Advanced Terrain Grass

ATG is a grass rendering solution which was built to replace the detail rendering system of Unity's terrain engine in order to get rid of almost all spikes on the CPU — which usually are caused by new grass patches coming into sight when your camera is moving. So ATG will help you to reduce work on the CPU – but at the same time it might add some more costs on the GPU side.

Next to this initial purpose ATG offers significant improvements regarding lighting and wind animation.

ATG is *not* an authoring tool: It simply takes the grass and details you have added to your terrains and takes over their rendering. That being said ATG is compatible with any 3rd party authoring solution that allows you to create grass density maps for built in terrains within the Unity editor like e.g. Terrain Composer, Map Magic (not runtime), World Creator or Gaia. Runtime distributed grass currently is not supported.

Benefits

ATG was built to avoid major spikes on the CPU and gives you quite stable and reliable costs per frame, or — in order to be more specific — time slices and shifts work from the main thread to a worker thread and the GPU.

It lets you use advanced lighting features such as physically inspired specular highlights and translucent lighting and enables any grass prototype to even cast real time shadows.

ATG lets you reset the world's origin and move terrains around to handle floating point issues.

Limitations

ATG is mostly suitable in case you are CPU bound.

ATG needs DX11, Metal (tested on OS X – not iOS) or OpenGLCore (Linux or Windows, Mac is not supported). Xbox One and PS4 should be fine as well although i have not been able to test it.

ATG currently only works on squared terrains and does not support *Detail Resolutions* < 256. Billboarded grass is not supported either.

ATG currently does not support lightmaps. And it does not support real time lightmaps as well, which — unfortunately — is by design, as Unity does not let us access real time lightmap data from script.

In order to get a rather smooth start I recommend to have a look at the included demos and start with the section Demos first.

In case you just want to get right to the meat, start with the Components.

All requirements as far as textures and meshes regarding your grass models and textures are concerned are covered in the section Model and texture Guide.

If you want to get the best out of ATG please have a look into the section Optimization.

In case you just want to use the ATG shaders with third party rendering solutions such as Vegetation Studio head over here.

Using ATG with URP is covered here.

Compatibility

Main Camera

Currently AFG needs your camera to be tagged as “MainCamera”. Grass may be rendered by all other cameras as well like mirror or reflection cameras, but as AFG uses Culling Groups it needs a reference point which at the moment is fixed coded and set to “MainCamera”.

Forward Rendering

In case your camera uses **forward rendering** and you have any depth based image effects applied (like ssao) you have to assign the **ATG_Internal-DepthNormalsTexture** shader in *Edit -> Project Settings -> Graphics*:

- Switch to *Debug* (using the dropdown icon in the upper right corner of the inspector) in order to bring up all inputs.
- Change *Depth Normals* to *Custom Shader*, then assign the *ATG_Internal-DepthNormalsTexture* shader.

Deferred Rendering

In order to implement deferred translucent lighting and physically inspired specular reflections ATG ships with its own deferred light and deferred reflection shaders.

So in case your camera uses **deferred rendering** you have to assign the **ATG deferred shaders** in *Edit -> Project Settings -> Graphics*:

- Under the *Built-in shader settings* change *Deferred* to *custom*, then assign the *ATG_Internal-DeferredShading* shader.
- Also change *Deferred Reflections* to *Custom* and assign the *ATG_Internal-DeferredReflections* shader.

ATG and AFS, Lux, Alloy or UBER

As AFS, Lux Plus, Alloy and UBER all bring their own deferred lighting and deferred reflection shaders you may have to enable support for their specific deferred translucent lighting solution.

Please note: Depending on which deferred lighting solution you use, you will get slightly different results. But the basic translucent lighting just should work.

Do so by editing the “AtgPBSLighting.cginc” and simply uncomment either:
`// #define USEALLOY`
or:
`// #define USEUBER`
Then reimport the ATG folder.

In case you use UBER please note that ATG translucency always uses “Translucency Setup 1” defined in the “UBER_Deferred Params” script. Recommended Settings are:

Strength Value between 8 and 10 should be fine.

Scattering I would keep it rather low e.g. 0.1.

Spot Exponent A value around 10 should match ATG’s lighting.

NdotL Reduction Keep it rather low <= 0.5 to get nicely backlit grass.

Please note: Physically inspired specular reflections on grass are only supported by the original ATG shaders and Lux Plus.

Table of Content

Grass vanishes after enabling RenderDoc
Using VR half the grass is missing

Performance

CPU Performance

Built in Unity grass might render pretty fast, at least as far as GPU performance is concerned. However, whenever the camera moves and new patches of grass come into sight, these patches have to be generated on the CPU, which simply causes spikes. Depending on the density of your grass and the complexity of the grass meshes involved this task might spike between 2 – 20ms or even more on the CPU.

ATG does not generate any meshes. It calculates matrices and sends these to the GPU in order to draw grass using DrawmeshInstancedIndirect. As the calculation of the matrices is done on a worker thread ATG is rather cheap on the main thread and usually costs between 0.2 and 1.0ms.

GPU Performance

Drawing a huge amount of grass always is quite expensive as it just causes a lot of overdraw. No matter how you actually render the grass, the GPU will most likely be bottlenecked by fill rate and the number of pixels the grass covers on screen.

ATG does not entirely solve this issue but addresses it by offering various techniques such as Two Step Culling [>]: Grass of course gets culled at the given culling or detail distance. But furthermore you may define a “Detail Fade Start”. Starting from this distance the ATG grass shaders start to cull a user defined amount of instances even before the overall culling distance. This lets you have very lush and dense grass close to the camera while the grass in the background will be less dense and thus cost less fill rate.

Nevertheless ATG shifts some work from the CPU to the GPU, so in case you are GPU bound it will most likely lower FPS – unless you can make use of Compute [>].

Draw Calls

Unlike Unity’s built in grass ATG can not combine different grass prototypes into a single patch and draw the whole patch with just one draw call. ATG has to draw each prototype for each patch using a single draw call. This will most likely crank up the number of draw calls which stresses both CPU and GPU but can be tackled using texture arrays and merging layers.
Merging layers [>]

In case you address modern PCs, Xbox One or PS4 enabling Compute [>] will reduce the number of draw calls as well. Although Metal supports compute using compute seems to slow things down.

Getting started

In order to get familiar with ATG I recommend creating a new project, importing the package and jumping into the demos. They will show all features of ATG and should give you a proper understanding of how it all works.

Please note:

ATG offers advanced lighting features such as translucency and physically inspired specular reflections on grass. So in case your camera uses **deferred rendering** you have to assign the **ATG deferred shaders** in *Edit -> Project Settings -> Graphics*:

- Under the *Built-in shader settings* change *Deferred* to *custom*, then assign the *ATG_Internal-DeferredShading* shader.
- Also change *Deferred Reflections* to *Custom* and assign the *ATG_Internal-DeferredReflections* shader.

In case your camera uses **forward rendering** and you have any depth based image effects applied (like ssao) you have to assign the **ATG_Camera-DepthNormalTexture** shader in *Edit -> Project Settings -> Graphics*:

- Switch to *Debug* (using the dropdown icon in the upper right corner of the inspector) in order to bring up all inputs.
- Change *Depth Normals* to *Custom Shader*, then assign the *ATG_Internal-DepthNormalsTexture* shader.

As soon as your project is setup properly we can get started with the Quickstart Demo:

Quickstart Demo

ATG is pretty easy to initially set up. In order to let you try it yourself the package contains a simple scene called “Quickstart” which only contains a terrain, a directional light and a camera.

Open the scene and find the terrain. You will notice that there already is some grass painted on top of the terrain. So all you have to do is simply add the “Grass Manager” script to the terrain. Hit “play”and you are done :).

Grass rendering in play mode is now done by ATG.

Please note: ATG by default only renders in play mode. In edit mode grass usually is handled by the terrain engine. Have a look at Draw in Edit Mode [>] to find out how to make ATG render the grass in edit mode as well.

Under the hood

As soon as you attach the script, it scans all the detail prototypes of the terrain, assigns default meshes in case a prototype is just a texture and creates the needed ATG materials. These materials are stored right in the folder where the main texture of the given prototype is located. Healthy and Dry Color are copied to the new materials and Min and Max Size are set.
As ATG can only perform uniform scaling, size is derived from Min and Max Height only.

Cull distance and density are set according to the settings on the terrain.

When you start the scene, the script will read in the height and detail maps of the terrain which are needed to render grass at runtime. This may take up to several seconds.

Then in play mode the entire rendering of all grass prototypes will be handled by ATG.

There are several things you might have noticed on entering play mode:

1. In case ATG is active it takes a while until all the terrain is filled with grass. Instead of getting an instant result you see the individual cells become visible one after the other.
2. Grass does not sway in the wind.
3. Grass is aligned to the terrain's surface which might look odd on tall prototypes like the "plant".
4. In case you have opened the stats window you may have noticed a drop of fps when ATG is enabled.

So let's address these issues one after the other:

Using Burst Init (1.)

By default ATG inits one cell per frame in case *Init complete Cell* is activated. This is the key to avoid heavy spikes on the CPU.

Nevertheless blocking in grass cells at start up might be very annoying. By activating *Use Burst Init* in the section *Render Settings* of the *Grass Manager* script you will force the script to at least init all cells within a radius around the camera specified in *Radius*.

Adding Wind (2.)

In order to add wind to the grass, you have to add the *ATG Wind* script to the scene as all ATG shaders use custom wind driven by a global render texture.

You may either create a standard Wind Zone from scratch and then attach the script – or simply drag the *ATG Wind Prefab* into the hierarchy.

Now when you hit "play" grass should nicely bend in the wind.
Find out more about Wind [>].

Changing the alignment of grass instances (3.)

Unlike the terrain engine ATG by default rotates all grass instances so they finally get rendered perpendicularly to the terrain's surface. This looks nice on rather low grass prototypes, but tall prototypes may look a bit odd.

In order to change this you have to edit the ***Rotation Mode***: Go to the *Grass Manger* script → *Section Prototypes* and find *Rotation Mode*. Now change this to *Not Aligned Random Y Axis*. All prototypes using this mode will be vertically aligned just like you know from the terrain engine. In order to make these prototypes still pick up lighting from the terrain slope you have to enable ***Write Normal Buffe****r* and additionally *Use Normal Buffer* in the material.

But you do not have to do so: Not writing and using the normal buffer will light the prototypes just as if they were vertically aligned.

Increasing FPS (4.)

In fact the Quickstart demo has pretty dense grass, so fps are most likely GPU bound which is caused by the heavy amount of overdraw.

Depending on your GPU using ATG might lower the final fps by 10–20% compared to simply using the terrain engine — which is quite a downer.

This might be caused by the slightly different culling or the complexity of the ATG shaders, the number of draw calls ATG produces, but in fact there often is simply no faster way to draw something on screen but just sending a bunch of merged meshes to the GPU.

Latter is what Unity's built in grass does: It merges all prototypes into one mesh per patch (which causes the spikes) and sends these to the GPU. ATG in contrast draws individual grass instances. And it draws each prototype or layer separately.

Please note: In other scenarios ATG might outperform the terrain engine even on the GPU like in the *Full Demo*. So the actual GPU performance heavily depends on the number of draw calls vs. overdraw.

Nevertheless ATG should reduce the amount of overdraw as well as the number of draw calls. It does so by introducing **two step culling** and **merging layers**:

Two step culling

The Grass Manager offers *Detail Fade Start* and the corresponding *Fade Length* which will lead to some of the grass instances fade out even before the overall *Cull Distance*. So you may either decrease the *Detail Fade* values or increase the number of instances which will be affected by this feature.

