tilt without reaching the center. So instead of a tilt of 0 to 1 in the complete center, you might want 0 to 2 or more, like that, at halfway of the radius its already at maximum tilt (1).

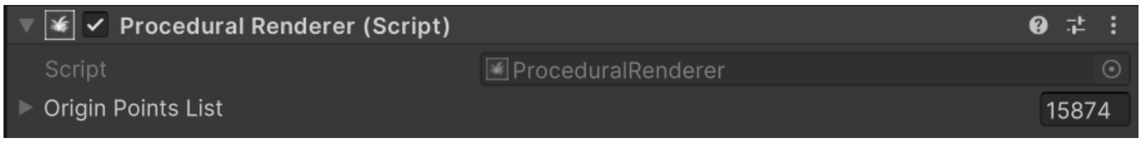
Please note, that there is a maximum of 100 Interactors at once, for efficiency's sake. If you want more, simply modify it in the script.


```
5 referencias
public class GrassInteractorManager
{
    private static Dictionary<int, GrassInteractorPoint> interactors;
    private static GraphicsBuffer interactorsBuffer;
    private static int maxInteractors = 100;
```

Procedural Renderer Base Class

The purpose of this class, is to serve as base to make your own procedural renderer faster, if you were to make you own. It encapsulates the general workflow of a procedural renderer with empty functions to override in the child, and some helper functions. Grass Renderer inherits from this class. Doesn't do much by itself.

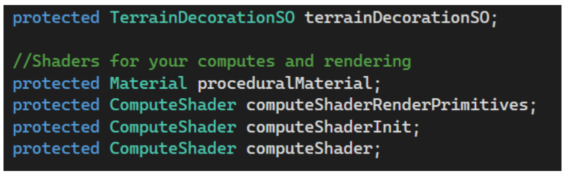
(You can ignore this script if you won't make your own renderers)



Procedural Renderer (Script)
Script ProceduralRenderer
Origin Points List 15874

It only has one [Serialized] property, Origin Points List: This list only holds Vector3 points, if you want to hold more data in each point (like tall grass ratio) you will need to make your own list. If your renderer only needs points to render upon, you can use it.

To Make your renderer, you will need to Create 5 different things:



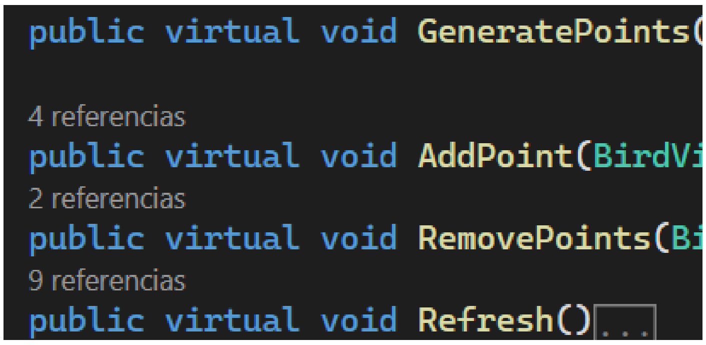
```
protected TerrainDecorationSO terrainDecorationSO;

//Shaders for your computes and rendering
protected Material proceduralMaterial;
protected ComputeShader computeShaderRenderPrimitives;
protected ComputeShader computeShaderInit;
protected ComputeShader computeShader;
```

- Terrain Decoration SO: Check Shader Properties/SOs content. It holds all the data of the decoration.
- Procedural Material: Material made with a Procedural Shader.
- ComputeShaderRenderPrimitives: Compute shader that calculates the arguments for Graphics.RenderPrimitivesIndexedIndirect() (example: triangle vertex array, max instance count)
- ComputeShaderInit: Compute shader that calculates everything that needs to be calculated only once at enable. (example: grass properties per blade, grass positions)
- ComputeShader: Compute shader that calculates everything that needs to be calculated each frame. (example: grass blades in frustum, wind force in each blade)

This script has some functions that you can override:

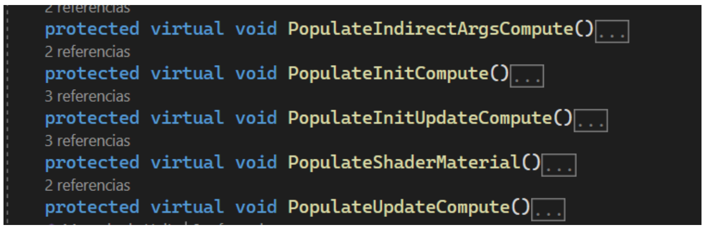
Brush Functions (optional, if you use a list that isn't origin Points):

- GeneratePoints(): Should Generate the Points List.
- AddPoint(): Should add a Point to the list in some coordinates
- RemovePoints(): Should remove all points inside the brush.
- Refresh(): Should Refresh the renderer with the updated point data.



```
public virtual void GeneratePoints(

4 referencias
public virtual void AddPoint(BirdVi
2 referencias
public virtual void RemovePoints(Bi
9 referencias
public virtual void Refresh()[...]
```

