Version 1.3.0

About

'Critias Foliage System' is an system that is designed to paint, render and extract millions of instances of foliage without taking into account all the foliage but only the one that is in the proximity of the player. The flow is quite close to Unreal's foliage painter. Add your foliage to the system, set it's runtime settings and start painting!

In one word it makes possible huge forests of more than 100k trees, with wind, colliders and LODs, plus millions of real mesh grass (no billboards) with wind, fading and density settings.

Notes and Limitations

The system requires gaming GPU's with true instancing support. The minimum settings are a GTX570. I've also tested on a GTX750TI Golden Sample and a GTX970 with good results.

If you system behaves bad on your machine or on the machines that you are testing on, I recommend to use Unity's build TreeCreator and foliage systems that are designed for very low-end machines!

With instancing the requirements are:

- Unity 2017.1+
- Unity 5.6.3f
- DX11
- DX11 equivalent graphic API's with instancing support
- GTX570/GTX750TI Golden Sample or equivalent GPU

Layout

The project contains the following layout:

- Assets
  - CritiasTreeSystem
    - Code
    - Examples
    - Shader

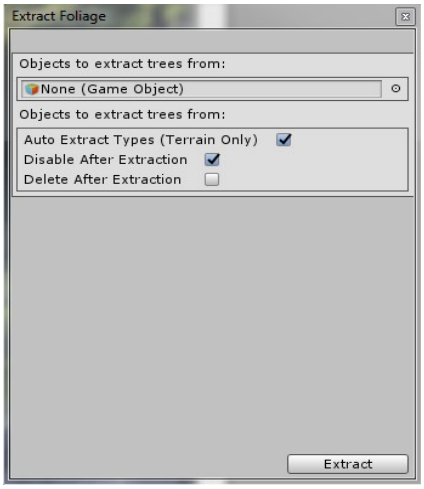
Installation

Add/copy the folder 'CritiasFoliageSystem' anywhere in your project where it suits your needs.

Quick Setup

IMPORTANT: Add a folder called 'StreamingAssets' (the default Unity streaming assets folder) into your project's root under 'Assets'. The system will save data there. After the project finished compiling go to 'Critias Foliage' on the menu bar and hit 'Create Painter'. That will create a new game object painter in the scene called 'Critias_Painter'. It will have three scripts attached, 'FoliagePainter' responsible for painting the foliage, 'FoliageRenderer' responsible for rendering the vegetation at runtime and 'FoliageColliders' that will add colliders to the foliage that has collisions.

1. **Extracting:** If you already have a terrain that contains trees you can extract them from the terrain and add them to the system. Select the painter ('Critias_Painter' game object) and go to the section 'Advanced' (we'll discuss about the others later) in the inspector and press the big button called 'Extractor' (right under 'Foliage Types' shown in the image). After you have pressed it you should see a setup dialog called 'Extract Foliage' (shown in the image). Add all your terrains (or other objects that have foliage as children) to the list of objects under 'Objects to extract trees from'. If 'Auto Extract Types' is on then the system will try to auto-configure itself and add all the tree types from the terrain to the system and extract all the tree instances after. If 'Disable After Extraction' is checked then all the objects will be disabled of if it's a terrain it will have it's flag 'drawTreesAndDetails' set to false. If 'Delete After Extraction' is enabled all the extracted terrain's tree instances or game objects will be deleted. Not recommended to be enabled unless you are very sure that your foliage is properly placed. After it is finished, hit 'F4' (Save) or go to Menu->Critias Foliage->Save and hit play. If you hit play without saving the grass will vanish.



2. **Painting:** Select the 'Critias_Painter' game object and add to it all the game object prefabs that you want to add to the terrain. The added prefabs must not be from the scene, they must be SpeedTree objects or prefabs. If they contain LODs with more than one LOD they will be considered trees and else they will be considered grass. Select the grass that you want to paint (using the small checkbox that is displayed at their upper-left corner) and paint. After you finished painting 'hit F4' (Save) or go to Menu->Critias Foliage->Save and hit play. If you hit play without saving the grass will vanish.

Advanced Setup

Important Notes:

1. Add a folder called 'StreamingAssets' (the default Unity streaming assets folder) into your project's root under 'Assets'. All the saves (you can save the foliage using 'F4') will be saved

there in a data file called 'FoliageData_+SceneName'.

2. Before hitting play, save your grass with 'F4' or the grass will disappear. Saving the grass will write all the grass hierarchy to the disk.
3. Create your foliage painter object with 'Critias Foliage->Create Painter' on the menu bar. It will create an object with three scripts attached, that is required for painting foliage in your scene.

Foliage Painter Sections:

The foliage painter script ('Foliage Painter') is divided into multiple sections. Here we'll talk about each section.

Help:

The help section is quite easy to understand, it provides some basic data about the system's functionality. You can paint grass using the left mouse button, you can erase grass with the right mouse button with a combination of shift to delete all foliage and ctrl to delete only the selected foliage types, 'F4' to save the grass. Sorry guys, I've tried to find a hook for detecting CTRL+S and save the foliage with that, but I couldn't find any information about that, so I've decided to set a new shortcut for the grass save. If you don't save the grass before hitting play it will all disappear since it is stored in memory before writing it to disk.