In order to increase the number of instances which will be affected by *Detail Fading* you have to edit the ATG materials and find the *Clip Threshold* property. Its default value is set to 0.3 which means that 0.3 of all instances will use the detail fade feature. If you raise it to 0.5 half the instances will fade out early thus you will get less overdraw.

Using this feature aggressively let's us come close to the FPS we get from using the terrain engine but of course lowers the density towards the cull distance.

Please note: Min Size must not equal Max Size in case you use two step culling and set *Clip Threshold* (in the material) to > 0.0 as otherwise the grass shaders will fade out all instances early on.

Merging layers

Another possibility to increase performance on both CPU and GPU is to **merge layers**. Although this is not really suitable in case of the Quickstart demo as both grass prototypes are rather different as far as their size is concerned I would like to mention this feature here as it may boost performance.

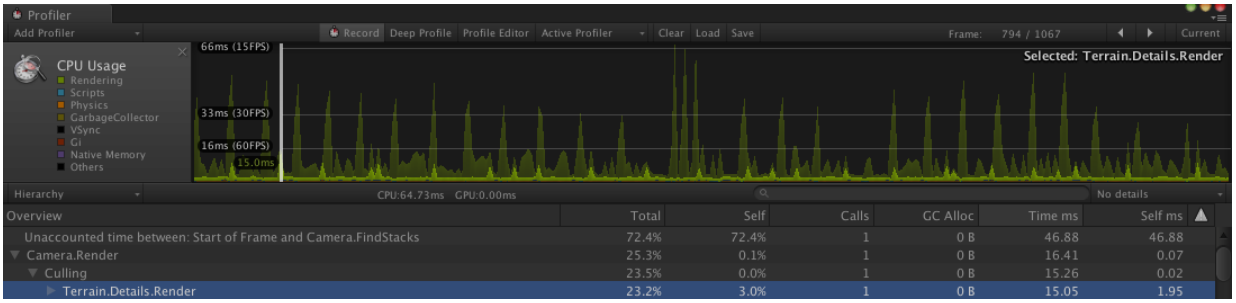
Merging layers [>]

Checking CPU performance

Let's get back to the demo and check CPU performance as this is what ATG first and foremost is about:

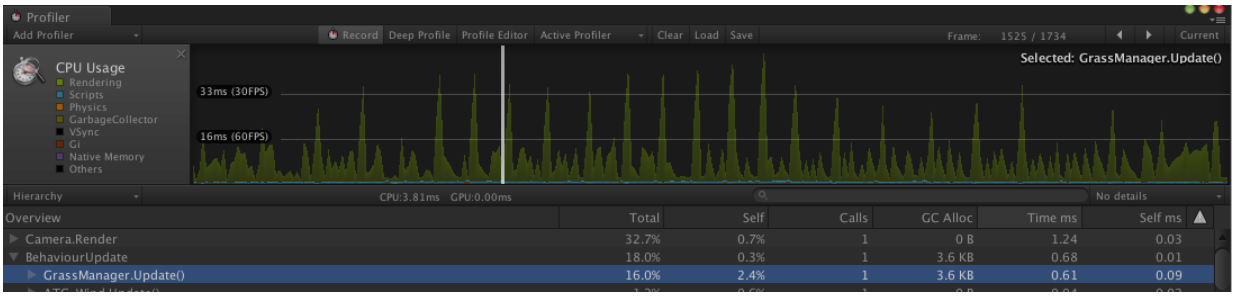
Select the camera and activate the assigned *Extended Flycam* script. So now, if you disable the *Grass Manger* script so that the terrain engine will render the grass, open the profiler, enter playmode and use WASD to move around, you will force the terrain engine to generate new patches of grass, which usually creates quite noticeable spikes on the CPU.

Go to the profiler and have to look for “Camera.Render” -> “Culling” -> “Terrain.Details.Render”, which should show the spikes from time to time on the CPU:



Terrain Engine – The bright green curve shows the costs for rendering the grass which spikes from time to time – up to 15ms. The dark green curve shows how long the CPU has to wait for the GPU. We are GPU bound here, as no other script is running on the CPU: no game logic, no AI, no nothing.

Next activate the grass manager, hit play and move around again. Then have a look at the CPU usage — especially the costs of the “BehaviourUpdate” → “Grass Manager”: That is where most costs of ATG will be listed:



Grass Manager – The costs of the Grass Manager are shown by the cyan colored curve which is barely visible. The dark green curve has slightly changed but still is pretty close to the result from the terrain engine.

Takeaways

- ATG runs at pretty stable 0.3 — 0.5ms on the CPU and leaves a lot of room for other scripts — while the built in grass will cause significant spikes from time to time.
- In this case ATG produces slightly more pressure on the GPU which can be tackled by using two step culling and merging layers.
- Use the profiler. Always, ever :)

Full Demo

This example is more complex and already setup. Its terrain has eight grass layers which use different settings and ATG shaders.

Please note: This demo uses advanced lighting features such as translucency and physically inspired specular reflections on grass. So in case your camera uses **deferred rendering** you have to assign the **ATG deferred shaders** in *Edit -> Project Settings -> Graphics*:

Under the *Built-in shader settings* change *Deferred* to *custom*, then assign the *ATG_Internal-DeferredShading* shader.

Also change *Deferred Reflections* to *Custom* and assign the *ATG_Internal-DeferredReflections* shader.

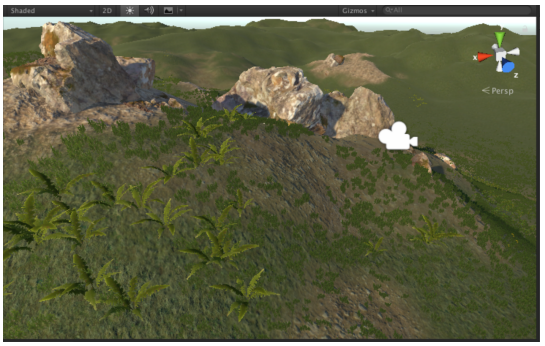
Prototypes

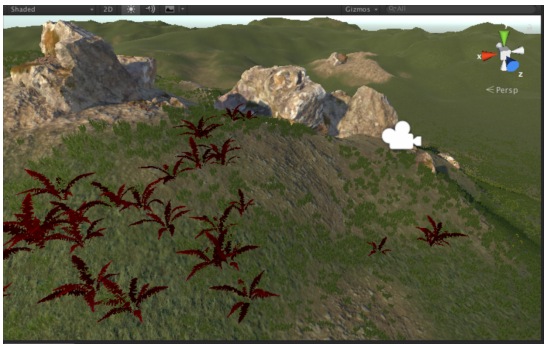
Generic grass has been added to the terrain as Detail Mesh using the Grass shader.
Rotation is set to *Aligned Random Y Axis* which makes it follow the terrain slope and leads to lighting which matches that of the terrain.
It uses the *Grass Array Shader* to mix different grass textures because detail layer 7 and 8 are set to be merged into this layer. Merging layers [>]
Specular Highlights are enabled and the corresponding texture array has been assigned so specular lighting as well as translucency are driven by this additional texture input.
Wind is sampled per vertex which will lead to some distortion of the mesh – but actually looks fine here.

Dry Grass has been added to the terrain as a simple texture so the script has automatically assigned one of the default meshes.
Rotation is set to *Not Aligned Random Y Axis* so it will always be positioned vertically.
In order to pick up the lighting from the terrain, Write *Normal Buffer* is checked.
As it is rather huge it is set to *cast shadows*.
Dry Grass uses the *ATG Grass Base Shader* which is set to use the *NormalBuffer* to pick up the lighting from the terrain.
Specular Highlights are disabled so the shader does not read a second texture. Translucency is solely calculated based on the albedo texture and not masked.

Small Stone has been added to the terrain as Detail Mesh using the VertexLit shader.
Rotation is set to *Aligned Random XY Axis* which makes it follow the terrain slope and adds some random rotation around the X and Y axis.
It uses the *ATG VertexLit* shader which allows us to add smoothness to the alpha channel of the albedo texture and assign a normal texture so the shader can do fully physically based lighting. Specular color however is simply derived from a single color input.

Fishbone Fern has been added to the terrain as Detail Mesh using the built in VertexLit shader. Latter because its mesh is setup to support the foliage shader and therefore contains various vertex colors which — in case we assigned the built in grass shader — would give a pretty strangely colored result in the scene view as shown below:





VertexLit Shader in scene view

The hacked VertexLit shader AFG ships with does not multiply the vertex colors on top of the albedo. Therefore we won't see any Healthy or Dry Colors but it will not strangely colorize meshes which are Prepared to be used with the foliage shader.

Grass Shader in scene view

The grass shader will apply vertex colors to albedo. So it lets you preview Healthy and Dry Colors but it will also show vertex colors baked to the mesh what in case of the fishbone fern mesh would and in a result like this.

Rotation is set to *Aligned Random Y Axis* which makes it follow the terrain slope and helps to avoid ugly intersections with the terrain.
Its material is set to not do any backface culling as the mesh is single sided and has assigned a secondary texture which is compatible with the textures used by the advanced foliage shaders and stores the normal as well as translucency and smoothness.
Fern casts *real time shadows*.

Nettle has been added to the terrain as Detail Mesh using the VertexLit shader as — just like the fern — its mesh is set up for the ATG foliage shader.
Rotation is set to *Not Aligned Random Y Axis* so it will always be positioned vertically.
The ATG foliage shader does not support picking up the terrain normals. So *Write Normal Buffer* is unchecked.
Nettle casts *real time shadows*.

Pineapple Weed has been added to the terrain as Detail Mesh using the Grass shader.
You could compare it to the nettle regarding its shape and complexity. However as the pineapple weed does not have any horizontally aligned leaves we do not "need" proper specular reflections here and just use the ATG Grass base shader.
Rotation is set to *Not Aligned Random Y Axis* so it will always be positioned vertically.
Wind is sampled only at the pivot in order to avoid any distortions of the mesh which would look odd on the blossoms. Furthermore *Wind LOD* is set to 1 so the shader will sample a slightly smoother version of the wind texture.

02_Grass_Daisies and **03_Grass_brownTips** both have been added as simple textures. However as both are set to be soft merged with layer 1 (Generic Grass) they will finally be rendered using the mesh and material assigned to layer 1. Merging layers [>]

Running the Demo

When entering play mode you may move around using WASD / QE + Shift and the mouse. You will notice that a lot of prototypes cast real time shadows – which makes it quite expensive and might lead to flickering shadows on single cells (Unity < 5.6.3.).

Changing the grass rendering settings at runtime

You may hit "1" to restore the default grass density settings or "2" in order to set it to 2.0. The according functionality can be found in the "SwitchGrassSettings.cs" which calls the RefreshGrassRenderingSettings() function of the Grass Manager script.

Switching between cameras at runtime

In case you do cutscenes and have to switch between different cameras the Grass Manager will most likely fail to render the grass in the new camera. In order to solve this you have to make sure that the "Cam" variable in the Grass Manager scripts refers to the currently active camera.

You may either simply reset the “Cam” variable in the Grass Manager and make it look for a camera tagged as “MainCamera” – or you can manually set Cam = new camera.

If the two cameras you switch between are positioned at very different locations, the grass cells which are visible to the camera which gets activated most likely will not be initialized. So to avoid grass being rendered delayed in the new camera you may manually call the Grass Manager’s BurstInit() function. This will give you a **significant spike** on the CPU but ensures that all grass cells within the given Radius will be initialized on switch.

Please have a look at the “SwitchCameras” script to find out more.

Teleporting

In case you teleport your player/camera to a very different location the grass cells which are visible at the new position most likely will not be initialized. To avoid grass being rendered delayed at the new position you may call the Grass Manager’s BurstInit() function.

Please have a look at the “SwitchCameras” script to find out more.