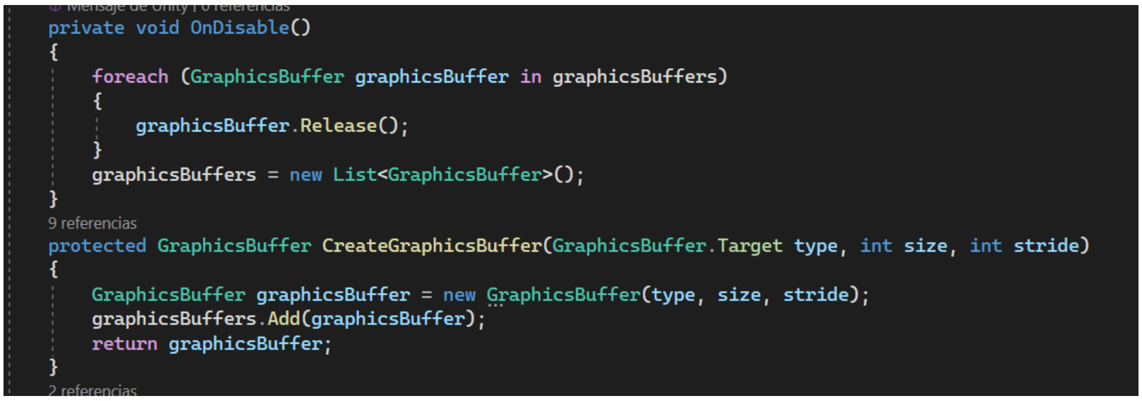
Compute Functions (Each one should populate computes/materials with Buffers/Properties)



```
2 referencias
protected virtual void PopulateIndirectArgsCompute()[...]
2 referencias
protected virtual void PopulateInitCompute()[...]
3 referencias
protected virtual void PopulateInitUpdateCompute()[...]
3 referencias
protected virtual void PopulateShaderMaterial()[...]
2 referencias
protected virtual void PopulateUpdateCompute()[...]
```

This class also has a helper function to create buffers. I found annoying having to dispose one by one all the buffers onDisable, each time I added another buffer having write it there too.

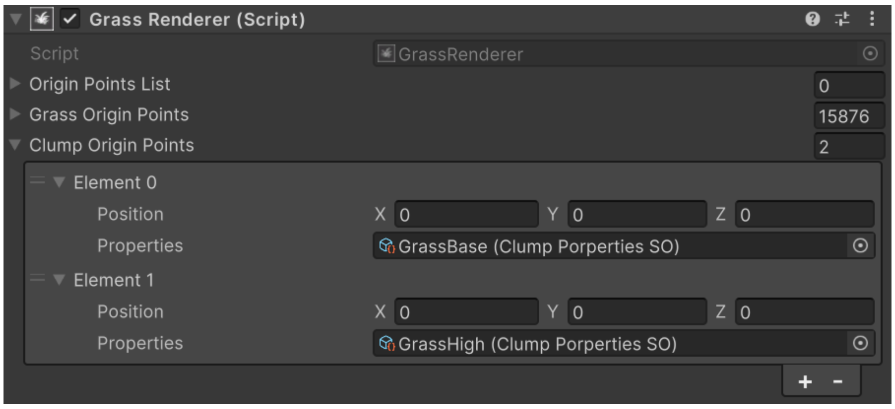
This function Creates a buffer, and adds it to a list to Release on disable. Simple but very comfortable.



```
Mensaje de Unity | 0 referencias
private void OnDisable()
{
    foreach (GraphicsBuffer graphicsBuffer in graphicsBuffers)
    {
        graphicsBuffer.Release();
    }
    graphicsBuffers = new List<GraphicsBuffer>();
}
9 referencias
protected GraphicsBuffer CreateGraphicsBuffer(GraphicsBuffer.Target type, int size, int stride)
{
    GraphicsBuffer graphicsBuffer = new GraphicsBuffer(type, size, stride);
    graphicsBuffers.Add(graphicsBuffer);
    return graphicsBuffer;
}
2 referencias
```

Grass Renderer

This script renders the grass on the screen. Nice!



Grass Renderer (Script)

Script	GrassRenderer
Origin Points List	0
Grass Origin Points	15876
Clump Origin Points	2
Element 0	
Position	X 0 Y 0 Z 0
Properties	GrassBase (Clump Porperties SO)
Element 1	
Position	X 0 Y 0 Z 0
Properties	GrassHigh (Clump Porperties SO)

It inherits from Procedural Renderer, but you shouldn't need to modify it. The only thing of note is that it needs to be set properly inside a tile Decorated, to get the decoration SO and compute references. Also, you should use one renderer per type of decoration per tile.

It has a different point structure than Procedural renderer, adding clump points, and clump ratio to it.

This way you can have tall grass, low grass, and everything in between!

(Refer To Clump properties SO to know how to modify them)

You can also create more clump points manually if you want to pull to center or make all grass blades point in an angle towards the clump point. Useful for vortex-like grass clumps.

4: Procedural Grass GPU Shader Properties/SOs

There is a lot of SOs and Properties that you can modify to create your custom grass. Each one has a lot of different fields, so you can truly get the look you want. Even the examples done don't cover all the looks you can manage by tweaking these. Experiment!

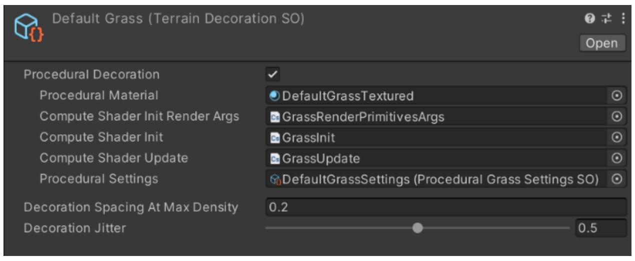
This is the execution order that results in your grass:

Terrain Tile Decorated -> Decoration SO* -> Grass Renderer -> Grass Clumps SOs.

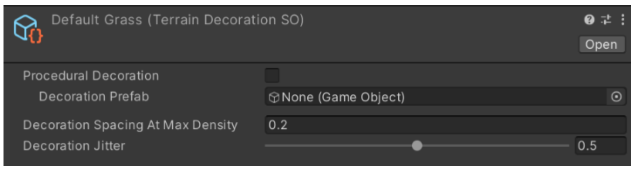
*Decoration SO -> Material + Computes, Grass Settings -> Render Settings.

