Custom Tree Importer 3.1

Hi there and welcome to the documentation of the Custom Tree Importer.

Benefits

The custom tree importer lets you import manually modelled trees which get adjusted on the fly to be compatible with Unity's tree creator shaders and wind zones. You may even create smoothly fading LOD groups using CTI's custom LOD shaders which support physically based lighting.

Requirements

Please do not expect the custom tree importer to simply make any tree just work: You will have to spend some time tweaking your model as the custom tree importer needs trees to be imported as nested, hierarchical objects.

You should be quite familiar with your 3d app and do have a basic understanding of the tree creator and things like "main wind", "main turbulence" and "edge turbulence".

About this documentation

This documentation will give you a brief overview of the differences between the two CTI shader families, their features and requirements.

It describes how you have to set up your manually modelled trees in your 3D app to get them imported properly. In case you are new to CTI, i recommend to start with the Quickstart Guide. Otherwise you may directly jump to the In depth Guide, which covers all requirements and describes CTI's control tags one by one.

Finally it covers the setup of LOD groups and creation of custom billboards and describes pretty much any step you have to take to get your trees up and running – including the usage of all advanced features.

In case your have updated from an earlier version please have a look at the change log to find out about all the changes and improvements.

Table of Content

Before you start

- Before starting please make sure that your project is set to use the **linear color space** in *Edit -> Project Settings -> Player.*
- In case your camera uses **deferred rendering** you also have to assign the **CTI deferred lighting shaders** in *Edit -> Project Settings -> Graphics*:
  - Under the *Built-in shader settings* change *Deferred* to *custom*, then assign the *CTI_Internal-DeferredShading* shader.
  - Also change *Deferred Reflections* to *Custom* and assign the *CTI_Internal-DeferredReflections* shader.
- In case you use Unity >= 5.6.0 you will have to enable instancing on each material manually.

Please note: As the project contains shaders which support Unity 5.5 as well as Unity 2017.2 you may get a lot of shader warnings – just ignore them :)

Introduction

Why do usual custom made trees not work using the built in tree creator or speed tree shaders? Why do they go black or completely get distorted? The answer to all this is bending.

Tree Bending

Bending is calculated within the vertex shader – using the inputs from the the built in wind zone(s) AND the inputs from the trees themselves, which tell the shaders which parts of the tree are affected by the wind and how.

The built in as well as the advanced tree creator shaders support complex bending which consists of 3 blended animations:

1. **Main wind**, which animates the entire model along the wind direction.
2. **Main turbulence**, adding a higher frequent animation along the y-axis.
3. **Edge turbulence**, which is added to leaf planes and animated the vertices along the given normal at an even higher frequency.
   In order to make the whole bending more believable, there even is a fourth factor:
4. **Per-leaf / per-branch phase variation** which let you make all branches swaying at slightly different frequencies.

The tree creator shaders expect main wind and main turbulence values for each tree/branch/leave plane to be stored in UV2 whereas edge turbulence and phase variation must be stored in vertex color green and red.

Usually all these parameters are missing on manually modeled and imported trees. And here the custom tree importer script jumps in:

- It will create UV2 according to the setup of your model.
- It will apply vertex color red which controls per-leaf / per-branch phase variation in case you name all parts of your tree according to the naming conventions (see: 3.4. for details)
- However vertex color green which controls edge turbulence should be set manually. Simply add some green (values between 0.05 and 0.2 should be fine) to the outer vertices of your leaf planes. The custom tree importer script will read your values and pass them to the final mesh.
- In case you have chosen to bake leaf tumbling and/or use LOD Groups the script will also add UV3 (Please have a look at section 4. Leaf Tumbling to find out more).

Next to the rather common wind animation described above the custom tree importer also supports some more advanced techniques like **leaf bending**, **leaf tumbling** and on top of that **leaf turbulence**.

While **leaf bending** more or less is for free **leaf tumbling** and **leave turbulence** need additional information to be baked into the mesh — which will increase the needed data per vertex (+32 or + 48 bit) — and some more instructions in the vertex shader.

CTI tree shaders

Since Version 3 the package supports classical tree creator like trees as well as Speedtree like trees rendered using LOD Groups which means, that you can import different LOD versions of your tree, group them under a LOD Group Component and even add a custom billboard.