Compute

By default the grass manager does not use the Compute [>] feature. So you may want to check it out by enabling it in the manager and see how your frame rate changes. You may have to wait a few seconds tho to get the right picture as usually FPS increases after 3–5 seconds.
Please note: Do not enable compute while the scene is running. You have to quit play mode and restart the scene.

Takeaways

- The various shaders ATG ships with lets you add different types of details to the terrain like grass, flowers, smaller foliage such as fern and small rocks in order to create lively and beautifully lit environments just using the built in terrain engine.
- Using different rotation modes and slightly different wind settings allow you to easily find the optimal settings for each prototype.
- Merging Layers helps to reduce the number of draw calls and most likely will speed up rendering.
- The RefreshGrassRenderingSettings() function of the Grass Manager script let’s you change various rendering settings at runtime.
- You may switch between different cameras or teleport your player.
- Shadows need Unity >= 5.6.3
- Using Compute [>] may significantly improve the frame rate.

Wind Demo

This scene lets you dive into the ATG wind settings. It contains the **ATG Wind Prefab** as well as the **ATG Wind Viz Prefab**. Latter is just a simple plane using the **Wind Visualisation material** and **WindViz** shader, so it will show the generated global wind texture.
By default it shows the *main wind strength*. In order to view the *noise* which will drive the gusting you have to edit the *Wind Visualisation* material and change *Visualisation* to *Turbulence*.

Most important: Casts of this scene are the plants' prefabs from the *Full Demo,* whose materials can be tweaked here way more easily than in the actual scene they are used.

So just hit play, then e..g rotate the wind zone and watch how the global wind texture gets updates and the prefabs react to it. Now you may tweak the materials and set their wind properties.

Wind properties of the Grass shaders [>].

Wind properties of the Foliage shader [>].

Please note: When using Unity < 5.6.3 changing the materials in play mode while the Grass Manager script is rendering the grass will not have any effect because cells are rendered each using a copy of the original material – and these copies do not get updated unless the cells are regenerated.

In order to find out more about wind and the inputs of the wind script please have a look at the chapter Wind [>].

Where to go next

Now that you have made the first steps using AFG it is time to dive deeper into the system.

- Find out more about the Grass Manager script [>].
- Find out more about the Shaders [>].
- Find out more about how to setup and tweak Wind [>].
- Find out more about possible Optimizations [>].
- Model and texture guide [>].

Floating Origin Demo

This example is simply derived from the Full Demo and uses a small script to shift all Objects parented under the "WorldContainer" and the camera according to the distance of the camera to the world's origin.

If you run the demo you will notice that the white and red columns suddenly jump in space when you move around. These jumps indicate a move of the camera and the terrain – whereas the columns do not move at all.

Takeaways

- In order to prevent wind from causing hiccups or pops you have to move the "ATG Wind Prefab" as well.
- The ATG grass manager now uses "LateUpdate". According to the floating origin script you use (which may use "LateUpdate" as well) you may have to adjust the "Script Execution Order" (in "Edit" → "Project Settings" → "Script Execution Order") and make sure that your floating origin script will be executed *before* the grass manager script.

BOTD Demo

The BOTD demo shows how content from Unity's "Book of the dead" demo could be used along with ATG.

Meadow Grass: The grass uses a rather simple mesh but the *Grass Array Shader*. The assigned texture array contains three different grass textures which all of course have to fit the given grass mesh and then are randomly assigned to the grass instances to create some more variation.

Dry Grass and Dry Plant: Both share the mesh and thus can be merged. The texture array contains a dry grass and a dry plant texture. By using *soft merge* we have full control over where dray grass or dry plants get drawn and can even assign pretty different sizes. Bending, rotation and Dry and Healthy color however are the same for both.

Broadleaf Shrub: This plant has pretty flat or horizontal aligned surfaces and should use the *foliage shader* in order to get nice specular highlights.

Fern: Is a rather complex model and uses the *foliage shader*.

Takeaways

- If you use rather generic meshes and make different textures just match these meshes you can merge even quite different plants or simply add more variation by using texture arrays: This gives you a nice performance boost or just some more variation.

Draw in Edit Mode

Since version 1.1. ATG supports grass being rendered by ATG in edit mode.

In order to use this feature your scene has to contain a camera tagged as “MainCamera”. If so you will get a checkbox called **“Draw in Edit Mode”** at the top of the Grass Manager’s inspector. Check it to start ATG in edit mode.

ATG will sync your camera and make it follow the scene camera. But in most cases you won’t see any grass at all right away.

In order to make the grass actually show up just slightly move the editor camera (e.g. by clicking and holding the right mouse button and then dragging the mouse) to force ATG to update the currently visible grass cells.

Painting new Grass

ATG does not automatically update its internal arrays. So if you have checked **“Draw in Edit Mode”** and paint new grass it will not show up – unless you hit **“Update Grass”**. Doing so may take a while so in case you need an instant response I recommend to uncheck **“Draw in Edit Mode”**, rely on the terrain engine instead and only check the grass once in a while by activating **“Draw in Edit Mode”** and hitting **“Update Grass”**.

Please note: In case you have multiple terrains and multiple grass managers in your scene you will have to check “*Draw in Edit Mode”* in all grass managers – or just the one you are interested in.

In case you have a bunch of terrains in your scene it might take several minutes until all terrains are configured. So in this case I recommend dropping the feature. Do so by commenting the line “[ExecuteInEditMode]” in the “GrassManager.cs”.

ATG Components

ATG is a rather simple system and consists of only 3 different components:

- The Grass Manager script which does pretty much everything and especially the rendering.
- The Shaders which are needed to render the data provided by the Grass Manager.
- The Wind script which generates a global wind textures needed by the shaders.

Grass Manager Script

The Grass Manger is the heart of ATG and does almost all the work. It converts height and density from the terrain to the needed internal format, handles all prototypes and prepares and sends data to the shaders.

In order to make ATG take over the grass rendering on any terrain in your scene, simply add the Grass Manager script to it. A first and simple configuration of the script should be done automatically.
Please note: ATG renders grass only when you are in play mode. As long as you are in edit mode Unity's terrain engine will render the grass.

When the initial configuration is done you may tweak the settings using the Grass Manager's inspector:

Edit Mode

Please note: In order to use *Draw in Edit Mode* your scene has to contain a camera which must be tagged as "MainCamera". Furthermore you have to open the scene and game view side by side.

Draw in Edit Mode: If checked ATG will take over grass rendering in edit mode. Otherwise it will only render the grass in play mode. The grass manager will make your camera follow the scene camera in order to draw the grass around the current view position. It will also set *Cameras* to "All Cameras" and disable *Compute* in case it is activated.

Update Grass: When pressed ATG will update the grass density maps and regenerate the grass. Use this to sync painted and rendered grass.

See: Draw in Edit Mode [>]

Functions

Get Prototypes: will read in or update prototypes from the terrain. If no mesh or material is assigned the script will assign a default ATG mesh and a base grass material which gets stored right next to the texture assigned in the terrain engine.
Please note: Min Size, Max Size and Noise as well as Healthy and Dry Color will always be updated unless you check "Freeze Size and Color". So i recommend to change these parameters using the terrain engine, then sync – or check the checkbox below.

Use this button to sync terrain and script settings.

Freeze Size and Color: Check this in order to keep any possible custom changes you made to Min Size, Max Size and Healthy and Dry Color in the final ATG material even if you update the prototypes.

Toggle Grid lets you turn on and off a projector which projects the current grid onto the terrain. This is helpful in case you want to optimize the prototype distribution.

Universal Render Pipeline

Universal Renderr Pipeline: Check this in case you are using the universal render pipeline and enable compute. This will make the compute shader bypass HiZ culling which currently is not supported.

VR: Force Single Pass Instanced

VR: Force Single Pass Instanced: If you are using *Mock HDM* the script is not able to detect "Single Pass Instanced" rendering. So check this in order to get rid of missing grass instances.

Render Settings

Cameras: If set to *All Cameras* grass will be rendered in all cameras, else only the camera tagged as MainCamera will render the grass.
Please note: ATG always renders grass relative to the main camera which has to be tagged as such. But it might draw into other cameras as well (like mirror or reflection cameras e.g.). In case you switch the camera from e.g. a third person to a first person camera, the first person camera has to be tagged as camera.main as well.

Layer: Lets you select the layer the grass will be rendered on. Select a proper layer in case you use layer based culling (Camera.layerCullDistances).

Ignore Visibility: If checked the culling group will get all culling spheres within the specified distance regardless of any visibility.
Please note: Usually it should be unchecked but in case you have any problems with grass not being rendered at all, checking this should solve the issues. Grass not being rendered most likely is caused by the fact that you have a 2nd camera (for the GUI e.g) in your scene. Checking "Ignore Visibility" has only minor performance drawbacks – so you are pretty safe to use it in case you encounter any problems.

Light Probe Proxy Volume: Currently ATG does not support light maps and blending light probes looks either really strange (and blocky) or will produce high costs on the cpu. So you may create a *Light Probe Proxy Volume* which covers the entire terrain and assign it here. Then the grass shaders will sample ambient lighting form this volume instead of always picking up the global ambient lighting from the sky.
Please note: This is rather cheap on the CPU but heavy on the GPU as it will add 3 more texture fetches per pixel. Furthermore *Light Probe Proxy Volume*s have a pretty low resolution of 32x32x32 pixels max – but enough to lit a canyon differently than mountain top or flat plain.

Cull Distance: Distance at which the grass will be culled (equals "Detail Distance" in the terrain settings).
Fade Length: Length over which the grass will fade out before it gets culled.

Detail Fade Distance: Distance at which a user defined amount of instances start to fade out. See two step culling for further details.
Fade Length: Length over which these instances will fade out.

Shadow Fade Start Grass: Distance at which shadows casted by details using the grass shader will start to fade out.
Fade Length: Length over which the shadow casters will fade out.

Shadow Fade Start Foliage: Distance at which shadows casted by details using the foliage shader will start to fade out.
Fade Length: Length over which the shadow casters will fade out.

Cache Distance: Distance at which Cell Contents will be removed from memory.

Use Burst Init: If checked all cells within the specified **Radius** around the main camera will be initialized at start. If unchecked cells will be initialized one each frame.

Cells per Frame: Lets you specify how many new cells will be generated per frame. The recommended value for e.g. a FPS game is "1" as this will have the smallest possible footprint. However in case your camera moves very quickly you may encounter missing or popping in grass patches. Slightly increasing the number of cells per frame will help to solve this problem.
Please note: Although the calculation of the matrices happens on a worker thread, creating a new cell also produces some work on the main thread. So in case you create up to 4 cells per frame instead of 1 the costs of the grass manager's update will most likely increase. Look for "InitCellContentDelegated" in the profiler.

Grass Density: Equals *"Detail Density"* of the terrain settings and allows you to draw less or more grass instances based on the given density in the density maps.

Buckets per Cell: Determines how many "pixels" of the Detail maps will form a cell and therefore the number of cells as well as their size. The smaller the cells the less vertices will be sent but skipped by the GPU as the frustum based culling will do a good job. However – the smaller the cells the more work the CPU will have. I recommend a cell size of 8 – 32m.
In case you use *compute* the ideal cell size most likely will be larger tho.

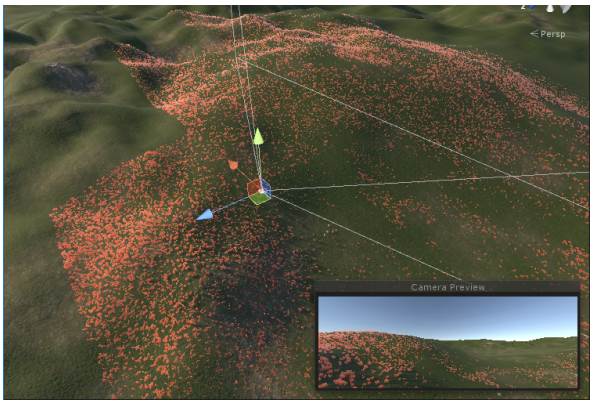
Compute

Since version 1.09 ATG supports using a compute shader which means that all visible cells and layers will be processed by a compute shader which does a distance and visibility check per instance and creates one single draw list for each layer. The visibility test contains a simple frustum culling as well as HiZ culling usually resulting in way fewer instances which actually have to be rendered by the GPU.