Brush:

This section contains the hand-painting parameters. It contains:

- *Brush Radius*: Brush radius in meters.
- *Foliage Density*: The painted foliage density in range (0.1-200). At 1 density we will have 1 instance of grass per 50 square meters and 1 instance of a tree per 2000 square meters.
- *Surface Align*: NOTE: This value has been moved in the info of each of the foliage type. It is now set per type rather than globally. This value is also used if you extract details data from the terrain. If we should align the painted object to the painted collider's surface, and how much we want to align to that surface. A value of 0% will not align the painter instance at all to the surface, a value of 50% will align it only half way to the surface and a value of 100% will align it completely to the surface. This value is only used for grass and non-billboard tree objects. All billboards will be perfectly aligned up.
- *Slope Filter*: A range between 0° and 180°. At a range of 45°-60° we will only paint on surface's whose slope is between 45° and 60°. If you want to paint vines or other details on a sealing only make sure that you set a range of 160°-180°. 0° paints on surfaces that are straight up and 180° on surfaces that are looking straight down.
- *Scale Uniform XYZ*: If painted objects will have a uniform scale or if you wish to scale each axis separately.
- *Rotate Y Only:* If we want to rotate only around the Y axis or if we want to apply a random rotation around all axes. Leave it on for trees and grass and turn it off for pebbles or rocks.
- *Y Offset:* NOTE: This value has been moved in the info of each of the foliage type. It is now set per type rather than globally. This value is also used if you extract details data from the terrain. How much we want to push in or out of the painted surface the foliage instance. Very useful when painting trees on slopes and we want to make the tree stick into the ground instead of sitting on it.
- *Paint Static Only:* If we are going to paint on static game objects only or if we are going to

paint foliage on dynamic meshes too.

Foliage Types:

This is the most important section and we are going to dedicate a little more information here especially about the foliage type and what kinds of foliage types the system can support.

You can add a new foliage type by grabbing the prefab/SpeedTree from the project explorer and dropping it to the location called 'Drop foliage prefabs here!'.

The system supports 5 foliage types:

- SpeedTree Tree (SPEED_TREE_TREES)
- SpeedTree Billboard Tree (SPEED_TREE_TREE_BILLBOARD)
- SpeedTree Grass (SPEED_TREE_GRASS)
- Other Tree (OTHER_TREE)
- Other Grass (OTHER_GRASS)

When adding a new prefab to the system it will try and configure it automatically. If the added prefab has a path that ends with a '.spm' it will be automatically configured as a SpeedTree. If it has 1 LOD it is detected as a SpeedTree grass type, if it has more than 1 LOD it will be detected as either a SpeedTree tree with billboard if it has a 'Billboard Renderer' or a SpeedTree without billboard if it doesn't have a 'Billboard Renderer' component. If the prefab does not end with '.spm' it will be detected as type other tree if it has more than 1 LOD and other grass if it has one LOD or no LODs at all.

It is important to understand the differences between the types, and we'll make a few notes of what each type supports:

- *SpeedTree Tree:* It is of a tree type and it supports multiple LODs, meshes with multiple materials, SpeedTree wind and SpeedTree style cross-fading. It's maximum draw distance is 1000m by default.
- *SpeedTree Billboard Tree:* It is of a tree type and it supports multiple LODs, meshes with multiple materials, SpeedTree wind, SpeedTree style cross fading and billboards will be generated for that type when painting or generating that type. It's maximum draw distance is 1000m by default.
- *SpeedTree Grass:* It is of a grass type and supports a single LOD, a single mesh with only one materials and SpeedTree wind. It's maximum draw distance is 100m by default.
- *Other Tree:* It is of a tree type and supports multiple LODs, meshes with multiple materials. It's maximum draw distance is 1000m by default.
- *Other Grass:* If is of a grass type and supports a single LOD and a single mesh with only one material. It's maximum draw distance is 100m by default.

If the detected type is incorrect you can modify it manually later. It is very important to get the type right. The auto-detector should be enough (at least for SpeedTree), but there are cases when it can't detect perfectly and you have to make sure it's the correct type.

There is a difference between grass and trees because the trees are drawn with less optimizations since the instance count is a lot lower and the grass is drawn more optimally since the instance count is probably tens of thousands larger than the tree instance count. The trees are tested per-instance for frustum culling while the grass is tested per-cell (batch) for culling. If you change the type there will be some checks to see if the conversion is possible, and the whole hierarchy will be re-built to

accommodate the new settings.

Options:

Refresh Types: Use this to refresh the types if you modified any properties like the LOD distances directly from the prefab or if you changed a texture to a prefab and it doesn't show in the system.

Clean Data: Use this to clean foliage data that contains any hashes that do not exist any more in the system. Can be used when you deleted a prefab without manually removing it from the system or when an 'Key Not Found' exception might be thrown.