LOD trees can be placed using the terrain engine or as game objects and support billboarding in both cases. These trees need the LOD Tree Shaders to be assigned which work slightly differently compared to the built in or advanced tree creator shaders.

While tree creator like trees usually render very fast, LOD trees suffer from the same issue SpeedTrees do: In case you have a lot of them the performance may drop unless you use other 3rd party solutions like Critias Foliage or Vegetation Studio.

The rendering of tree creator like trees may be improved using the Advanced Foliage Shader v5 — especially as far as the transition between mesh tree and billboard is concerned.

Depending on your use case you may either use the tree creator like shaders or the LOD shaders. But please make your decision early on as both shader families need slightly different inputs as far as your meshes and textures are concerned.

The following comparison might help.

	Tree Creator Shaders	LOD Tree Shaders
Shader Model	2.0	3.0

Lighting Bark	Built in BlinnPhong (not physically based) *Might be deferred and pbs when using the AFS tree creator shaders.*	Built in Standard Specular (physically based).
Lighting Leaves	Built in translucent lighting (not physically based, always forward) *Might be deferred and pbs when using the AFS tree creator shaders.*	Custom translucent lighting (physically based, forward and deferred. Deferred translucent lighting needs a tweaked "Internal-DeferredShading" and "Internal-DeferredReflections" shader both provided with the package).
Bending	Advanced bending including leaf tumbling and leaf turbulence as well as the support for directional and radial wind zones.	Advanced bending including leaf tumbling and leaf turbulence as well as support for directional and radial wind zones when placed as game object. Trees placed using the terrain engine only support custom directional wind.
Billboards	Only supported for trees placed using the terrain engine. Billboards will be Unity's standard tree billboards: pre lit, full 360° view angles. Real time shadows are not supported.	Supported for trees placed using the terrain engine as well as for manually placed trees. Billboards will be lit in real time, may cast and receive real time shadows but do not support full 360° view angles as they only consist of 8 views (all from side, none from above).
Geometry	Leaves need double sided geometry – at least if you want to get proper billboards.	Leaves may use double or single sided geometry. Latter will save a lot of vertices which will be good for CPU, memory, bandwidth and the GPU. You can not mix double and single sided leaves within a single tree.
Multiple bark materials	Not supported. You would have to create a texture atlas.	Up to 2 bark materials are supported when using the LOD Bark Array shader. You may also mix details and global maps. Find out more >
Tessellation	Not supported	Supported by the bark shader (beta).

You can simply switch between the assigned shaders (in case you use double sided geometry of course) to make your tree either use the advanced tree creator shaders or the LOD tree shaders. **But please note:** LOD tree shaders need slightly different texture inputs.

Requirements

Geometry

In order to be able to calculate main wind and main turbulence for the trunk, the branches and leaf planes the importer script has to know about the different parts of the tree and their origins or pivots. For this reason you will have to import trees as **nested, hierarchical objects**. ***Tip:*** *Just have a look at the included demo trees to find out more.*

Flattened meshes will be processed far less accurate (if at all) and do not allow you to control all features supported by the importer and the tree creator shaders.

The **pivot of each element** of the tree must match the position where it is connected to its parent as it is taken as origin for any calculation: Setting up your geometry this way you will allow the script to calculate wind and turbulence relatively to the distance to its origin.

Please note: Some 3D apps need that you freeze all transformations before exporting your tree as .fbx (like modo 7 does), others do not (maya 2013).

Smoothing Groups

In order to get smooth shaded trunks and branches the smoothing angle has to be set according to your model's geometry. Do this either in your 3D app or in the unity inspector like you would do for any other model.

Please note: More important than on trunks and branches the smoothing angle will get when it comes to leaf planes.

As edge turbulence or edge fluttering is calculated using the vertices' normals, you have to make sure, that there are no hard edges on your leaf plane. Otherwise bending might break the geometry.

Further requirements

The meshes for **leaf planes** have to be double sided in case you want to use the built in or advanced tree creator shaders. Single sided geometry does not work with the tree creator shaders due to its lighting and shading functions and the way billboard textures are calculated internally by Unity.