Please note: Compute needs Unity >= 5.6.3. (DX11) and Unity >= 2017 (Metal)

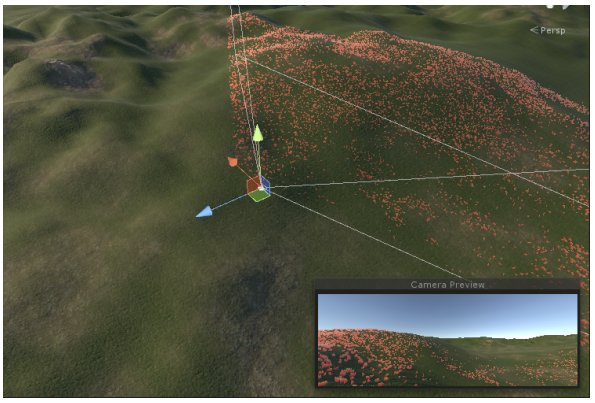
Enable Compute: If checked ATG will use a compute shader to cull individual instances and merge all cells of a layer (prototype) into one huge batch.
Please note: Using compute on a GTX 970M and DX11 gave me significant performance improvements while using it on a GT 750M and Metal lowered my FPS.

Instance Size: As the compute shader performs a super simple frustum culling by just projecting the pivot into clip space you should set "Instance Size" to the max size of any prototype in order to avoid grass being culled although it should be visible.
What it does is simply drawing the instance's pivot forward along the camera's forward vector.



Compute disabled

Grass is culled per visible cell only:
220 FPS on DX11 and GTX 970M at 1080P.



Compute enabled

Grass is culled per cell on the CPU, then culled per instance on the GPU using frustum and HiZ culling:
380 FPS on DX11 and GTX 970M at 1080P.

Memory Usage

As compute creates one huge batch per layer or prototype it needs additional memory in the VRAM to store the final results. Please keep an eye on this: The script outputs the *"Buffer Memory Usage"* to the console at start which is the additional amount of VRAM needed by ATG and compute.

Buckets per Cell

Buckets per Cell define the final cell size. In case you do not use compute you most likely want smaller cells so the CPU will do a better culling job. In case you use compute you should use larger Cells (e.g. 32 – 64m) as the GPU will refine the culling per instance. Furthermore, using larger cells will most likely reduce the memory usage as the final buffer sizes are computed based on a better fitting average amount of possibly visible instances.

Shadows

When using compute only "visible" instances will be drawn by the GPU which also means that only "visible" instances will cast shadows. This will lead to missing and popping shadows especially if the sun is low. Consider raising the *"Instance Size"* to get around this. Doing so will result in less accurate culling though.

Floating Origin

Enable: If checked ATG will support a floating origin in order to allow you to create huge scenes and get around floating point issues.

The grass gets repositioned in the shaders which produces relatively low costs. But as ATG relies on culling groups, all culling spheres actually have to be moved. The costs for this depends on the number of cells (equals number of culling spheres), so using compute and rather huge cells will help to reduce the overall costs on the CPU for moving the culling spheres.

Debug

Debug Stats: If checked the script will prompt the number of queried cells, the number of visible cells and cells which have to be initialized on screen in play mode. In case the number of cells which have to be initialized is greater than 0–4 you will most likely notice some missing cells on

the terrain or cells simply popping in. This indicates that your camera is moving faster than the script can initialize new cells. Consider raising the cell size slightly – although this will shift some more work to the GPU.

- **Queried Cells:** Number of cells culled by the culling group based on distance and – if *Ignore Visibility* is unchecked – visibility.
- **Visible Cells:** Actual number of visible cells. This number might be different from the number of *queried cells* in case *Ignore Visibility* is checked.
- **Cells to init:** Number of cells which have to be initialized on the worker thread. If this number does not change over time the worker thread most likely has produced an error.

Debug Cells: Checking this lets you preview the cells' status in game view (you have to activate Gizmos in the Game view window as well). Cells will be drawn as flat boxes.

- **Solid, overlayed green cube:** Initialized and cached cell.
- **Cube with green outline:** Visible and actually drawn cell.
- **Cube with red outline:** Visible but not drawn cell as it is has not been fully initialized yet.

Saved Terrain Data

Saved Terrain Data lets you write the merged density maps and calculated cell structure to disk as a scriptable object. Reading that information from the scriptable object will speed up the initialization of the terrain: I saw times going down from 4s to 0.004s. If no data is baked the script will do all needed calculation at start. So Saved Terrain Data is kind of **mandatory** in case you want to **spawn terrains at runtime**.

Please note: Whenever you do any changes to the grass distribution, the number of prototypes or cell size you have to rebuild the saved terrain data.

Tip: *Do not save any terrain data while you are working on the terrain. Just do it before you write a build.*

Prototypes

This section lets you edit and set up the prototypes. You may choose between two different views:

- **Single Prototype** will show only one prototype which can be selected from the thumbnails above. It shows the ATG related properties as well as the related material editor and is pretty handy to set up all properties at once.
- **All Prototypes** shows all prototypes in a listview and lets you quickly compare settings like size or choose a proper layer to merge with.

Mesh: The mesh used to draw the prototype. You may assign a custom mesh here at any time. In case a mesh is assigned, hitting "Get prototypes" will not overwrite the existing assignment.

Material: The material used to draw the prototype. You may change the material according to your needs but make sure it uses one of the ATG shaders. In case a material is assigned, hitting "Get prototypes" will not overwrite the existing assignment.

Rotation Mode:

- **Aligned Random Y Axis:** Grass will be aligned to the terrain normal which looks nice for shorter grass and actually is the fastest way to draw it.
- **Aligned Random YX Axis:** This is only suitable for objects using the vertex lit shader like small rocks which shall follow the terrain normal but also get a second rotation.

- **Not Aligned Random Y Axis:** Instances will always be vertically aligned which fits the way Unity renders grass. It is more expensive than **Aligned Random Y:**
  In order to mimic Unity's grass rendering you will also have to check **Write Normal Buffer** below and active **Use Normalbuffer** in the associated material.

  Write Normal Buffer: Check this in case you use **Not Aligned Random Y Axis** and want to store the terrain normal in order to use it to light the instance. Otherwise the instance will be lit using a normal simply pointing upwards. This feature is only supported by the grass shaders.

Cast Shadows: Lets you determine whether a prototype shall cast shows or not. Shadows of course are quite expensive, so – if you enable them on some of the prototypes – please adjust the Shadow Fade properties properly.
Please note: Shadows tend to flicker when using Unity < 5.6.3.

Please note: The following 3 values will be overwritten whenever you press "Get Prototypes" – unless you check "Freeze size and Colors". So I recommend adjusting them using the built in terrain tools.

Min Size is derived from *Min Height* in the *Edit Grass Texture* or *Edit Grass Detail* window of the terrain engine. Min Width is ignored as we can only uniformly scale instances.

Max Size is derived from *Max Height* in the same windows.

Please note: Min Size must not equal Max Size in case you use *two step culling* and set *Clip Threshold* (in the material) to > 0.0 as otherwise the grass shaders will fade out all instances early on.

Noise equals *Noise Spread* in the same windows.

Layer to merge with determines the layer you would like this prototype to be merged with on start or when you save the terrain data. 0 means that the layer will not be merged as the index starts with 1. Merging Layers [>]

Soft Merge If checked the script will not skip the layer's density map but combine it with the one of the layer it is set to be merged with at runtime. Using this option will cost some more memory but gives you full control over the distribution of the merged layers.
Merging Layers [>]

Shaders

The package ships with various terrain detail shaders which covers a pretty wide range of different use cases. All shaders however have in common that they support procedural instancing which – in short – means, that they can read needed input data from Structured Buffers rather than MaterialPropertyBlocks.

In this case all shaders will read the "unity_ObjectToWorld" matrix from a compute buffer called "GrassMatrixBuffer".

Apart from the VertexLit shader all shaders support translucent lighting, specular reflections and advanced wind features.

The latest ATG version also contains some *Depth Prepass Shaders* which may slightly improve performance. Find out more [>]

Grass Base

This is the base shader which gets applied automatically to any terrain detail setup to use the built in grass shader.

Shader Inputs

Albedo (RGB) Alpha (A) expects the main diffuse texture (RGB) and transparency (A)

Alpha Cutoff If the alpha channel of the Base texture contains different shades of gray instead of just black and white, you can manually determine the cutoff point by adjusting the slider.

Healthy Color (RGB) Bending (A) and **Dry Color (RGB) Bending (A)** Just like you may now from the built in grass shader the ATG grass shader supports tinting grass using two different colors which get applied according to the given noise specified in the layer settings of the Grass Manager script. While RGB of these colors will affect the tinting, alpha may reduce bending or wind strength.

Use NormalBuffer If checked the shader will use the passed terrain normals to mimic the lighting you are used to getting from the built in grass. This is only needed and useful in case the according grass instances do NOT use *Rotation Mode: Aligned Random Y Axis* and you have checked *Write Normal Buffe*r in in the layer settings of the Grass Manager script.

Bend Normal Lets you define how strong wind affects lighting.

Enable specular Highlights If checked the shader expects the following texture to be assigned. If unchecked translucency is solely controlled by *Translucency Strength* and specular highlights will be disabled.

Trans(R) Spec Mask (G) Smoothness (B) A combined texture which contains:

- Translucency in the red channel.
- a Specular Mask in the green channel.
- Smoothness in the blue channel.

You have to check *Enable specular Highlights* to make the shader sample this texture.
Please note: This texture should be imported using *sRGB (Color Texture)* **unchecked**.

Translucency Strength acts as a factor which gets multiplied with the translucency value sampled from the *Trans(R) Spec Mask (G) Smoothness (B)* map and lets you fine adjust final translucency. If *Enable specular Highlights* is unchecked this will be the only value to adjust the translucency strength.

Clip Threshold determines the amount of instances which will get culled early on by the built in two step culling.

> **Enable Debug** If checked the shader will tint all instances which are affected by the two step culling using the *Debug Color*.
>
> **Debug Color** lets you specify the color used if *Enable Debug* is checked.
>
> **Please note:** Min Size must not equal Max Size (in the Prototype settings) in case you use *two step culling* and set *Clip Threshold* to > 0.0 as otherwise the grass shaders will fade out all instances early on.

Scale Mode (built in and URP 10 and 11 only) Lets you choose if details fading out are being scaled along XYZ or XZ only. Using XZ may give you nicer results here. Far distance based fade always uses XYZ to give you the best visual results.

Wind

Strength is the final multiplier for the actual wind strength applied.

Sample Wind at Pivot If checked the shader will sample the wind strength at the pivot for all vertices of the given instance. Otherwise it will sample the wind strength at the position of each vertex. Use this feature in case you do not want any distortions of the original mesh.

Wind LOD (int) lets you specify the LOD level at which the global wind texture gets sampled. Higher LOD levels will create smoother bending.

Grass Array

This shader directly derives from the base shader but uses texture arrays instead of just a single texture. Use this shader to simply create some more variation or in case you merge layers.