Generate Billboards: Use this if there are any billboards where they shouldn't be. It will re-generate the whole billboard hierarchy.

Bake Collision: Use this to bake out the collision of the trees. Used when you don't need dynamic runtime collision generation. Recommended for less than 10000 trees.

Bake Navmesh: Bake out the renderers that can affect the navigation mesh from a tree. They will be generated only for trees that have collisions. An 'CRITIAS_Holder_RENDERERS_FOR_COLLIDER_MESHES_NAVMESH' object will be generated that can be deleted after generating the navmesh.

Bake All: Extract the desired instanced from the system and adds them to the standard Unity hierarchy. Not recommended for a large (> 1000) amount of objects.

How do LODS work: The LOD distances are computed based on the values set in the prefab. Their end distance will be calculated with: '((1.0f - lodGroupCurrent.screenRelativeTransitionHeight) * maxDistance)'. It is based on the existing LOD settings in the prefab and the type is chosen in real-time inside the system.

Foliage Type Info:

There are some settings that will be available per foliage type. Just select the foliage type in the inspector and expand the 'Foliage Type Info' drop-down.

There you can change:

- Foliage Type (check about the types in the previous section).
- Max Draw Distance: 0-100m for grass, 0-1000m for trees. The default is 100m for trees and 30m for grass.
- Enable Shadow: If the type should have shadows rendered. Not recommended for grass. Use very sparingly.
- Enable Collision: If the type should have collision enabled. Ignored for grass, used only for trees.
- Render Type: Only used for grass types. The render types are: INSTANCED and INSTANCED_INDIRECT. INSTANCED will use the 'DrawMeshInstanced' API and INSTANCED_INDIRECT will use the 'DrawMeshInstancedIndirect' API. The second API is recommended for the grass types (even though it will not render shadows) since it will not copy the position data to the GPU each frame, it will smartly cache it based on the already visible

cells. The default 'WindTree_Grass' shader is compatible with that draw mode.

- Enable Bend: Enables bend on grass that have a shader the support it. The default 'WindTree_Grass' supports it.
- Bend Distance: At what distance from the watched position transform we should start to bend the foliage.
- Bend Power: How much we are going to bend the grass.
- Hue Color: Used for SpeedTrees, can change the hue color at runtime.
- Main Color: Used for SpeedTrees, can change the main color at runtime.
- Surface Align: If we are going to align the painted grass to the underlying surface. Set per type and not globally, so that we can paint both grass that might be aligned and trees too.

Advanced:

In the advanced section you can set some advanced settings for the system and you have some advanced features.

1. Extractor: The extractor will add instances to your system. It will extract all tree instances from a terrain if the passed object is a terrain and all objects owned by a root game object if the passed object is a Game Object. Extracting from a terrain allows you the option to automatically generate the foliage types while if we extract from an object the foliage types must already be added to the system and only the objects that are added to the system will be extracted from root Game Objects. You also have the option to disable/delete the extracted objects.
2. Extractor Terrain Details: The extractor will get the detail instances from the terrain and add it to the system. You can map existing terrain details to foliage types that are already added to the system. The extractor will attempt to map the types automatically but it is not foolproof anyway. Manual tweaking might be required. You can also tweak the extraction density per type and can extract from multiple terrains at once.
3. Billboards: The billboard section will set some settings per billboard batch. Check 'Generate LOD Group' if you want an LOD group attached to the billboard batch, set the 'Cull Screen Size' to the screen fade size you wish and the 'Animated CrossFade' if you want them to fade nicely when they are at a great distance. After you made modifications to this sections don't forget to regenerate the billboards with 'Generate Billboards'!
4. Editor Rendering Data: All this data is related to editor rendering and not the RUNTIME!
    1. *Grid Draw Mode:* Draws various grids for visualization. 'DRAW_GRIDS' 'DRAW_SUBDIVIDED_GRIDS' will draw the edit time cells that are not empty and 'DRAW_DRAWN_*' will draw all the grids that are currently drawn in edit mode.
    2. *Draw Neighboring Cells:* How many cells around the player are drawn. On 1 only the 9 cells are drawn around the player, at 2 there are drawn 25 cells around the player, for 3 about 49 cells are drawn for the largest view distance. Keep at 1 for best performance.
    3. *Draw Grass Cell Distance:* The distance at which we are going to draw grass. Keep under 50 for best editor performance.
    4. *Draw Tree Shadows:* If we should draw shadows for the trees in the editor. Turn off for the best editor performance.
    5. *Draw Tree Last LOD:* If we should draw the trees with the first or the last LOD. Recommended to be on for best performance.
5. Set Foliage Data Name: Allows you to change the default name of the data file that contains all the data.

Usage

The usage is quite simple.

1. Follow the 'Setup' described above.
2. Add the required foliage prefabs to the system.
3. Paint your foliage in your world with your favorite painting/placing method.
4. Press 'F4' to save and hit 'Play' in order to see your awesome creation at awesome FPS!
5. Go to step '3' again.

Shaders

The system comes with it's own set of built in shaders. They are responsible for multiple tasks and they can be used with your own prefabs.