When using LOD Groups and the LOD tree shaders even single sided leaf planes are supported.

The meshes for the **trunk** and the **branches** should slightly overlap in order to avoid gaps, which simply means: Model your branches and all sub branches as independent geometry and make them sink a bit into its parent. Please have a look at the provided demo trees to find out more.

Tessellation

If you plan to use tessellation on the bark, things will get a bit more complex. Please have a look into the chapter Tessellation to find out more, as tessellation will affect they way you have to model the geometry.

Materials

The importer only supports 2 materials: Bark and Leaf. Bark and leaf materials have to be split up into separate meshes. Only Meshes using the the bark material may have a second material assigned. Submeshes on meshes using the leaf material are not supported.

Both materials have to be named properly:

- The bark material's name must contain "bark" or "Bark".
    - In case you use a 2nd bark material its name must contain “bark2nd.
- The leaf material's name must contain "leaf, "Leaf", “leaves” or “Leaves”.

The script simply searches for the lowercased version of both, so naming e.g. the bark material: "My fancy Barkmaterial" would just be fine.

Using multiple leaf textures

In case you would like to use different textures with the leaf shader — like for dry twigs and fully green branches — you will have to manually create a simple **texture atlas** that contains both and lay out the UVs of the meshes for the dry twigs and fully green leaf planes accordingly.
Please add enough padding around each texture in order to reduce artifacts from mip mapping. Consider setting the texture’s wrap mode to clamp.

Using multiple bark textures

In case you would like to use different bark textures e.g. for the trunk and the branches you may create a second bark material whose name must contain “bark2nd”. Just assign the materials as you are used to. On import the resulting submeshes will be merged and the IDs of the assigned bark materials will be stored in vertex color blue which allows us to either sample texture set 0 or 1 in the Array shader – all within a single draw call. That being said it should be clear that the bark and bark2nd material share the same uvs and tiling values.

Two bark materials are only supported by the LOD Bark Array shader.
Find out more >

In case you want to use multiple bark textures using the tree creator like shaders things will get much more complicated. So i do not really recommend this.

But if so you would have to manually create a texture atlas too: Place the textures next to each other along the x-axis, shrink them about at least 2x16 pixel along the x-axis and create an overfill on that 16 pixels where you simply repeat the last row of pixels. Sounds complicated? It is…

Textures

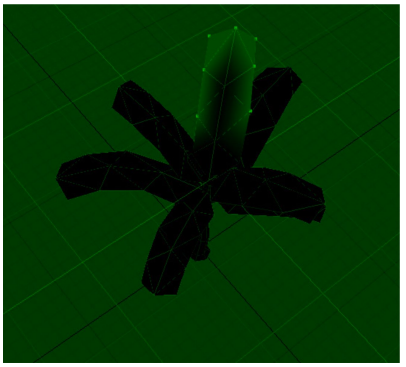
Just like Unity's built in tree creator shaders all CTI shaders use a bunch of combined textures in order to make the number of needed texture lookups as small as possible. Usually the tree creator would combine all needed textures but in our case we will have to do it on our own. Please note that the advanced tree creator shaders use a slightly different texture layout compared to the LOD tree shaders.

Find out more >

Vertex Colors

Some of the needed bending information is stored in vertex colors. While the importer script will set up most of them automatically, you have to assign **vertex color green** (which controls edge fluttering of the leaf planes) manually.

Setting up Edge Fluttering



Edge fluttering will add some high frequent movement along the given vertex normal. It is controlled by the vertex color's **green** channel which defines the edge stiffness: Green=0 --> no fluttering / Green=1 --> full fluttering.

Add edge fluttering by simply adding some green to the outer vertices of the leaf planes..