Terrain Decoration SO

This SO serves as a simple way to reference all the files/SOs needed to render any decoration, and generate points fields.


Default Grass (Terrain Decoration SO)

Procedural Decoration	✓
Procedural Material	DefaultGrassTextured
Compute Shader Init Render Args	GrassRenderPrimitivesArgs
Compute Shader Init	GrassInit
Compute Shader Update	GrassUpdate
Procedural Settings	DefaultGrassSettings (Procedural Grass Settings SO)
Decoration Spacing At Max Density	0.2
Decoration Jitter	0.5


Default Grass (Terrain Decoration SO)

Procedural Decoration	
Decoration Prefab	None (Game Object)
Decoration Spacing At Max Density	0.2
Decoration Jitter	0.5

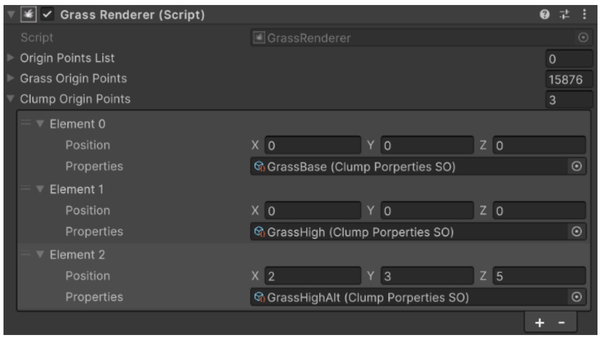
- Procedural Decoration: Pending TODO. Check if the decoration is procedural. Otherwise, you can instance a mesh in the GPU
- Procedural Material: Material made with a Procedural Shader.
- Compute Shader References: Compute shaders for your procedural renderer. Check Procedural Renderer Base Class if you want to make your own.
- Procedural Settings: You can put any SO here. Should have all the settings for your procedural decoration. For grass its grass settings SO.
- Decoration Spacing: How far the decorations points are from each other at max (1) density.
- Decoration Jitter: How much the decoration jitters from the original position. That way it isn't an obvious grid.
  0 means no change, 1 means that a point can move until the next. 0.5 means that a point can move halfway to the next.

Clump Properties SO

This SO defines properties of a clump of grass. The intent is to be able to render the same grass with different short/tall looks, without the need of another decoration defined.

The first SO assigned to the renderer is declared as the base properties, all the ones following are declared as secondary clumps. Every point in the renderer has its location, its secondary clump index, and ratio.

This means that in this renderer :



```
Grass Renderer (Script)
Script                GrassRenderer
Origin Points List                                  0
Grass Origin Points                             15876
Clump Origin Points                                 3
  Element 0
    Position          X 0        Y 0        Z 0
    Properties        GrassBase (Clump Porperties SO)
  Element 1
    Position          X 0        Y 0        Z 0
    Properties        GrassHigh (Clump Porperties SO)
  Element 2
    Position          X 2        Y 3        Z 5
    Properties        GrassHighAlt (Clump Porperties SO)
```

- GrassBase is the Primary Clump.
- GrassHigh and GrassHighAlt are secondary clumps.
- You can have combinations of:

(0) GrassBase + GrassBase

(1) GrassBase + GrassHigh

(2) GrassBase + GrassHighAlt

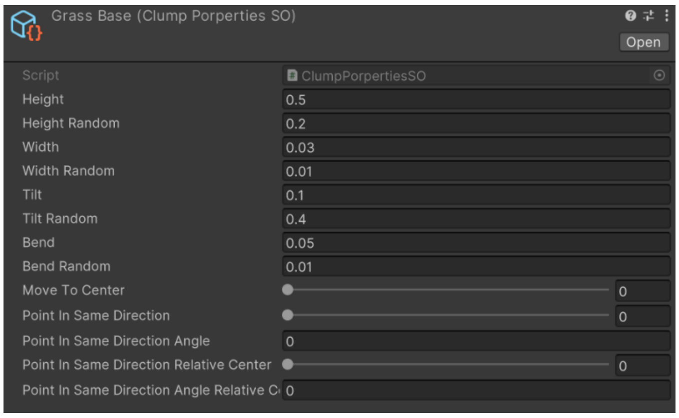
With index 0: Doesn't matter the ratio of the point, except in the color of the grass. (0 is base color, 1 is secondary color)

With index 1: Ratio 0 is GrassBase, Ratio 1 is GrassHigh, Ratio 0.5 is half GrassBase half GrassHigh.

With index 2: Ratio 0 is GrassBase, Ratio 1 is GrassHighAlt, Ratio 0.5 is half GrassBase half GrassHighAlt.

Its imposible to make half GrassHigh and half GrassHighAlt. as the primary clump is the grassBase. (You shouldn't really need that many combinations anyways. The base properties + extra ones at certain points should be more than enough)

Clump properties:


Grass Base (Clump Porperties SO)

Script	ClumpPorpertiesSO
Height	0.5
Height Random	0.2
Width	0.03
Width Random	0.01
Tilt	0.1
Tilt Random	0.4
Bend	0.05
Bend Random	0.01
Move To Center	0
Point In Same Direction	0
Point In Same Direction Angle	0
Point In Same Direction Relative Center	0
Point In Same Direction Angle Relative C	0