There are 4 included shaders:
1. WindTree_Master
2. WindTree_Grass
3. DetailDithered
4. DetailDitheredMetallic
5. NullShader

and an additional 'CritiasUtils.cginc' utility include.

WARNING: All the 'WindTree' shaders are designed for SpeedTree trees and should not be used for other tree types! In the future we will have a default grass shader for non-SpeedTree grass but it is not the case at the moment!

The 'WindTree_Master' will be set automatically to SpeedTree tree types and it adds to them GPU computed fading, GPU computed LOD cross-fading.

The 'WindTree_Grass' will be set automatically to SpeedTree grass types and it adds to them GPU computed Y scaling based on distance.

WARNING: If you have issues and have errors with the 'WindTree_Grass' try and re-import it and add it to the 'Always Included Shaders' list.

The 'WindTree_Billboards' will be used for SpeedTree batched billboards. It has 2 possible drawing methods for the shadows. The first uses the 'addshadow' directive and will draw shadows that rotate with the billboard. If you do not use the 'addshadow' directive it will draw shadows that behave exactly like the SpeedTree billboards, however it will have that specific shadow artifact on the billboard.

The 'NullShader' is a shader that doesn't draw anything but is used to make certain objects invisible, like the SpeedTree objects attached to the player's camera from which the wind parameters are copied through a MaterialPropertyBlock. Very important if we want the objects that copy the wind to be invisible and not have a lot of trees around the player's head.

The 'DetailDithered' shader should be used by you and set to your terrain details. It is a 'PBR' shader

that has a BaseColor_Rough map and a normal map and computes a dithered fade per instance very fast directly GPU.

The 'DetailDitheredMetallic' is the same thing but designed to work for metallic items.

The 'CritiasUtils' provides some utilities for computing dithering or fading per instance.

Writing Custom Shaders:

If you want to write your own shaders for the system or adapt existing ones there are some parameters or utilities that will facilitate that. For each foliage type batch sent to the GPU there will be some additional data set to each shaders.

Those are:

1. 'CRITIAS_MaxFoliageTypeDistance' – Sent for each batch. It represents the maximum distance that the foliage type will be drawn at. You can calculate the distance between the camera and the instance with the following code snippet:

```
{
	float3 campos = _WorldSpaceCameraPos;
	float3 pos = float3(unity_ObjectToWorld[0].w, unity_ObjectToWorld[1].w,
				unity_ObjectToWorld[2].w);

	float dist = distance(pos, campos);
}
```

You can use that distance to fade your instance (there you go, poetry). Check the 'DetailDithered' and 'WindTree_Grass' shaders for examples. It is very likely that this value is the most important for your custom shaders.

2. 'CRITIAS_FoliageMaxDistanceLOD' is sent for objects that have LODs and are used by the SpeedTree tree shader to calculate the current lod cross-fading value. It is highly unlikely that you will benefit much from using that value.

3. 'CRITIAS_InstancePositionBuffer' is sent for shaders and instances that have the 'INSTANCED_INDIRECT' flag set. It is compatible with grass foliage types only. Check the default 'WindTree_Grass' to see how it is used

4. 'CRITIAS_Bend' is sent for shaders and instances that have the 'Enable Bend' flag set. Check the default 'WindTree_Grass' to see how it is used.

I highly encourage you to use SpeedTree for your foliage's needs. It comes with wind, optimizations my special provided shaders etc... that will make your life a lot easier when developing foliage.

I also hope that in the future based on your updates and requests, this list of included shaders might grow or at least include snippets of code for popular vegetation shaders already existent on the Asset Store.

Compatible shaders:
Advanced Foliage Shader V5 – Thanks to Lars!

RUNTIME

There are some settings at runtime that you should know about. Except the settings that you are going to modify through the painter runtime API, you can also set some other settings through the 'Foliage Renderer' and 'Foliage Colliders' script.

The 'Foliage Renderer' will have the following properties:

- Draw Instanced: If we should draw everything instanced. If disabled no grass will be drawn.
- Use Light Probes: Only used if we don't use instancing, and will be used only for trees.
- Allow Draw Instanced Indirect: If we allow the usage of 'DrawMeshInstancedIndirect'. Usually there would be no reason to turn this value off.
- Grass Density: Global grass density that will scale the count of grass instances drawn.
- Wind Transform: To what transform we will attach the dummy wind objects. Defaults to 'Camera.main.transform'.
- Bend Transform: If we have foliage that has bending enabled and has a shader that supports bending, what position will be sent to the shader in order to bend the grass based on distance. Should be set to a dummy object that is placed at the player's feet. Default to 'Camera.main.transform' if not set. Must be set if you want correct results using grass bend!
- Used Camera Culling: What camera we are going to use for frustum culling. Default to 'Camera.main'.
- Used Camera Drawing: What camera we are going to use for drawing. Defaults to 'null', that is will draw the foliage for all the cameras in the scene.
- Used Layer: Layer used for rendering. Default to 'Default'.
- Apply Shadow Popping Correction: Since frustum culling only does not take in account shadow casters then we are force drawing trees as shadows only if we have this option on.
- Shadow Popping Correction Distance: At what distance we are going to draw the trees around the player

Note that the last 2 options are a work-around until a smarter shadow-casting culling algorithm can be implemented.