Shader Inputs

Most shader inputs match those of the base shader. So I would like you to have a look there. Special inputs are:

Albedo Array (RGB) Alpha (A) Here the shader expects a texture array

Number of Layers -1 (int) The number of textures contained by the texture array **minus one**. You have to specify the number of layers stored in the texture array here. As the index starts at 0 you would have to enter 1 in case your array has 2 layers — or 2 in case the array has 3 layers. This will lead to a more or less evenly distribution of all textures stored in the array.
But actually you can even foul the shader. Imagine you have 2 layers in the array so the correct number to enter would be … 1:) Setting it to 2 instead would make the shader pick up the first texture for ⅓ of the instances and picking up the 2nd texture for ⅔ of the instances.
This always only affects the last texture layer in the array, as the shader simply will pick up the last texture if the calculated index is larger than the number of actually stored layers.
Please note: You only have to set this value in case you use *Texture Mix Mode = Random* or *BySize*.

Texture Mix Mode lets you determine how the shader mixes the textures of the texture array:

- **Random** Textures will be mixed randomly (needs slightly different sizes tho).
- **BySize** small instances will use layer 0 of the texture array, mid size instances layer 1 and large instances layer 2. Actually I have not checked this for more than 3 layers…
- **SoftMerge** select this in case you use soft merged layers. Then the shader will pick a texture from the array based on the original layer.

In order to find out more about texture arrays, please have a look here.

Foliage

The foliage shader derives from the AFS advanced foliage shader and supports more advanced manually authored wind bending and truly physically based shading – compared to the grass shaders. It is the most expensive ATG shader but allows to even render more complex models like the fern in the *Full Demo*.

Models must follow the latest AFS vertex color bending requirements which means that:

- primary bending is stored in vertex color alpha.
- secondary bending is stored in vertex color blue.
- phase is stored in vertex color red.

- edge flutter is stored in vertex color green.

To find out more please have a look at the Model and Texture Guide.

Shader Inputs

Culling Use “Off” in case you use single sided geometry (recommended). “Back” would be the correct choice for double sided geometry. “Front” is available just because it would be the third possibility…

Albedo (RGB) Alpha (A) Main diffuse texture (RGB) Transparency (A).

Healthy Color (RGB) Bending (A) and **Dry Color (RGB) Bending (A)** Just like you may now from the built in grass shader the ATG grass shader supports tinting grass using two different colors which get applied according to the given noise specified in the layer settings of the Grass Manager script. While RGB of these colors will affect the tinting, alpha may reduce bending or wind strength.

Bend Normal Lets you define how strong wind affects lighting.

Enable specular Highlights If checked the shader expects the following texture to be assigned. If unchecked translucency is solely controlled by *Translucency Strength* and specular highlights will be disabled.

Normal(GA) Trans (R) Smoothness (B) A combined texture which contains:

- the normal in the green and alpha channel.
- translucency in the red channel.
- smoothness in the blue channel.

To find out more please have a look at the Model and Texture Guide.

Smoothness Smoothness applied in case you have not assigned/enabled the *Normal(GA) Trans (R) Smoothness (B)* texture.

Specular Reflectivity lets you define the specular color.

Horizon Fade Image based reflections might produce some kind of “light leaking” on normal mapped surfaces as the final reflection vector might point *behind* the actual surface resulting in reflections which simply are “impossible”. Horizon Occlusion fixes this.

Read more here: http://marmosetco.tumblr.com/post/81245981087

Translucency Strength acts as a factor which gets multiplied with the translucency value sampled from the *Normal(GA) Trans (R) Smoothness (B*) map and lets you fine adjust final translucency. If *Enable specular Highlights* is unchecked this will be the only value to adjust the translucency strength.

View Dependency lets you control the strength of the translucent lighting according to the given view angle. Low values will add a lot of translucent lighting even if the view angle is rather steep, while higher values will add translucent lighting only, if you more or less just directly look at the front of the given triangle.

Wind

Strength Primary (X) Secondary (Y) lets you fine adjust wind:

- X is the final multiplier for the actual wind strength applied to primary bending.
- Y is the final multiplier for the actual wind strength applied to secondary bending.

Sample Wind at Pivot If checked the shader will sample the wind strength at the pivot for all vertices of the given instance. Otherwise it will sample the wind strength at the

position of each vertex. Use this feature in case you do not want any distortions of the original mesh.

Wind LOD (int) lets you specify the LOD level at which the global wind texture gets sampled. Higher LOD levels will create smoother bending.

VertexLit

This shader will be automatically assigned to all details using the built in VertexLit shader. It does not support translucent lighting nor any bending and is suitable for static objects like small rocks, sticks or any other kind of greeble which shall not sway in the wind.

Shader Inputs

Albedo (RGB) Smoothness (A) While RGB should just contain the regular albedo texture, alpha here should contain a smoothness map.
Healthy Color and **Dry Color** Just like you may know from the built in grass shader the ATG vertexLit shader supports tinting instances using two different colors which get applied according to the given noise specified in the layer settings of the Grass Manager script.

Specular Reflectivity determines the specular color used by the shader.

Enable Normal Map if checked the shader will sample the following texture.

Normal map Just a regular normal map.

Depth Prepass Shaders

ATG ships with the *"Grass_InstancedIndirect_Array_DepthPrepass"* and the *"Grass_InstancedIndirect_Base_DepthPrepass"* shaders. These shaders perform a (cheap) depth pass first, then they do the (expensive) regular pass. Doing so is common in Unity's HD SRP and other game engines when it comes to alpha tested geometry as alpha testing does not let us benefit from early depth testing by design.

And although these shaders will render the geometry twice, they may actually *slightly* increase performance depending on the amount of overdraw which is produced by the grass, thus is connected to your camera perspective: The lower and closer the camera is to the ground and the higher the grass density is the bigger the chance to speed things up – and may it just be 2–5 FPS.

Please note: The depth prepass shaders are only suitable when you use deferred rendering.

Wind

All ATG shaders which support wind are driven by a **global wind texture** which contains a slightly modulated wind strength and some additional noise just for the grass shaders in order to create some more distinguished gusting wind.

This wind texture is a render texture and gets updated each frame which enables us to rotate the wind direction at runtime without creating weird bending grass instances.

Using a wind texture allows us to have spatial differences concerning wind strength and gust without having to implement any fancy per instance logic in the vertex shader and gives us a solid and shared foundation for grass, foliage and maybe even tree shaders.

Wind.cs

The global wind texture is created by the *Wind.cs* script, which has to be assigned to your main directional wind zone as it gets its main inputs like wind turbulence and direction from that wind zone:

Turbulence from the wind zone settings will affect the contrast of the global wind texture while the rotation of the wind zone will drive the scroll direction. *Main* (wind strength) from the wind zone settings will be directly pushed to the grass and foliage shaders.

Either add this script to your main wind zone or – in case you have none – drag the *ATG Wind Prefab* into your scene.

Inputs

Resolution: Resolution of the created global wind texture. Keep it as low as possible to save resources like memory and GPU time – but 512 just should be fine.

Wind Base Tex: In order to be able to create a global wind texture you have to provide a proper base texture. ATG ships with a default wind base texture (*Default Wind Base texture.psd*). This texture contains various grayscale perlin noise textures at different scales in each of the color channels (RGBA). While RGB will only be used to calculate the final wind strength the texture assigned to the alpha channel (A) will contribute to the wind strength but also determine the wind gusting added in the ATG grass and foliage shaders.

Wind Composite Shader: Shader which is used to render the final global wind texture. Assign the included WindComposite shader – or a proper custom one.

Wind Multipliers

> **Grass:** Global multiplier for the wind strength as retrieved from the assigned wind zone and applied in the Grass shaders.
>
> **Foliage:** Global multiplier for the wind strength as retrieved from the assigned wind zone and applied in the Foliage shader.

Size: Determines the area in world space the global wind texture covers before it gets tiled and repeats. Actually it is: 1.0f / Size in meters, so 0.01 = 100m x 100m.

Speed: Speed at which the global wind texture will “scroll” in world space. The actual speed depends on the used *Size* as *Speed* is measured in UV space. So in case you change *Size* you will have to adjust *Speed* as well.

Speed Layer 0, Speed Layer 1, Speed Layer 2, Grass Gust Speed These are multipliers for the overall set *Speed*. In order to create a lively global wind texture you must use varying values here.

Grass Gust Tiling: RGB channels of the assigned *Wind Base Texture* will be sampled using fixed UVs as they already contain perlin noise textures at various scales – however you may finetune gusting wind as applied by the grass shaders by changing this parameter.
Higher values will create smaller waves of gusting wind.

Jitter Frequency: Defines the frequency of a small scale non directional jittering added on top of the wind animation calculated based on the wind render texture – which is directional. Jitter helps to hide the quantization of the samples and adds some more interesting details.

Jitter High Frequency: Frequency of a small scale non directional jittering added on top when gusts of wind blows over the grass. High frequency jitter will make grass “shiver”.

Wind as calculated by the built in grass shader

```
Final Strength =     Bending (from “Wind Settings for Grass”)
                   * Vertex.Color.a (from Mesh)
                   * Healthy/Dry.Color.a (from Instance)
```

Wind as calculated by ATG grass shaders

```
Final Strength =     Main (from WindZone)
                   * Grass/Foliage Multiplier (from ATG Wind)
                   * Vertex.Color.a (from Mesh)
                   * Healthy/Dry.Color.a (from Instance)
                   * Wind Strength (from Material)
                   * Wind Sample (from Render Texture)
```

This looks far more complex but:

a) does not really add much costs to the ATG shaders.
b) defines a single master input which is *Main* from the WindZone.
c) gives you a lot of control.

Setting up Wind

As all parameters mentioned above are quite theoretical I have added a **ATG Wind Viz Prefab** which lets you preview the generated global wind texture.

Please open the scene **03 Wind Demo**.

This scene contains the **ATG Wind Prefab** as well as the **ATG Wind Viz Preb**. Latter is just a simple plane using the **Wind Visualisation material** and **WindViz** shader, so it will show the generated global wind texture.

By default it shows the *main wind strength*. In order to view the *noise* which will drive the gusting you have to edit the *Wind Visualisation* material and change *Visualisation* to *Turbulence*.

The scene also contains several prefabs from the *Full Demo,* whose materials you could tweak here way more easily as in the actual scene they are used.

Model and Texture Guide

All terrain details get drawn by the Grass Manager using Graphic.DrawMeshInstancedIndirect which needs the submesh index to be specified. This is set to 0. So please make sure that your meshes do not contain multiple materials.

As grass usually produces a lot of overdraw I recommend not using simple quads but custom meshes which tightly enclose the space actually used in the assigned texture.
Adding some more vertices is definitely worth it.

Tip: *Even if you have assigned a prototype as a simple texture to the terrain you may just assign a custom and optimized mesh in the Grass Manager.*

Importing custom grass meshes

In case you use custom meshes on grass these meshes need vertex colors to be set up properly. Actually you just have to bake bending values to vertex color alpha:

- **vertex color alpha = 0** means no bending at all. Assign this at the lower vertices of your mesh.
- **vertex color alpha = 1** means full bending. Assign this at the upper vertices of your mesh.

Additionally the grass shaders support slightly different bending driven by **vertex color red**. When sampling the global wind texture the final position will be calculated adding vertex color red on top – which means that if a vertex had vertex color red = 1 and another vertex color red = 0 they would sample the wind texture at different positions in worldspace with a distance of sqrt(1+1).

That being said, I recommend using only very small differences in vertex color red. The grass in the full demo e.g. uses vertex color red = 1 and 0.95.
Please note: You should keep vertex color red as blue and green as well close to 1.0 as otherwise the preview in edit mode might look totally off.

Importing custom foliage meshes

The foliage shader supports more advanced bending and shading.
As far as bending is concerned, it follows the latest AFS requirements which means:

- primary bending is stored in vertex color alpha.
- secondary bending is stored in vertex color blue.
- phase is stored in vertex color red.
- edge flutter is stored in vertex color green.