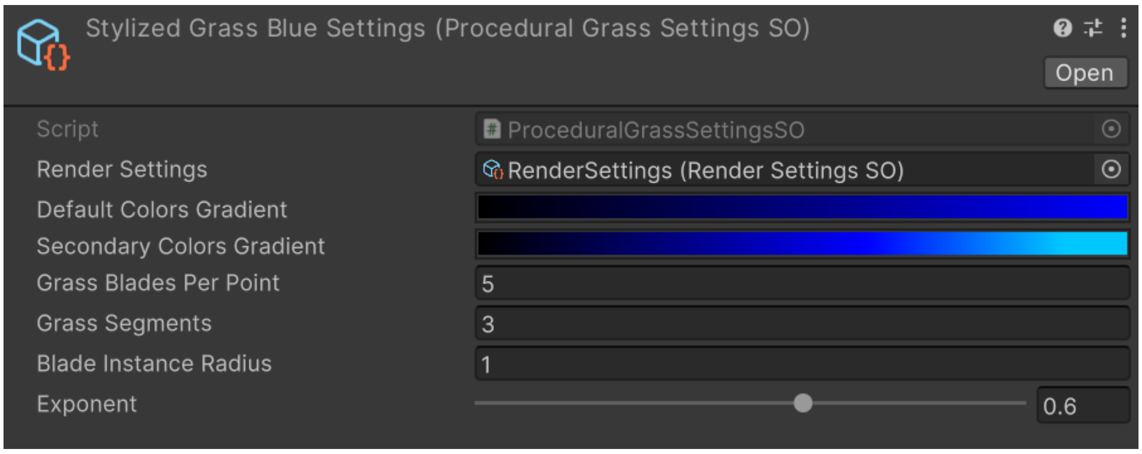
- Height: minimum height in units.
- Height random: maximum height added to minimum in random value.
- Width: minimum width in units.
- Width random: maximum width added to minimum in random value.
  Total Height/Width of a grass blade from this clump:
  0.5 + 0-0.2 Height ~0.6 units, 0.03 + 0-0.01 Width ~0.035 units
- Tilt: How minimally tilted is the blade. 0 is completely upwards. 1 or more is completely horizontal.
- Tilt random: maximum value added by randomness to the tilt value.
- Bend: How much is the blade curved backwards. 0 is straight line, the larger the number, the more curved it is.
- Bend Random: the same as the others, random value added to the minimum value.

Clump specific properties: These only work in secondary clumps, as the base shouldn't be in any position specifically. for best effect, ratio should be 1.

- Move to center: moves the grass blades to the Clump position.
- Point in the same direction: Rotates the grass blades towards the global angle.
- Point in the same direction Angle: the angle to rotate towards.
- Point in the same direction Relative Center: Rotates the grass blades towards the Clump Position.
- Point in the same direction Angle Center: the angle to rotate towards relative towards the Clump Position.

Procedural Grass Settings SO

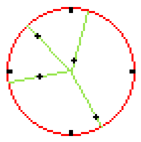
This SO contains the grass specific settings, that aren't dependent on Clumps.



Stylized Grass Blue Settings (Procedural Grass Settings SO)

Script	ProceduralGrassSettingsSO
Render Settings	RenderSettings (Render Settings SO)
Default Colors Gradient	
Secondary Colors Gradient	
Grass Blades Per Point	5
Grass Segments	3
Blade Instance Radius	1
Exponent	0.6

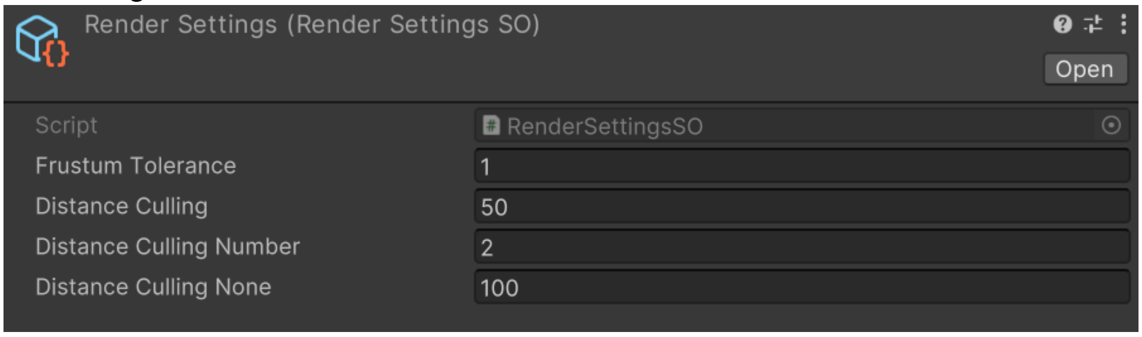
- Render Settings SO: This SO contains the render settings, refer to its page.
- Default colors gradient: The gradient where base colors will be picked.
- Secondary colors gradient: The gradient where secondary colors will be picked.
  The resulting color is a blend of the two gradients, depending on the ratio of the blade. 0 is base, 1 is secondary. 0.5 is half base, half primary.
  If the color is not accurate, keep in mind that you don't actually use the gradient, just colors from points based in segments. Either move the key points, or use more segments.
- Grass blades per point: the number of grass blades per one point in the renderer. 1 is one in the point, more results in an irregular circle around it. (number of red circle segments)
- Blade Instance Radius: the radius of units of the circle if more than one blade per point (green radius) (example of a generation of 4 blades per point) ------>



- Grass segments: number of square segments per blade. More means more definition, but more geometry. Last one is a triangle.
- Exponent: How is the segment distribution. 1 is linear, 0 is all in the end of the blade height.

Render Settings SO

This SO defines the render settings for a decoration. It changes the usage of resources of a decoration with LOD.