The 'Foliage Colliders' will have the following properties:

- Watched Transform: What transform we will watch for refresh.
- Collision Distance: At what distance around the player we'll add colliders.
- Collision Refresh Distance: Apply a collider refresh only after the 'Watched Transform' moved this distance.
- Used Layer: To what layer we'll add the colliders. Defaults to 'Default'.

API

If you want to use the system either at runtime or at edit-time make sure that you do not use the 'FoliagePainter' script directly! If you do, it will be at your own responsibility and it is not supported.

If you want to do modifications request from the properties of the script an 'FoliagePainterRuntime' with 'GetRuntime' for runtime operations and an 'FoliagePainterEditTime' with 'GetEditTime' for edit-time operations.

Those operations are supported and perform various tasks on the system. I hope that in the future the list of supported operations can grow. At the moment runtime tree removing from the system is supported and based on the requested features, more can be added in the future.

There are certain constants global constants, defined in 'FoliageGlobals' (defined in 'Foliage.cs') that define the behavior of the system. The default cell size is 100m with 5 subdivisions. That means that the trees are going to be put in 100m cells and the grass in 20m(100m/5 subdivs) cells. There are some considerations for those values. If you will have larger tree cells less billboard batches will be generated, however the tree culling will take somewhat more time. Note that each billboard batch is a GameObject that takes extra time on the CPU.

For example if you have a 1x1km terrain with a 100m cells and it is full of trees that have billboards then the system will generate 1000m/100m^2 that is 100 batches. For a 4x4km terrain that is 4000m/100^2 that is 1600 batches.

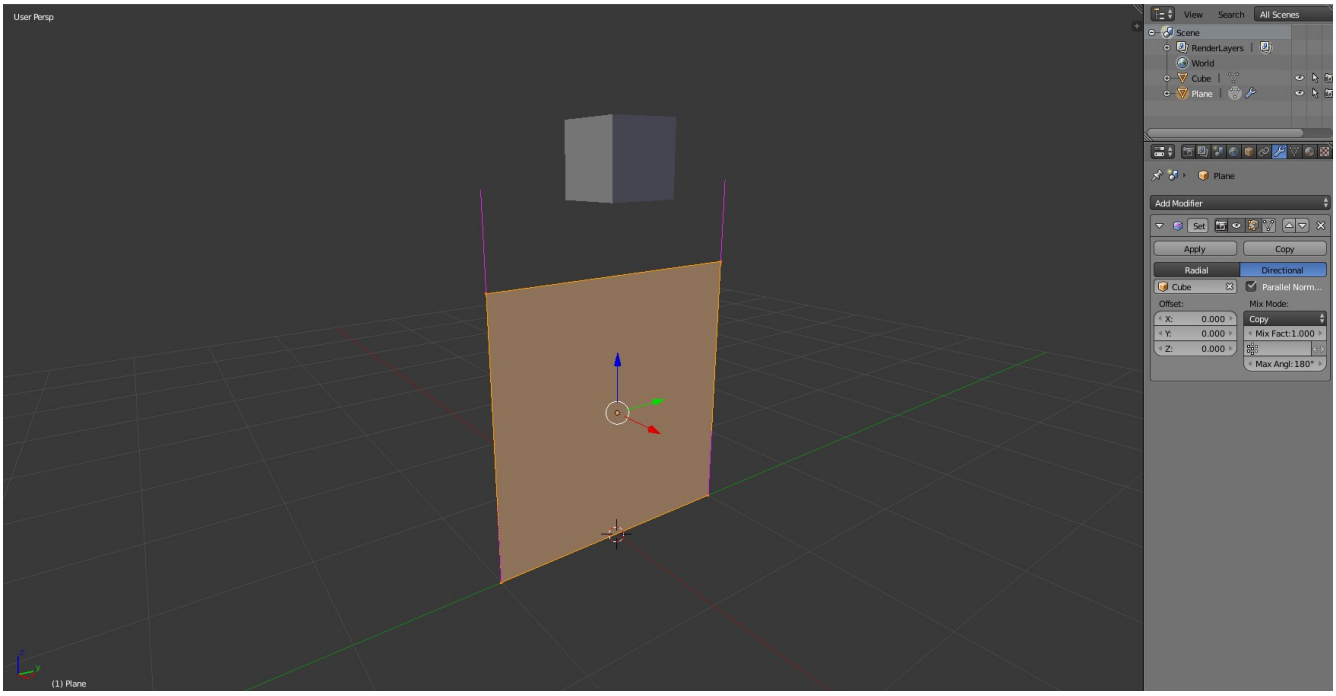
The default 100m value with 5 subdivisions will work just fine for terrain that are from 1x1 to 4x4km in size, while a value of 200m with 10 subdivisions should be fine for terrain 5x5 to 8x8km in size. Just crunch the numbers and try to obtain a count of batches that is as close to something like 1000 as possible.