In order to find out more, please go to the AFS documentation and look for bending mode = vertex colors.

Textures

Both the grass as well as the foliage shader have one quite special texture input:

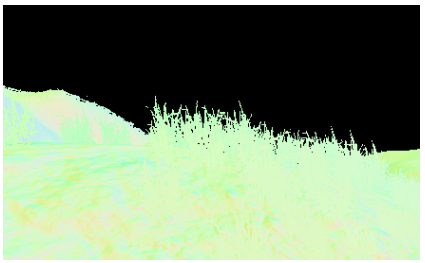
Grass Shader

Trans (R) Spec Mask (G) Smoothness (B) A combined texture which contains:

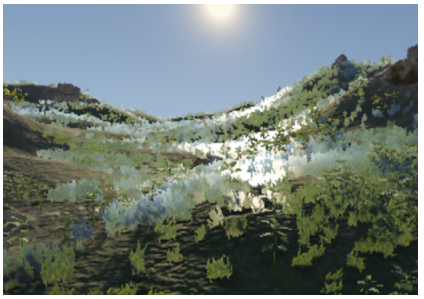
- the translucency mask in the red color channel.
- the specular mask in the green color channel.
- the smoothness map in the blue color channel.

While you should be familiar with the translucency mask and smoothness map the specular mask is quite special. It is needed to add some physically inspired specular highlights on grass.

Grass does not have face aligned normals but normals which are set to match the normal of the underlying terrain in order to give you a smooth and matching lighting.



Deriving specular highlights from such normals usually would end up in some very strange reflections as grass will reflect light as if it was a flat surface.

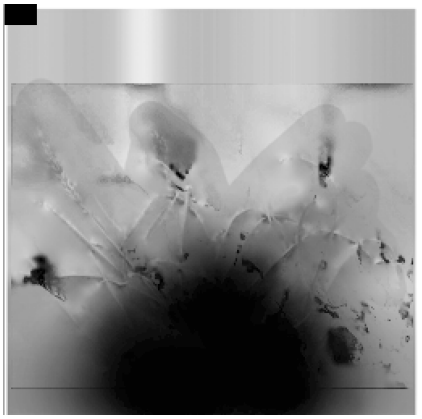


By adding a simple mask which suppresses specular highlights at parts of the grass texture which definitely do not point upwards we can easily get rid of this and create believable highlights even with such normals.

The lighting is then handled by the ATG BRDF properly.

The actually used specular mask in this case is quite generic and does not have anything to do with the used albedo texture: It simply masks out all highlights at the bottom and adds some “shading” at the upper parts.

Tip: *You may however do much better and derive the specular mask e.g. from a normal map. Just pick the green channel which contains the up/down component and tweak it to your likings.*

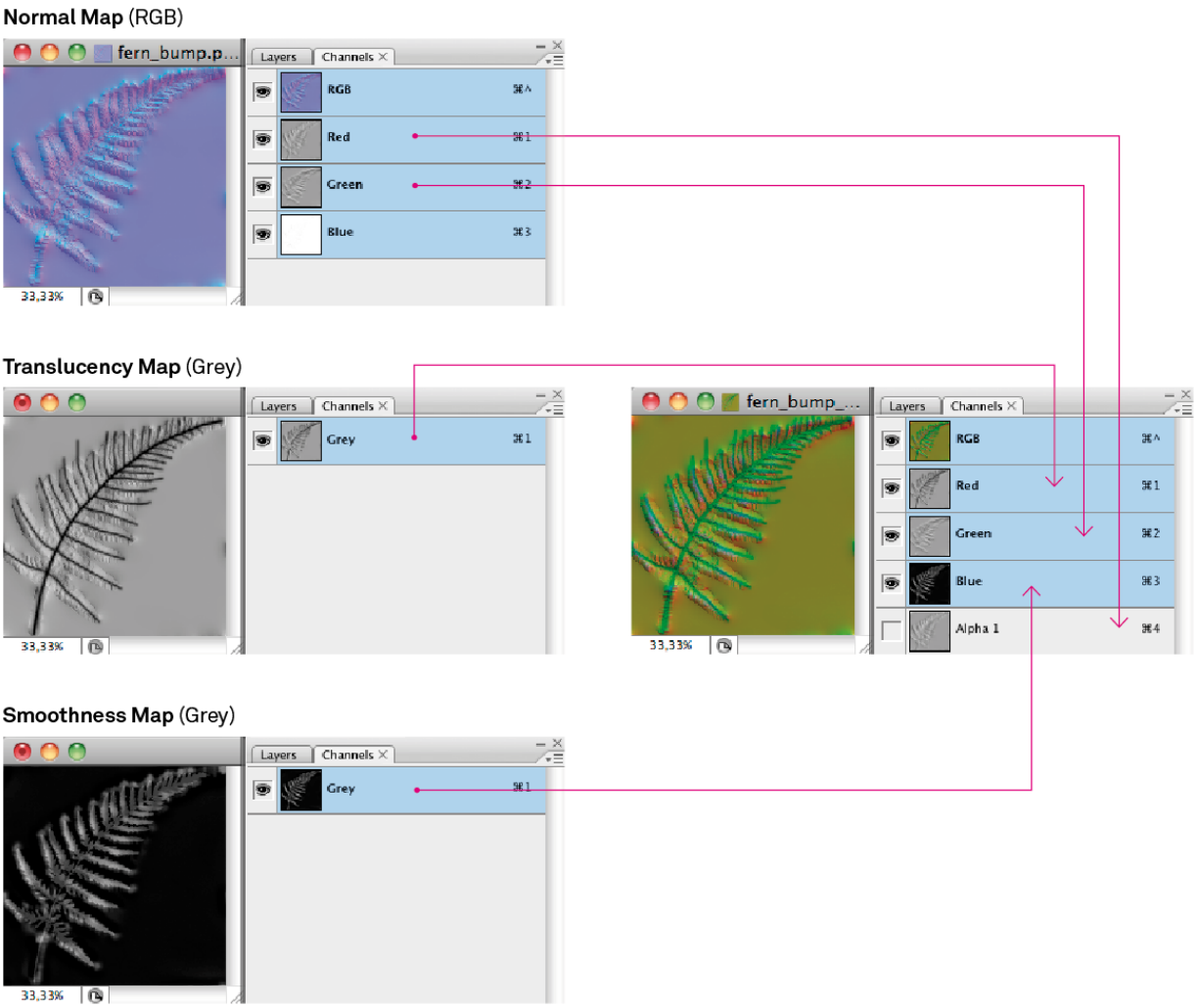


Foliage Shader

Normal (GA) Trans (R) Smoothness(B) A combined texture which contains:

- the normal in the green and alpha channel.
- translucency in the red channel.
- smoothness in the blue channel.

You will have to manually combine this texture e.g. using photoshop like shown in the image below.



Combined Normal/Translucency/Smoothness Texture Color Channel Layout

Please note: These combined textures do not contain a color or albedo texture. So *sRGB (Color Texture)* must be unchecked in the import settings.
As at least the **Normal (GA) Trans (R) Smoothness(B)** texture contains a normal map you should set the *Filter Mode = Trilinear*.

Optimization

Optimizing grass meshes

Using just the standard quad by simply adding a texture is probably the worst thing you can do — as you will most likely waste a lot of fill rate: Hardly any grass texture will fully fill that quad. Use custom meshes instead, which tightly enclose the actual texture.

Do not care about the number of vertices that much: Adding 2 or 3 more vertices by creating a proper and better fitting shape should still improve rendering performance, because it will help to save fill rate.

Furthermore: Only using a single plane per grass instance will need you to crank up the density which slows down the rendering. So I highly recommend creating a small cluster of 6–12 nicely

tessellated planes (which help to save fill rate and improve bending) and use this as a grass prototype.

Optimizing Draw Calls

As explained in **Draw Calls** ATG can not merge different prototypes within a given cell into one single mesh like the built in grass system – but usually invokes one draw call for each prototype.

In order to reduce the number of draw calls you may consider optimizing the cell size, your prototype distribution (by getting rid of just 2 single instances of a prototype in a given cell) as well as merging various layers into just one.

Optimizing Cell Sizes

In order to set the cell size you have to specify the number of **Buckets per Cell** – which will calculate the final cell size automatically.

The smaller the cells the less vertices will be sent but skipped by the GPU as the frustum based culling will do a good job. However – the smaller the cells the more work the CPU will have, because smaller cells means:

- more draw calls.
- more materials which have to be instantiated (one per cell * per layer).

So the best cell size depends on the fact if you are a) CPU bound or b) GPU bound.

If you are CPU bound, consider using larger cells. If you are GPU bound consider using smaller cells. Rule of thumb: A cell size of 8 – 16m should be fine.

When using **compute** larger cell sizes are preferable.

Optimize prototype distribution

In order to avoid causing an additional draw call for just a few instances of a specific prototype you may finetune the distribution of your prototypes by taking cell borders into account: If there are only e.g. a few flowers leaking from one cell into another – just get rid of these. This may be a time consuming task but worth it, especially in areas where you encounter performance problems.

Tipp: *Use the projected grid to visualize the cells on the terrain.*

Try to keep the number of prototypes per cell as low as possible. 4 –6 should be fine. 12 would be bad. In case you need more variation within the single cells use texture arrays and/or merge layers.

Merging Layers

This option is only suitable for prototypes which can at least **share the same mesh.**

Merged layers will be drawn with **just a single draw call**, so they use the same material as well as the same mesh. That being said, different settings in the original materials like Wind Strength or Healthy and Dray Colors will be lost.

This of course will give you less variety as all merged layers will be drawn using the texture from the layer they are merged with — unless you make this layer use the **Grass Array Shader** and

assign a texture array, which contains the textures of all merged layers. The array shader then will pick one of the textures from the array, bringing back variety.

In order to merge layers you have to set **Layer to merge with** in the prototype section of the grass manager. You do not have to set this for the target layer. Leaving this at “0” means that this layer will not be merged. You only have to set it on the layer you want to be merged.

Simple Merge

In case you **simply merge** layer A) with layer B) the grass manager will entirely skip layer A) — but adds its density to the density of layer B).

Size settings as well as the distribution of layer A) will be lost.

This mode will entirely skip one detail layer resulting in less memory consumption.

See image 2).

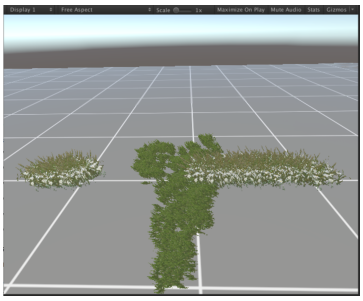
Soft Merging Layers

You can also **soft merge** layer A) with layer B). In this case the density map of layer A) will not be removed. This option needs more memory but allows us to recover the original distribution and size settings of layer A).

In order to actually recover the original distribution of the merged layer A)

- layer B) must use the **Grass Array Shader.**
- the order of the textures in the texture array must match the order of the layers.

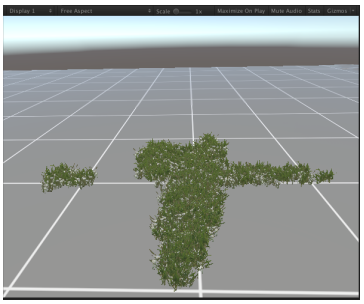
See image 3).



1) Edit mode

Green Grass and *Grass with white flowers* are added as 2 separate prototypes.

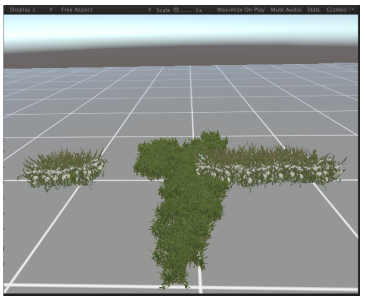
They have slightly different sizes and uneven distribution.