Script	RenderSettingsSO
Frustum Tolerance	1
Distance Culling	50
Distance Culling Number	2
Distance Culling None	100

- Frustum tolerance: the tolerance for occlusion culling. The greater the number, the less strict it is. 0 is no tolerance at all. It should be big enough that your decoration doesn't disappear in camera, but not too much, or you will render decorations that aren't in your camera.
- Distance culling: How far decorations start to behave differently to reduce resource usage.
- Distance culling number: For decorations that are abundant (like grass) defines after what number of skipped decorations, it should render one of them when the culling is in effect.
- Distance culling none: How far decorations start to simply don't render.

Example with 100 points:

In distance 0, out of the 100 points only those in the camera + tolerance frustum are rendered.

In distance 100 > X > 50, apart from the camera frustum, the remaining points, only half are rendered.

(1 is skipped, 2 is rendered. 3 is skipped, 4 is rendered. Etc.…)

(If the culling number was for example 5, Only 1 out of every 5 points would be rendered.)

In distance >100, nothing is rendered.

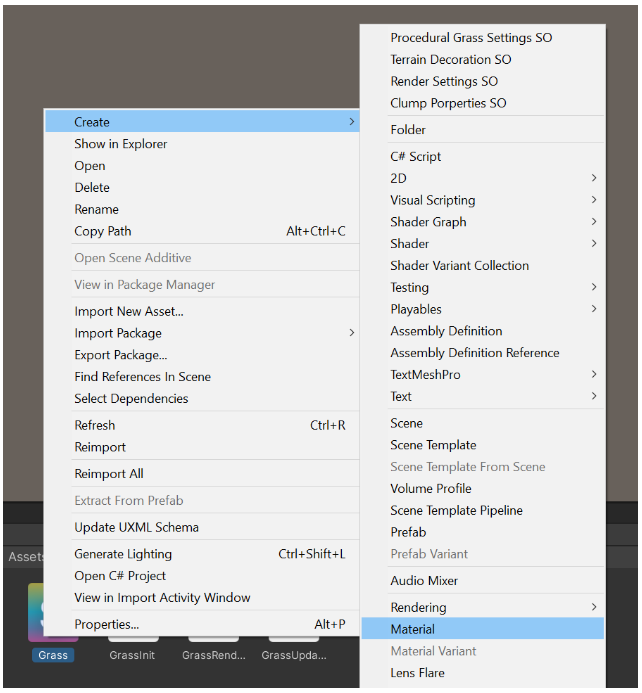
Shader Properties

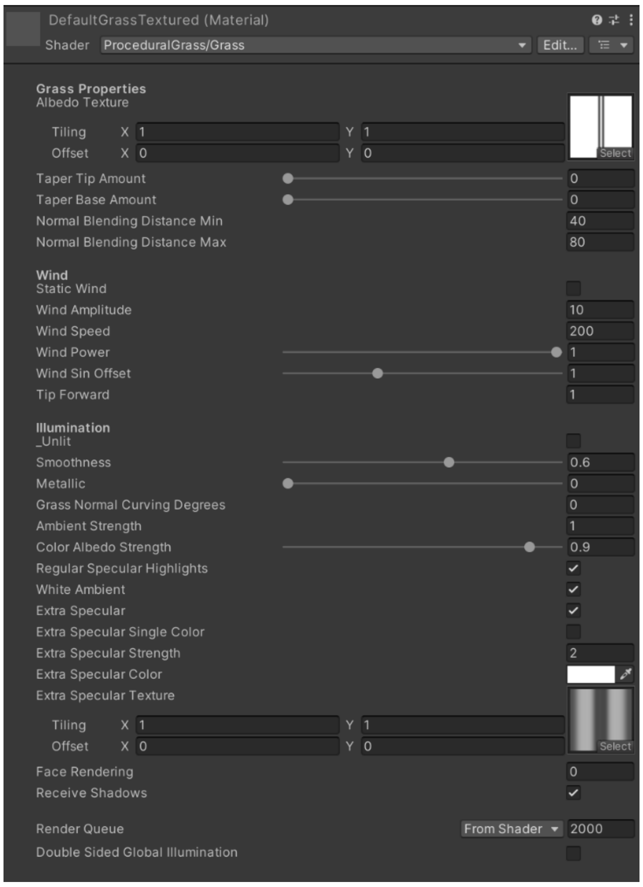
The grass also has a procedural shader. This shader can be used to create a material.

This material is used to change the look of the grass rendered. The difference between this and the computes/SOs is that this is applied to the mesh, to color it differently. It's the last step between the code and the screen.

Grass Properties:

- Albedo texture: the albedo on the grass.
- Tapper tip/base: How much the width is reduced in the tip or base. You can make it smaller in the tip, or base, or both. Giving you different shapes.
- Normal blending distance min/max: the minimum and maximum distances where normal are replaced by the ground. Less than minimum is no changes. Between both is a linear function. More than max is just the terrain normal instead. (this setting helps with flickering lights in faraway distances, reducing abrupt changes in normal values)




DefaultGrassTextured (Material)
Shader ProceduralGrass/Grass Edit...