As a general note, bigger cells mean less work on the CPU and more work on the GPU. I'll leave it to you to decide whether your bottleneck is the GPU or the CPU.

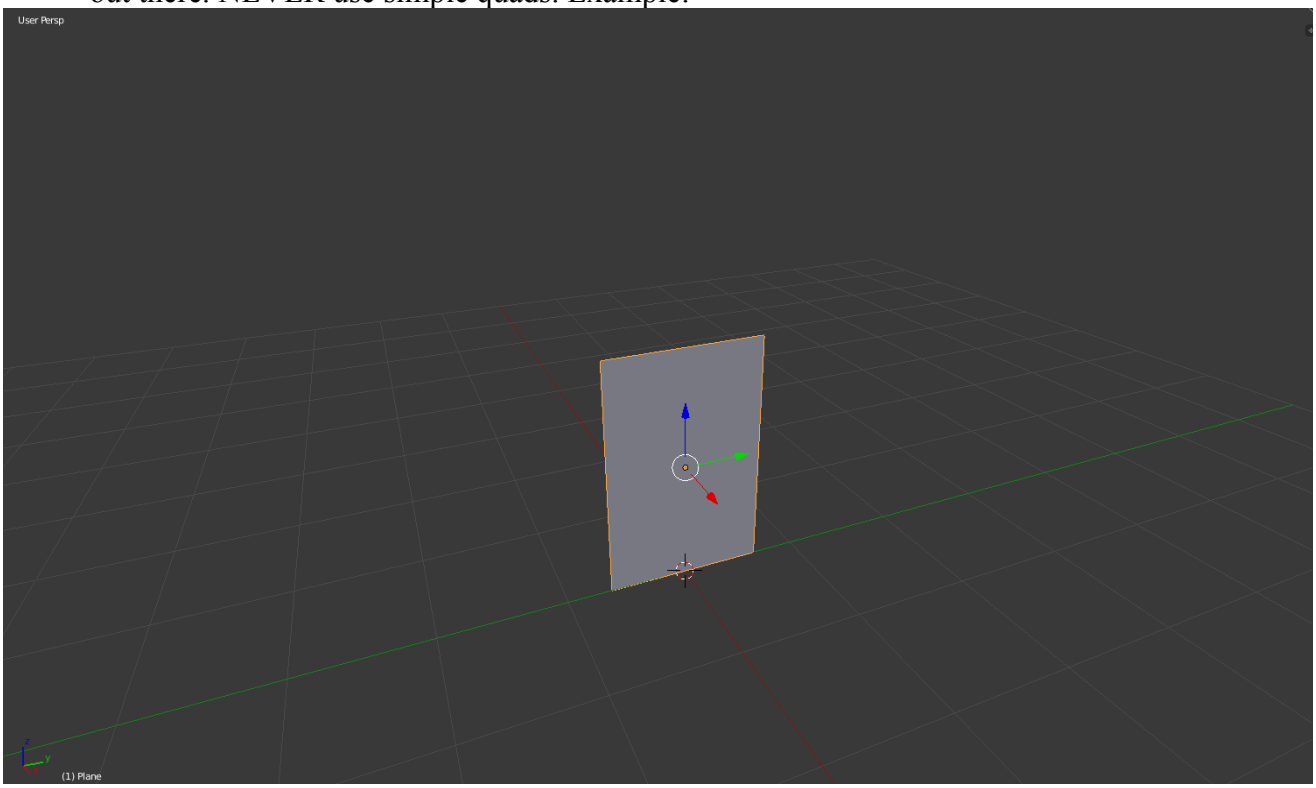
NOTE: If you change those values all the data that you have generated will be invalidated and you will have to start from scratch. So be careful when changing those values, and do it at the beginning of the project based on your world's size.

Common Pitfalls (Please Read)