Tip: *Make sure that the connecting point (or pivots) of the leaves have vertex color green=0. Otherwise leaves might loose connection to their parent geometry.*

Adding Ambient Occlusion

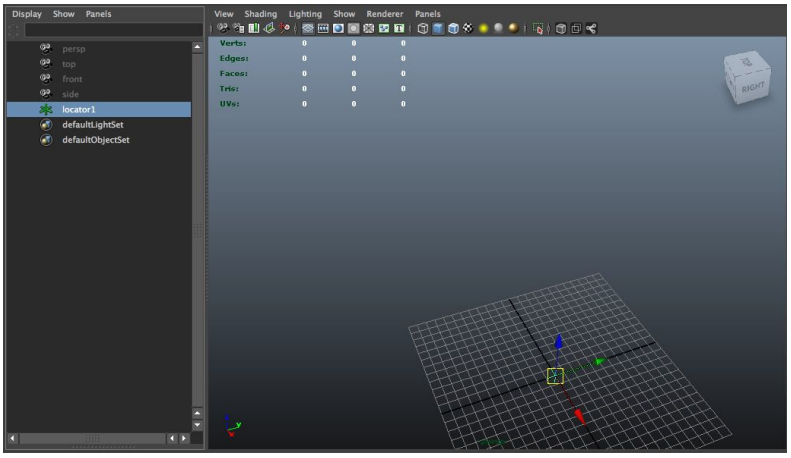
Ambient occlusion might be baked into vertex color alpha – depending on the bending mode you want to use. How you will do this is up to you and depends on your 3d App. In order to bake ao within Unity you can use a small script, which can be found it here: http://adrianswall.com/shared/unity_vertex_ao.rar

Quickstart Guide

Let's create a simple tree with just a trunk, a branch and one leaf plane.
According to the requirements we have to set up a proper hierarchy, proper materials and paint some simple vertex colors. Modeling the geometry, creating proper uvs or setting up textures will not be covered in this quickstart guide.

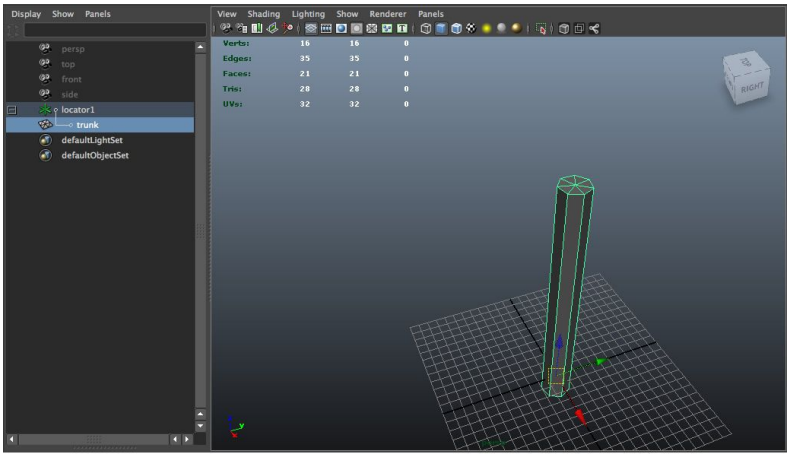
Basic Geometry and hierarchical Setup

Create an empty **locator** and position it at 0, 0, 0.



Add a simple cylinder as **trunk**, rotate and scale it properly and make it a child object of the locator.

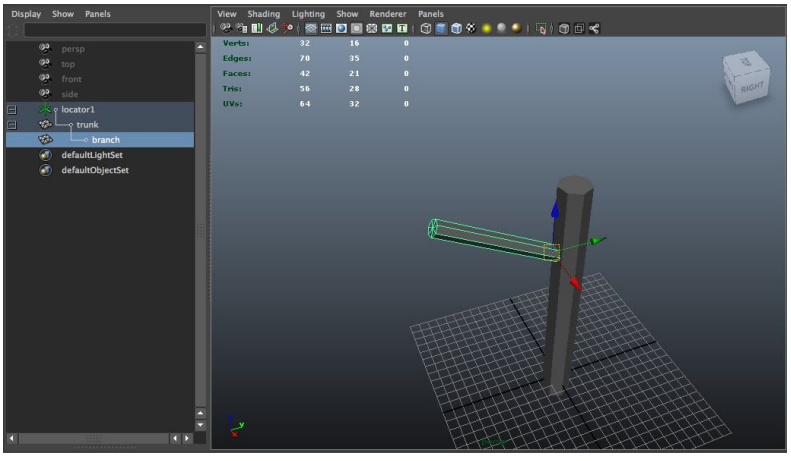
Move the trunk's pivot to the origin of the tree at 0, 0, 0.