Grass Properties
Albedo Texture
Tiling X 1 Y 1
Offset X 0 Y 0
Taper Tip Amount 0
Taper Base Amount 0
Normal Blending Distance Min 40
Normal Blending Distance Max 80

Wind
Static Wind
Wind Amplitude 10
Wind Speed 200
Wind Power 1
Wind Sin Offset 1
Tip Forward 1

Illumination
_Unlit
Smoothness 0.6
Metallic 0
Grass Normal Curving Degrees 0
Ambient Strength 1
Color Albedo Strength 0.9
Regular Specular Highlights ✓
White Ambient ✓
Extra Specular ✓
Extra Specular Single Color
Extra Specular Strength 2
Extra Specular Color
Extra Specular Texture
Tiling X 1 Y 1
Offset X 0 Y 0
Face Rendering 0
Receive Shadows ✓

Render Queue From Shader 2000
Double Sided Global Illumination

Wind Properties:

- Static wind: disables Global Wind dependency, by setting the wind strength to max to all blades. useful if you don't want to use dynamic wind, and would like the grass to sway just a little (should reduce other values too).
- Wind amplitude: how much distance the grass will sway up or down in the diagonal of the tilt, before looping. Larger values mean larger distances.
- Wind speed: how much speed does the grass sway at in maximum wind strength. Keep in mind that in lower wind strength the grass will be slower too. (0.3 wind strength is 0.3 of the wind speed)
- Wind power: This is a multiplier to wind strength, might be useful in some scenarios, but its preferable to either tweak other values in the material, or tweak the global wind values.
- Wind sin offset: the offset for the points of the blade. The larger it is, the more offset in the sin animation is between each point. At 0 the grass blade moves all at once, and at 1 there is a full cycle difference between the tip and base.
- Tip forward: simple value to move the tip forward. Might be useful for strong winds, but its preferable to modify tilt from SO or other values.

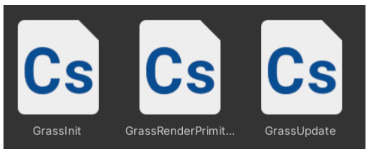
Illumination Properties:

- Unlit: Check this if you want the grass to be completely unlit, taking the color directly from the clump ratio and settings.
- Smoothness + metallic: regular lighting values, just like default material from unity.
- Grass normal curving degrees: how many degrees the grass blade will be split and changed. 0 means that the normal is just the blade aiming forward. 0.5 means that the normal is rotated for both points 0.5 angles inwards, in opposing directions, creating the effect of more geometry. (might make flickers because of lighting reflections)

- Ambient strength: since the grass is technically just a plane (with 0 normal curving degrees) if the light is near perpendicular the grass, or behind it, it simply renders black. That's why normally, you color stuff that isn't illuminated directly with the scene ambient lighting. So, by default the strength is 1. I noticed that for some effects, you might want to tweak this value. With lower ones, it has stronger contrast. While higher ones give the effect of being unlit. The grass blade always picks the brighter value. Its illuminated one, or the minimum ambient * ambient strength value.
- Color albedo strength: I noticed that the physically based illumination from unity is good for realistic effects, but not so much for stylized illumination. That's why I put a multiplier to the color and albedo. 1 is physically based illumination, while 0 is just black and white (except ambient color) this multiplier is good, since that makes lighting from other sources more pronounced. 0.9 for example makes the green grass slightly paler, but makes the color blue actually illuminate the grass blue. (physically based green albedo is only really illuminated by green lights)
- Regular specular: Check if you want unity specular highlights.
- White ambient: Check if you want the ambient value to be greyscale instead of RGB. Takes the strongest RGB value from the ambient.
- Extra specular: physically based illumination is limited by actual conservation of energy, and math… Meh. Add more color to the specular by checking this value!
  (Based on the grass blade color)
- Extra specular single color: Check this if you want the extra color to be any specific one. change it later in the color field.
- Extra specular strength: How strong you want the extra specular to be.
- Extra specular texture: In case you want to add a nice multiplier of specular to the blade. Makes it look nicer by not being homogenous. (should be greyscale)
- Face Rendering: 0 means both faces, 1 means front face, 2 means back face.

3 Grass Shader Files

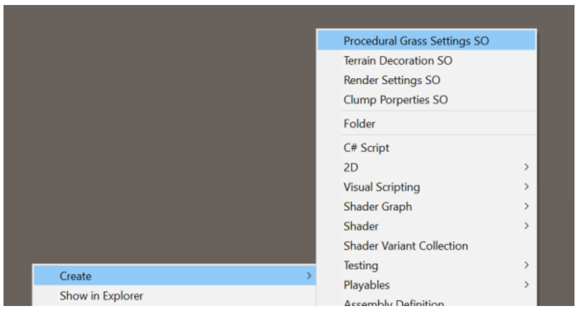
For the grass shader to function, you use 3 compute shaders. You don't really need to modify any of them. Just remember that they need to be referenced in the decoration SO properly.



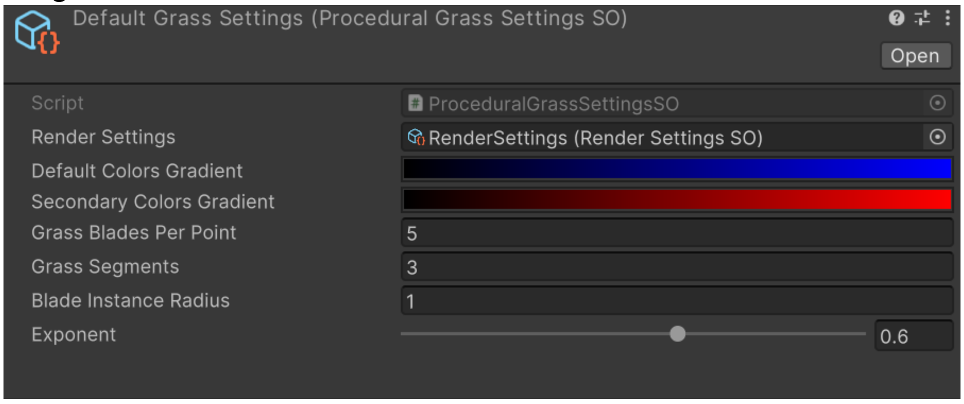
5: Getting started

Making your Custom Grass

For your custom grass, you first need to create your grass material, decoration SO, grass settings SO, render settings, and clump properties. (or copy/use another)