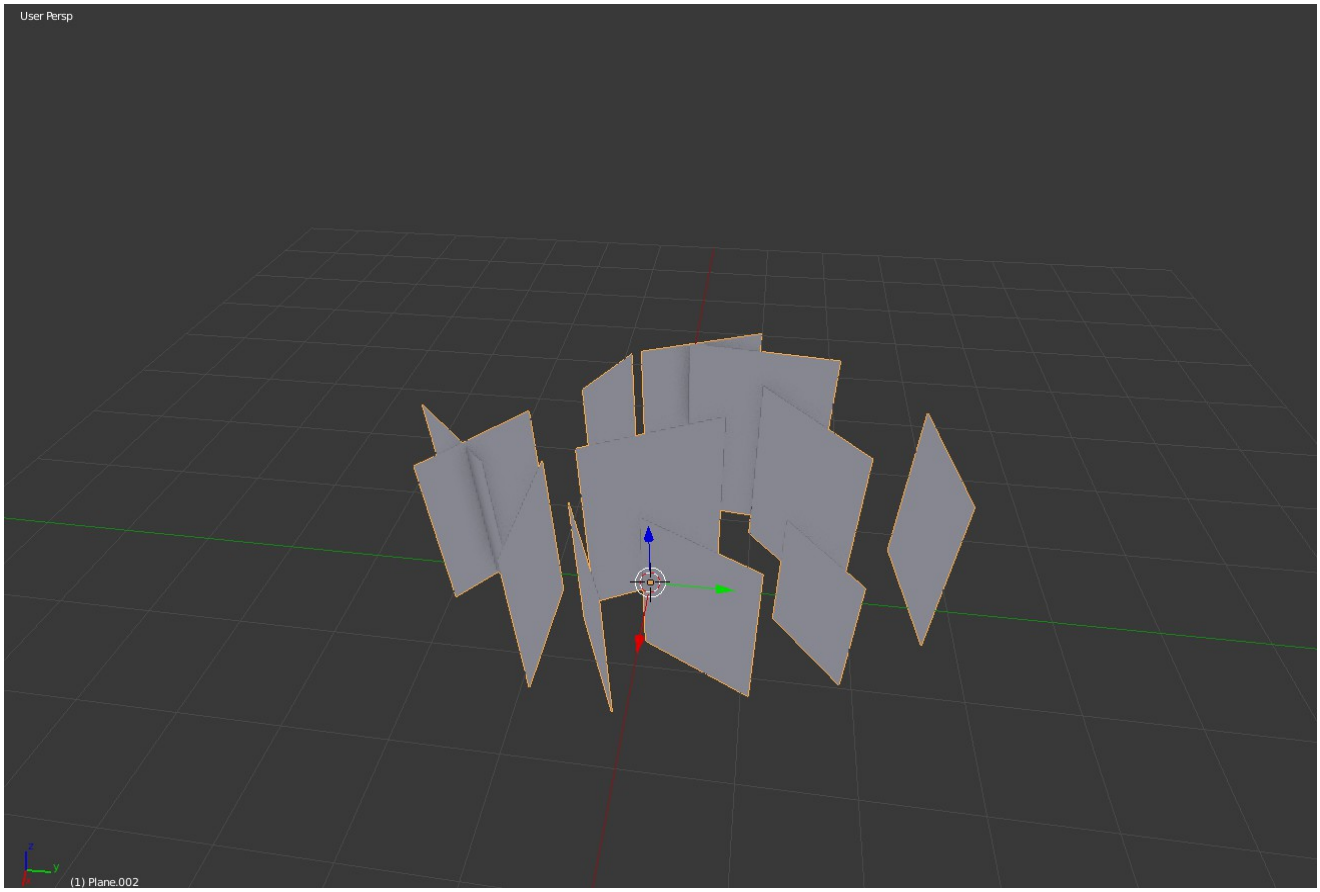
1. **Grass looking weird at different rotation angles**: The issue is solved by making the normals of the foliage face directly up, rather than the default front. An example of doing that using Blender would be to use the 'Normal Edit' modifier with a dummy cube right above the object, by setting the mode to 'Directional' and checking the 'Parallel Normals' option and selecting that cube. Now at all rotations the grass will look just fine. Example:



2. **Using quad meshes**: NEVER use simple quad meshes for grass. It will lead to the requirement of a huge amount of meshes to fill a rather small area of terrain and to an unnecessary high density requirement. It will, for sure, blow up to the system. And the problem is not in my system only, it is an universal problem, it appears in Unreal, CryEngine and all other engines out there. NEVER use simple quads. Example:



(BAD, really really BAD grass mesh.)