2) Simple Merge in play mode

As *Grass with white flowers* is set to be merged into *Green Grass* its **density gets added** to the density of *Green Grass*.

As *Green Grass* uses a **texture array** which contains both textures and *Texture Mix Mode* is set to *Random* the textures of *Green Grass* and *Grass with white flowers* will be randomly mixed.



3) Soft Merge in play mode

Using soft merge the **original distribution and size** of *Grass with white flowers* is kept.

Texture Mix Mode must be set to *SoftMerge*.

Merging – Step by Step

1. Create the needed texture array.
2. Select the material of the grass prototype which the other prototypes shall get merged to (target).
3. Change the shader of this material to the *Grass Array Shader.*
4. Assign the created texture array to the slot: *Albedo Array (RGB) Alpha (A).*
5. Now you have to specify which other prototypes shall be merged with the target prototype. Do so by e.g. selecting the 2nd prototype, then set its *"Layer to merge with"* parameter from 0 (not merged) to 1 (merged with 1st prototpye (target)). Index starts at 1 here.
6. Check *"Soft merge"* to keep the original distribution of the different prototypes.
7. Make sure that the shared material which uses the Grass Array Shader is set to: *Texture Mix Mode = SoftMerge.*

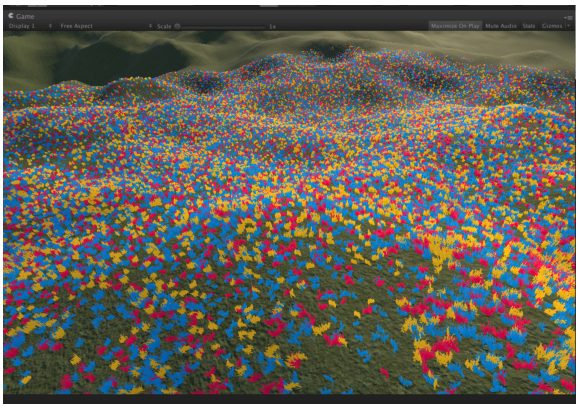
Use Compute

A pretty powerful option to boost performance is using Compute [>] in case your target platforms support compute shaders. At least on DX11 this may give you a quite significant increase regarding fps.

Texture Arrays

ATG ships with the **Grass Array Shader** which lets you assign a texture array instead of just a single grass texture. Doing so you may easily increase variety without having to add more and more prototypes to your terrain.

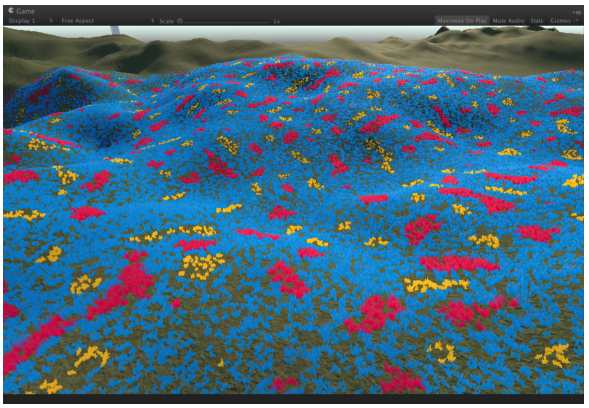
Furthermore texture arrays are needed in case you merge layers as described in the section above.



Textures randomly mixed

If *Mix textures based on size* is unchecked the shader will randomly mix all textures from the array. This will give you an even distribution of all textures.

Please note: Randomly mixing textures needs you to apply at least slightly different min and max sizes.



Textures mixes based on size

As the size of the instances varies based on perlin noise the mixing of the textures will reflect this noise and textures will mix in a more clustered manner. The size of the clusters reflects the size of the noise.

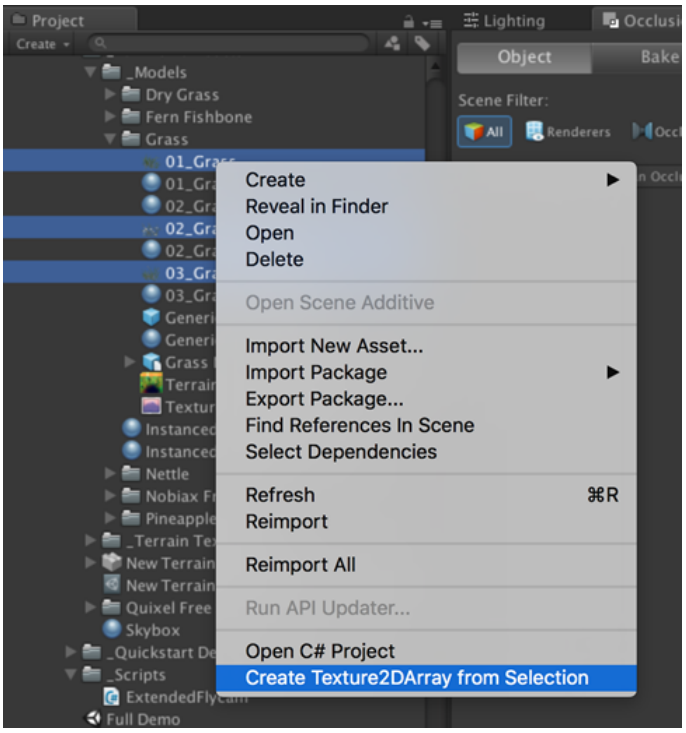
Please note: In this example we mix 3 textures and the 2nd one (blue) wins in most cases. *Size Threshold* is set to 0.625 so we get more instances using the 3rd texture (red) than using the first one (orange). Keep this in mind when creating the texture2DArray and order the textures accordingly.

Please note: In case textures are mixed based on size Healthy and Dry Color will be applied randomly and NOT based on perlin noise as otherwise it would look pretty boring.

Creating Texture Arrays

In order to create texture arrays simply:

- select the textures in the project tab.
- click the right mouse button to bring up the context menu.
- choose: *Create Texture2DArray from Selection* or *Create linear Texture2DArray from Selection.*



All selected textures have to match in size, color mode etc. In case they do not the script will show a detailed error message. Please fix all issues and try again.

In case everything is fine the script will bring up the safe dialog and lets you choose a location and filename.

Textures will be layered in the array according to the “alphabetical sorting” of their file names:

01_test.tga → layer 0

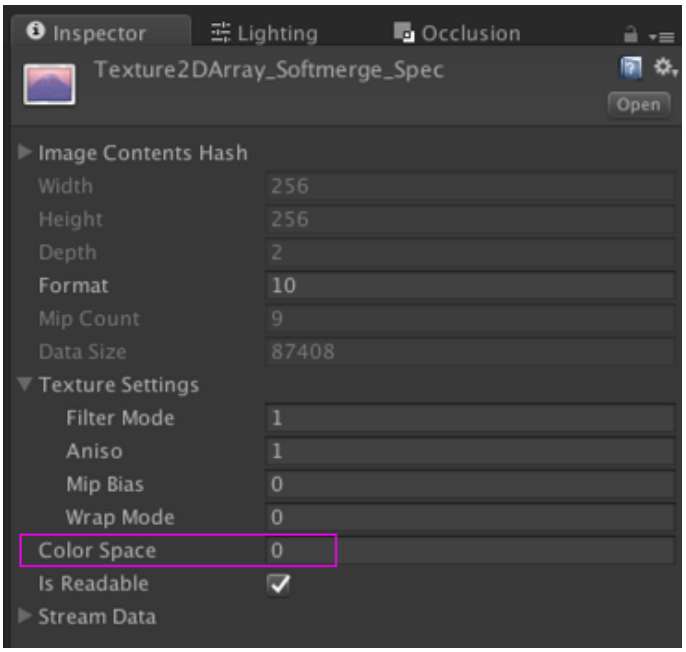
Abc.tga → layer 1

Flower.tga → layer 2

Tip: *In order to make sure textures will be stored in the correct order, prefix their filename using "01_", "02_" etc.*

~~**Please note:** In case you merge combined textures like the **Trans (R) Spec Mask (G) Smoothness (B)** for the grass array you have to manually set the *Color Space* to 0 – which equals unchecking "sRGB (Color Texture). In order to do so select the generated texture array and change the value accordingly. This is nasty but Unity simply does not have any API for doing this automatically.~~

Please note: Unity 2018.x? removed the possibility to change the *Color Space*… So now ATG contains a script which lets you explicitly save linear texture arrays.



ATG and third party packages

You may use certain ATG shaders and models with third party products like Vegetation Studio in order to benefit from ATG's advanced lighting and bending functions.

In order to do so you will always have to add the *ATG Wind Prefab* to your scene and set it up properly as all ATG shaders pick up wind only from that prefab and script. ATG wind >

Models, materials and textures just have to be set up according to ATG's specifications. However the shaders may support special vertex color inputs according to the given third party package as well.

Finally do not forget to assign the ATG deferred lighting and reflection shaders in case you use deferred lighting.

Vegetation studio

ATG contains 2 shaders which are dedicated to and compatible with Vegetation Studio: The base grass shader (VS_Grass_InstancedIndirect) as well as the foliage shader (VS_Foliage_InstancedIndirect). Both shaders let you benefit from ATG's advanced lighting and bending features even if all the placement and rendering is done by Vegetation Studio.

So in case you want Vegetation Studio to take over the rendering of all your grass you have to assign one of these shaders to your prefabs which then are used by Vegetation Studio.

Please note: Both shaders only work properly if you set the *Render mode* of the corresponding prefabs to *Instanced indirect* in the *Vegetation* Settings of Vegetation Studio. Using *instanced* only is not really supported as it does not make much sense and will slow down rendering.

The shaders expect wind inputs as provided by the ATG wind script.

Shader Inputs

The basic shader inputs are just the same as for regular ATG shaders. But you will find some new parameters listed under *VegetationStudio Specials* such as touch bending:

Ignore Fading (VS Pro): As VS Pro seems to have dropped the support for fading out details over distance you will have to check this in case you use the ATG shaders with VS Pro.

Use touch bend: If checked the shaders will support Vegetation Studios' touch bending.

> *Foliage Shader only* **Force (X) Radius (Y)** lets you determine the touch behaviour. *Force* equals the strength of the touch in relation to the displacement while *Radius* lets you define some kind of falloff. Set *Force* to your liking but keep *Radius* between 1 and 2.
>
> Overall however i have to admit that Vegetations Studio's touch bending does not allow us to derive any valuable information for more complex models such as a fern so touch bending will always look a bit odd.

The **Grass Base Shader** also lets you select the **Vertex Color Mapping**:

- If set to ***VS*** main bending should be stored in vertex color red and phase shift in vertex color green – just like in the grass meshes provided by VS.
- If set to ***ATG*** main bending should be stored in vertex color alpha and phase shift in vertex color red.

The **Foliage Shader** always expects bending information stored according to AFS documentation.

ATG and the Universal Render Pipeline (Preview)

Introduction and Update Guide

Since version 1.2 ATG comes with support for the Universal Render Pipeline which can be used with Unity 2019.3.b and above. Please note that URP is currently in preview.

In order to add support for URP follow these steps:

- Import the "ATG_URP_7.1.7_Support" or "ATG_URP_7.2_Support packages according to the URP version you use. They will add the "ATG URP Shaders" to your project as well as a URP Full demo scene which is ready to explore (02 Full Demo URP). The grass and foliage here uses materials which already have the new shaders assigned.
- Of course your project needs to be using the Universal Rendering Pipeline...

Updating your scene:

- If the URP is assigned in your *Project → Graphics Settings* check *Universal Render Pipeline* at the top of the *Grass Manger* inspector. This will prevent the grass manager from looking for the HiZ component which is not supported yet.