After that, just start filling the references and fields (if its tedious, I recommend just copy pasting the example ones, but remember that if you copy files with another reference, it will still point to the original one)

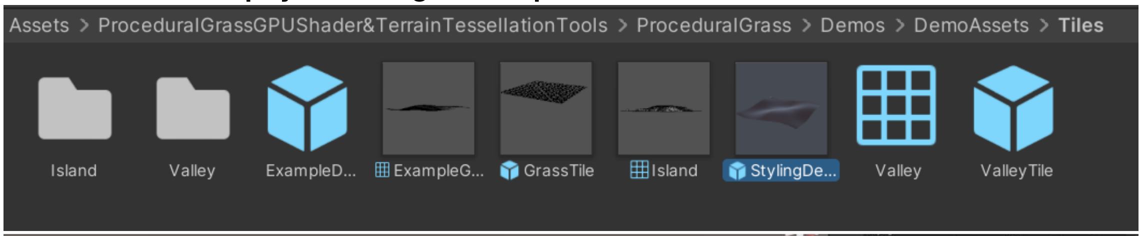

Default Grass Settings (Procedural Grass Settings SO)

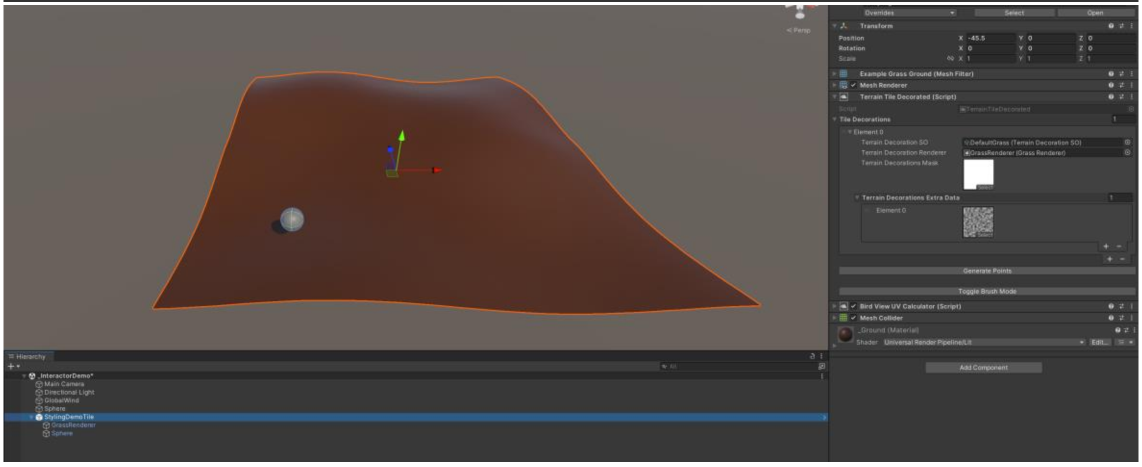
Open

Script	ProceduralGrassSettingsSO
Render Settings	RenderSettings (Render Settings SO)
Default Colors Gradient	
Secondary Colors Gradient	
Grass Blades Per Point	5
Grass Segments	3
Blade Instance Radius	1
Exponent	0.6

(Modified colors of the default grass)

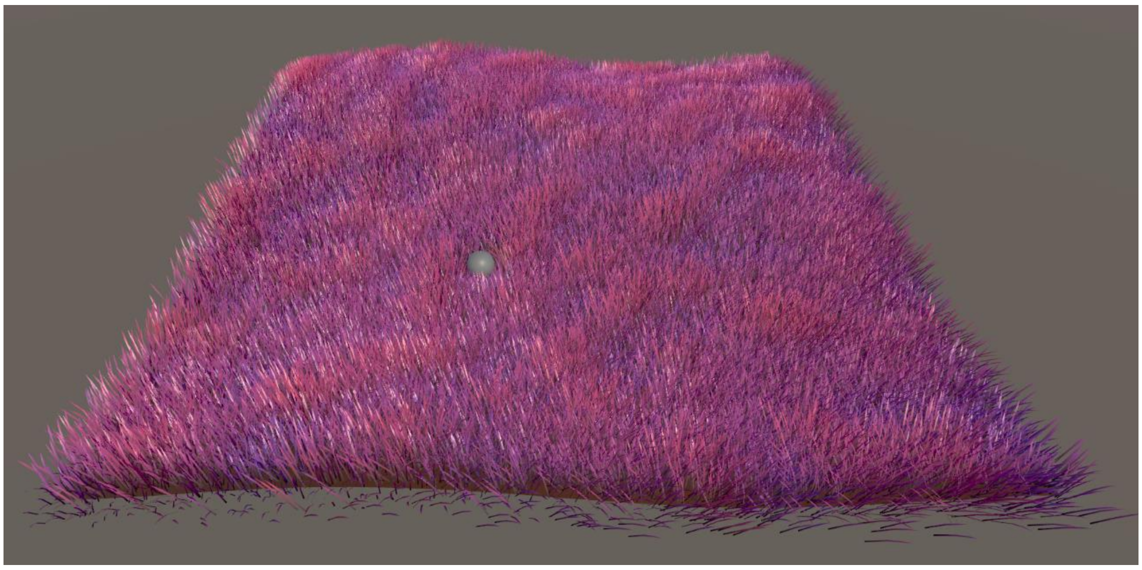
As the last step, just drag a tile prefab into the scene.