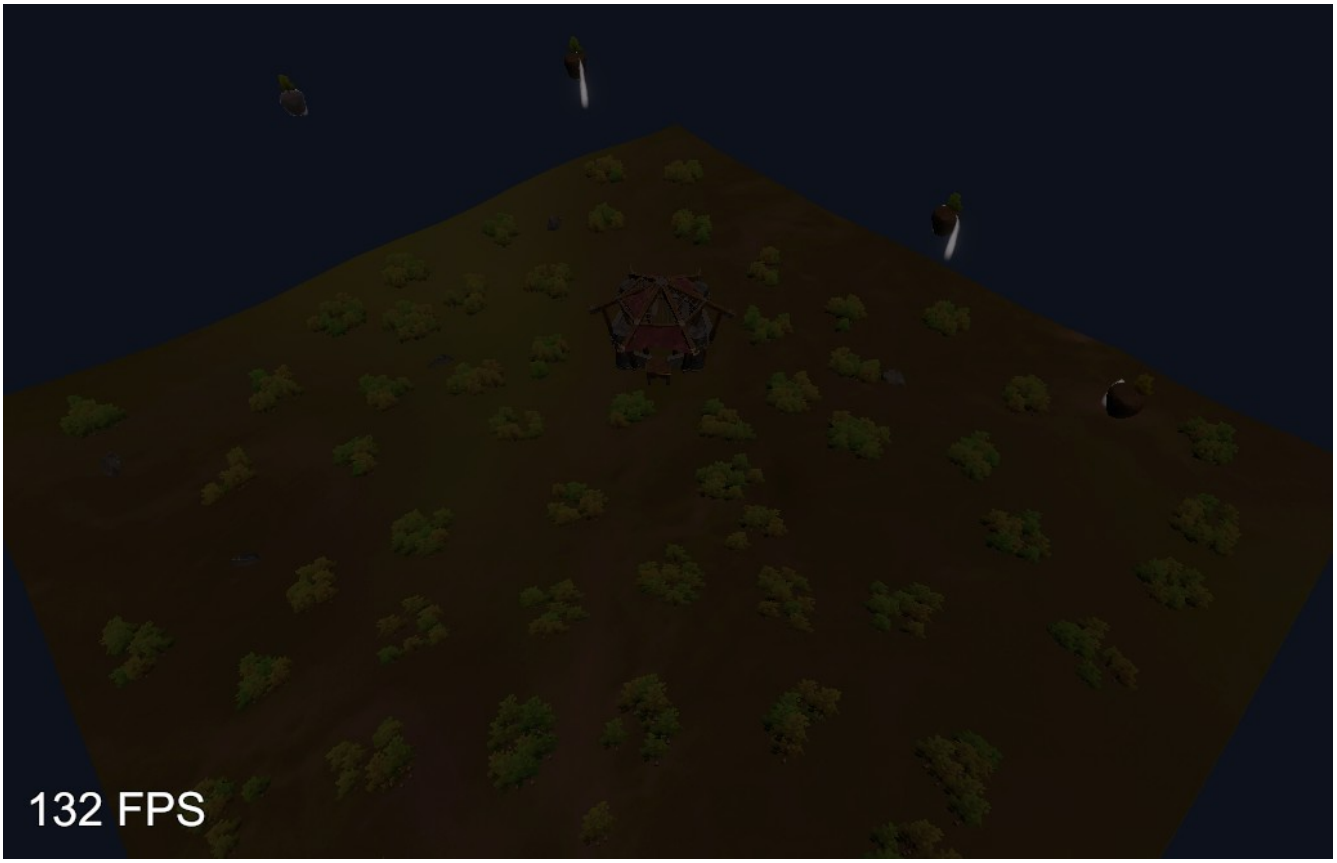
(Good mesh. No texture mapping for brevity. A single clump and there are many grasses.)

3. **Billboards looking bad at sunset/sunrise lighting**: It is very often that you might set up your scene for a sunset and the billboards look to bright an weird. The problem appears using the standard SpeedTree billboards too. This is what you might encounter:

The billboards look too bright, even if we are at a sunset scene. Anyway, they look very wrong. The fix does not come from any tampering with the system but by discovering the artist in yourself. To get a better look you can do the following:

- Tweak down the main light intensity and color when approaching the sunset/sunrise.
- Change the skybox's settings. I'd recommend using a custom skybox since the default Unity one is not great. It will influence the ambient spherical harmonic and the reflections.
- Change the ambient light intensity if it comes from the skybox or turn it to color mode and make the color less intense. On the skybox mode, an ambient value of 0.3 will usually do.
- Set up reflection probes around! Make sure that they are refreshed on time of day changes and their intensity is lowered with some time of day script!

After everything has been applied you should get the following effect:



Everything is evenly lit, the billboards don't look to bright any more. This is what you usually want. Not the best artistic example, but for any of your scene this is what you should do.

Tips & Tricks

How to bake realtime shadows on a terrain: (Special thanks to - Migueljb)
1. Use any tool to generate all the Speed trees you want and how you want them placed inside the terrain engine. Bake the lighting using Mixed Lighting and Shadow mask checked.
2. Load up Critias and extract the trees from the terrain. Extracting the terrain trees will hide the current ones that are inside the terrain engine so it acts like the new generated Critias trees are the ones that created the baked shadows. Select each tree type and check Draw Tree shadows.
3. Make sure your directional light is set to mixed mode with shadows.
4. In your quality settings make sure your Shadowmask Mode is set to Distance Shadowmask then set your shadow distance to where and how far you want the baked to realtime shadow to show up. Typically a good number is anywhere from 50 to 150 depending on how much performance you have and want to save etc.
5. Hit play and you got realtime shadows in your speed trees with baked shadows in the background so its a seamless transition looking good for your landscapes.

Techy

For those interested in the architecture of the system, this section will provide a little insight on the

overall idea. It begun as the old 'Critias Tree System' and after that the 'Critias Grass System', based on a grid that envelops a terrain. However that approach was not ideal for anything else that was not based on a terrain and the setup/extracting was not very intuitive.

This system uses a spatial hashing method that will let you directly paint on any surface (or extract existing foliage from any surface) and will not make you dependent on any terrain or other object. When an instance is added to the system a hash that is mapped to a cell is calculated based on the position of the foliage. If the cell already exists the foliage is added to that cell, else the cell is created and after that the instance is added to that cell. In that way we have a sparse array of spatially mapped cells. If a section of space will not have any foliage instances then no cell is mapped to that location, therefore that location is quickly skipped.