- Go through the used materials and change them to the AdvancedTerrainGrass URP shaders.
- Of course all other shaders used in your scene have to use URP compatible shaders as well. Please note: Unity's default URW terrain shader will show flipped normals...
- You may activate *compute* which will combine all cells and compute visibility according to the frustum. However it will not calculate visibility based on the HiZ buffer.
- You do not need the *Atg HiZ Buffer* script as it is not supported currently.
- The grid projector does not work using URP.
- **Please note:** As some features have changed you will have to adjust your materials.
    - If your grass material **does not use a *Normal(GA) Trans (R) Smoothness (B)* map** you have to **uncheck the *Sample Mask Map***. Otherwise the shader will sample a black texture which adds some unnecessary costs. Make sure you edit the **smoothness** and **transmission strength** so it does not look like a self illuminated mirror…
    - You may have to slightly adjust **wind settings** as well.
    - As lighting in URP works slightly differently you may raise the specular color of all grass materials to **get more specular reflections and highlights**.

ATG URP Shaders work slightly differently and have to be assigned manually.

- The grass and foliage shaders let you turn off alpha testing in case you use fully modelled grass blades like in ***Zelda: Breath of the Wild*** or low poly plants.
- There is only **one, unified grass shader** which supports either single textures or texture arrays. Just check *Enable Texture Arrays* in the material inspector if you need them.
- **Randomly mixed textures** are not supported. The grass shader in array mode expects layers to be softly merged. *Let me know if this is a big loss.*
- Transmission or **translucent lighting** exposes more parameters and works slightly differently.

XR

ATG supports *Single Pass* and *Single Pass Instanced* - however Unity's terrain only supports *Single Pass* (tested with Unity 2019.3.0b10, URP 7.1.2, Windows 10 and Oculus Rift). Apart from that: *Single Pass Instanced* seems to not work that well with Unity's *DrawMeshInstancedIndirect*: At least I could not spot any performance improvements using *Single Pass Instanced*.

When using *Compute* you should increase the *Instance Size* to get rid of flickering instances at the outer edges. A value of 7 gave me rather stable grass, a value of 12 also stable shadows. *Compute* will be disabled if *Single Pass Instanced* is enabled.

Shader Inputs

Grass Shader

AdvancedTerrainGrass URP/Grass

Surface Options

Alpha Clipping Check this if the shader shall do alpha testing which usually is the case.

Threshold this is formerly known as the cut *off* value.

Threshold Shadows Lets you define a different threshold for shadow casters to make grass less “solid” or “opaque” in case you use a slightly smaller value here .

Receive Shadows Lets you define if the object shall receive shadows.

Surface Inputs

Albedo (RGB) Alpha (A) The main texture with albedo in RGB and alpha in A.

Enable Texture Arrays Check this in case you need texture arrays (because you use soft merged layers)

Albedo (RGB) Alpha (A) Array The main texture array which is used if *Enable Texture Arrays* is checked.

Healthy Color (RGB) Bending (A) and **Dry Color (RGB) Bending (A)** Just like you may now from the built in grass shader the ATG grass shader supports tinting grass using two different colors which get applied according to the given noise specified in the layer settings of the Grass Manager script. While RGB of these colors will affect the tinting, alpha may reduce bending or wind strength.

Lighting

Use NormalBuffer If checked the shader will use the passed terrain normals to mimic the lighting you are used to getting from the built in grass. This is only needed and useful in case the according grass instances do NOT use *Rotation Mode: Aligned Random Y Axis* and you have checked *Write Normal Buffe*r in the layer settings of the Grass Manager script.

Bend Normal Lets you define how strong wind affects lighting.

Enable Mask Map If checked the shader will sample the:

Normal(GA) Trans (R) Smoothness (B) A combined texture which contains: the normal in the green and alpha channel, translucency in the red channel and smoothness in the blue channel.

Normal(GA) Trans (R) Smoothness (B) Array The combined texture array which is used if *Enable Texture Arrays* is checked.

Please note: If you uncheck *Enable Mask Map* make sure you lower the *smoothness* and *transmission strength* so your grass does not look like a self illuminated mirror thingy...

Smoothness Smoothness that gets multiplied on top of the sampled smoothness from texture input above. If *Enable Mask Map* make is unchecked this is the final smoothness.

Specular Specular color. As lighting in URP works slightly differently you may raise the specular color to get more specular reflections and highlights on the grass.

Occlusion Lets you dim ambient occlusion as derived from the vertex color.a (which defines main bending).

Transmission

Power Determines view dependency. Higher values will make transmission kick in only when the view ray more or less directly points towards the light source.

Strength Lets you scale transmission.

Shadow Strength As transmission might be totally eliminated by self shadowing this parameter lets you suppress shadows when it comes to transmission. Other lighting features (diffuse, specular) are not affected.

Distortion When calculating transmission the shader distorts the inverted light direction vector slightly by the given normal to simulate the scattering. Default value is 0.01. Higher values will give you more scattering and break up the uniform look.

The subsurface color will be derived from albedo.

Two Step Culling

Clip Threshold determines the amount of instances which will get culled early on by the built in two step culling.

Wind

Wind Strength is the final multiplier for the actual wind strength applied.

Jitter Strength is the final multiplier for the actual jitter applied.

Sample Size If set to 1 the wind texture will be sampled at the vertices world space position. If set to 0 all vertices of the given mesh will sample wind at the pivot. This parameter replaces the old *Sample Wind at Pivot* and is way more flexible.

LOD Level lets you specify the LOD level at which the global wind texture gets sampled. Higher LOD levels will create smoother bending.

Advanced

Enable Blinn Phong Lighting Lets you switch to a cheaper lighting model.

Enable Specular Highlights If unchecked the shader does not calculate direct specular highlights which makes it cheaper.

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

Foliage Shader

AdvancedTerrainGrass URP/Foliage

Surface Options

Alpha Clipping Check this if the shader shall do alpha testing which usually is the case.

Threshold this is formerly known as the cut *off* value.

Receive Shadows Lets you define if the object shall receive shadows.

Surface Inputs

Albedo (RGB) Alpha (A) The main texture with albedo in RGB and alpha in A.

Healthy Color (RGB) Bending (A) and **Dry Color (RGB) Bending (A)** Just like you may now from the built in grass shader the ATG grass shader supports tinting grass using two different colors which get applied according to the given noise specified in the layer settings of the Grass Manager script. While RGB of these colors will affect the tinting, alpha may reduce bending or wind strength.

Smoothness Smoothness that gets multiplied on top of the sampled smoothness from texture input.

Specular Specular color.

Enable Normal Smoothness Trans Map If checked the shader will sample the assigned:

Normal(GA) Trans (R) Smoothness (B) A combined texture which contains: the normal in the green and alpha channel, translucency in the red channel and smoothness in the blue channel.

Transmission

Power Determines view dependency. Higher values will make transmission kick in only when the view ray more or less directly points towards the light source.

Strength Lets you scale transmission.

Shadow Strength As transmission might be totally eliminated by self shadowing this parameter lets you suppress shadows when it comes to transmission. Other lighting features (diffuse, specular) are not affected.

Distortion When calculating transmission the shader distorts the inverted light direction vector slightly by the given normal to simulate the scattering. Default value is 0.01. Higher values will give you more scattering and break up the uniform look.

The subsurface color will be derived from albedo.

Wind

Wind Input *Math* uses the old method, *Texture* only uses input from the wind texture but samples it twice.

Primary Strength final multiplier for the actual wind strength applied to primary bending.

Secondary Strength the final multiplier for the actual wind strength applied to secondary bending.

Edge Flutter the final multiplier for the actual wind strength applied to edge fluttering.

LOD Level lets you specify the LOD level at which the global wind texture gets sampled. Higher LOD levels will create smoother bending.

Sample Size If set to 1 the wind texture will be sampled at the vertices world space position. If set to 0 all vertices of the given mesh will sample wind at the pivot. This parameter replaces the old *Sample Wind at Pivot* and is way more flexible.

Advanced

Enable Specular Highlights If unchecked the shader does not calculate direct specular highlights which makes it cheaper.

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

VertexLit Shader

AdvancedTerrainGrass URP/VertexLit

Surface Options

Receive Shadows Lets you define if the object shall receive shadows.

Surface Inputs

Albedo (RGB) Smoothness(A) The main texture with albedo in RGB and smoothness in A.

Healthy Color (RGB) and **Dry Color (RGB)** Just like you may know from the built in grass shader the shader supports tinting objects using two different colors which get applied according to the given noise specified in the layer settings of the Grass Manager script.

Smoothness Smoothness that gets multiplied on top of the sampled smoothness from texture input.

Specular Specular color.

Enable Normal Map If checked the shader will sample the assigned:

Normal Map a regular normal map.

Advanced

Enable Specular Highlights If unchecked the shader does not calculate direct specular highlights which makes it cheaper.

Environment Reflections If unchecked the shader will not sample any reflection probes which makes it cheaper.

Known Issues

Grass does not show up at all

This of course is the worst bug. But there are several ways you can try to make it show up.

- Check if your **camera** is tagged as MainCamera.
- Make sure that if you are in the editor there is no **scene view** tab opened when you are in play mode (e.g. simply click: Maximise On Play). Otherwise Unity might get confused about the main camera.
- Check **Ignore Visibility** in the Grass Manager's inspector. This will make culling spheres ignore the visibility test and should resolve problems which might be caused by occlusion culling or a second active camera.
- In case you have assigned a **Saved Terrain Data** simply rebuild it to make sure that it is up to date.
- Check **Debug Stats** and watch the Number of **Cells to init.** In case this number does not change at all, the worker thread most likely produced an error and you should contact me.
- Last but not least look for **errors** in the console.

Shadows

Shadows in Unity 5.6.1 and 5.6.2 might flicker – as some cells might randomly not be rendered properly to the shadow map. This issue is solved in Unity 5.6.3.

Cells are missing and/or pop in

If the camera moves or rotates quickly some needed cells might not be available because they have not been created yet. Please remember: ATG by default inits one cell per frame. So in case you have rather small cells but a huge cull distance, moving the camera will cause a huge number of new cells which have to be initialized.
Please consider raising the cell size or lowering the Cull Distance. You may also increase the number of **Cells per Frame**.

Editing

When deleting prototypes from the terrain, the settings in the Grass Manager will be corrupted. In this case you will have to go through all prototype settings and assign proper shaders and rotations… sorry about this.

Empty Prefab Previews

Preview thumbnails of the grass prototypes might be empty (in both the terrain engine as well as the grass manager editor). In order to fix this try to select the prefab in the project tab and right click “Reimport”.

Troubleshooting

Grass flickers when using compute

I haven’t been able to reconstruct this, but a user reported flickering grass when using *compute*. Lowering the *number of buckets per cell* fixed this problem (32x32 instead of 64x64).

Grass floats on the terrain in build

During the build process Unity optimizes the included meshes based on the related shaders and their usage/need for vertex colors and uvs. Unity does not recognize a mesh being used by ATG and its shaders so we have to somewhere declare the relationship between mesh and shader. Otherwise vertex colors most likely will be stripped.

Do so by e.g. creating a prefab and assigning a proper material (using an ATG shader).

Grass vanishes after enabling RenderDoc

Loading RenderDoc will wipe out the custom texture arrays?! So if you just enable RenderDoc and then enter play mode no grass at all may show up depending on if you use texture arrays or not.

Fix this by selecting the texture arrays in the project tab and *RMB → Reimport* them so you get a proper preview of the texture array’s layers.

Using VR half the grass is missing

It looks as if Unity will not return a proper result when using “UnityEngine.XR.XRSettings.stereoRenderingMode” in conjunction with MockHDM. So the inspector offers a checkbox: “VR: Force Single Pass Instanced” which you may check if you are using *Single Pass Instanced Rendering* and *MockHDM*.