Assets > ProceduralGrassGPUShader&TerrainTessellationTools > ProceduralGrass > Demos > DemoAssets > Tiles
Island
Valley
ExampleD...
ExampleG...
GrassTile
Island
StylingDe...
Valley
ValleyTile



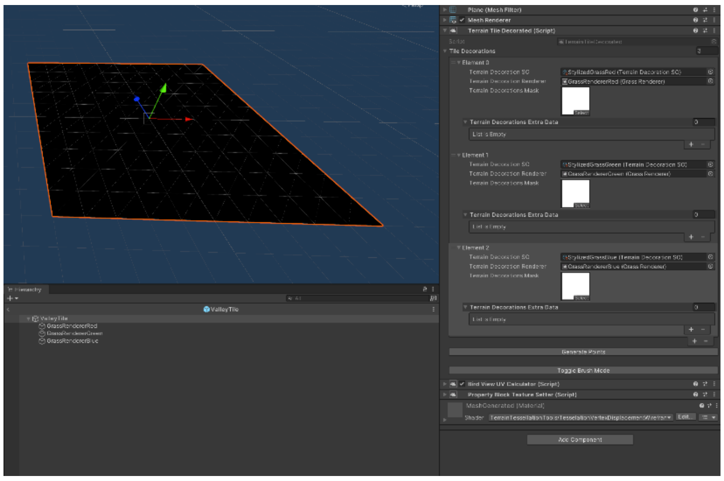
Transform
Position X -45.5 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Example Grass Ground (Mesh Filter)
Mesh Renderer
Terrain Tile Decorated (Script)
Script TerrainTileDecorated
Tile Decorations 1
Element 0
Terrain Decoration SO DefaultGrass (Terrain Decoration SO)
Terrain Decoration Renderer GrassRenderer (Grass Renderer)
Terrain Decorations Mask
Terrain Decorations Extra Data 1
Element 0
Generate Points
Toggle Brush Mode
Bird View UV Calculator (Script)
Mesh Collider
_Ground (Material)
Shader Universal Render Pipeline/Lit
Add Component

And click generate points to see how your grass looks! (make sure the camera points to the grass, or it will be culled)



Setting up your tile prefab

Every time you are going to make a different type of terrain, you will probably use a different set of decorations. When you make a Tile prefab you need to make sure everything is set correctly:



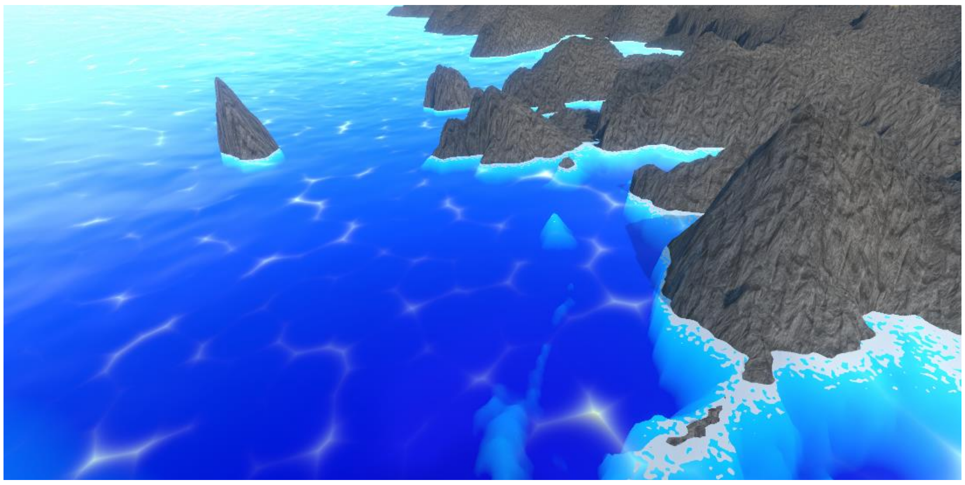
```
Plane (Mesh Filter)
Mesh Renderer
Terrain Tile Decorated (Script)
  Script                          TerrainTileDecorated
  Tile Decorations                3
    Element 0
      Terrain Decoration SO         StylizedGrassRed (Terrain Decoration SO)
      Terrain Decoration Renderer   GrassRendererRed (Grass Renderer)
      Terrain Decorations Mask
      Terrain Decorations Extra Data  0
        List is Empty
    Element 1
      Terrain Decoration SO         StylizedGrassGreen (Terrain Decoration SO)
      Terrain Decoration Renderer   GrassRendererGreen (Grass Renderer)
      Terrain Decorations Mask
      Terrain Decorations Extra Data  0
        List is Empty
    Element 2
      Terrain Decoration SO         StylizedGrassBlue (Terrain Decoration SO)
      Terrain Decoration Renderer   GrassRendererBlue (Grass Renderer)
      Terrain Decorations Mask
      Terrain Decorations Extra Data  0
        List is Empty
  Generate Points
  Toggle Brush Mode
Bird View UV Calculator (Script)
Property Block Texture Setter (Script)
MeshGenerated (Material)
  Shader  TerrainTessellationTools/TesselationVertexDisplacementWirefram
```

```
Hierarchy
ValleyTile
  GrassRendererRed
  GrassRendererGreen
  GrassRendererBlue
```

- Every decoration must have the SO, and the reference to the renderer in the prefab.
- The number of decorations should be equal to the number of renderers.
- The renderers should be immediate children of the tile.
- The renderers should be empty, otherwise all the points will be created just to be replaced immediately at point generation slowing the process.
- The renderers should be set up properly. (in grass, they should have the clump points ready)

6: Extra Observations & Content

Freebie: Somewhat nice water shader



This is a simple water shader I made with Shader Graph a while before.

I decided to use it for the island demo.

You can use it if you want :)