Add a simple cylinder as **branch** or duplicate the trunk, rotate, scale and position it properly and make it a child object of the trunk.

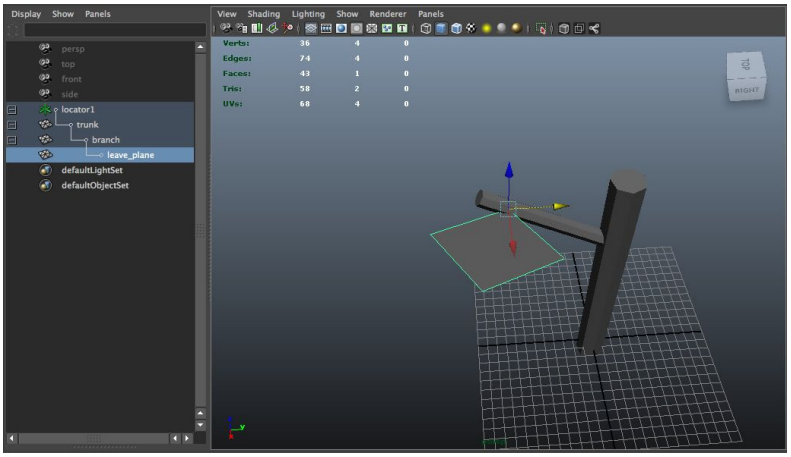
Make sure that the branch's pivot is located where the branch hits the trunk.

The branch should slightly overlap with the trunk in order to avoid small gaps.

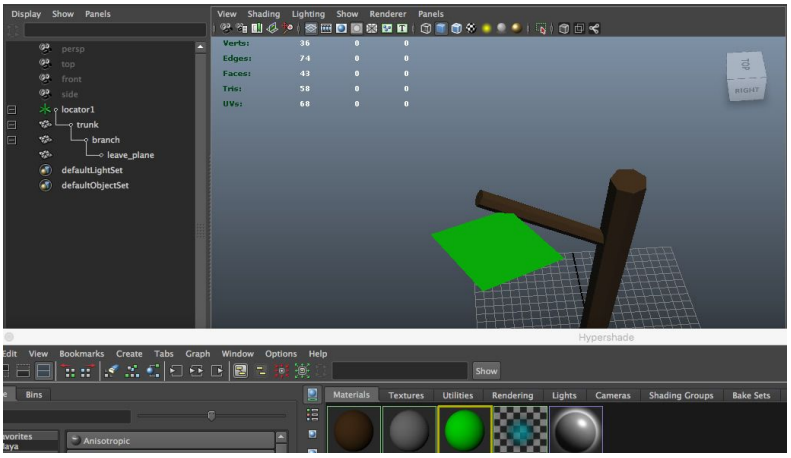


Create a simple quad as **leaf plane** and add it as child object to the branch on the first level.

Make sure that its pivot is located where it intersects the branch.

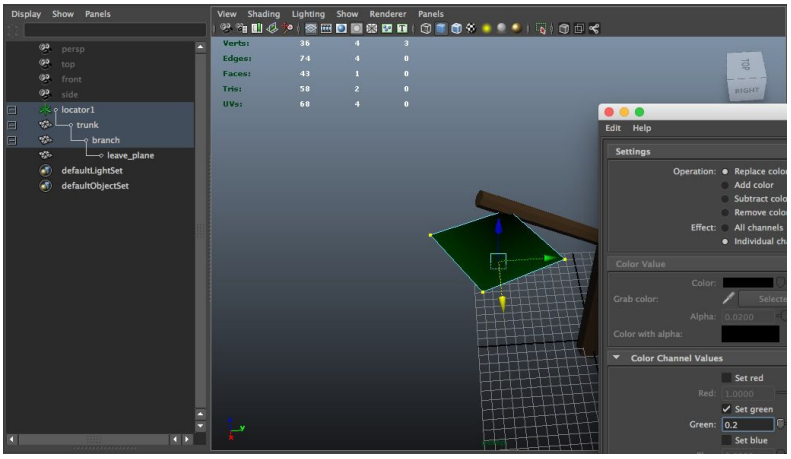


Create and assign the **bark and leaf materials**. Name them “*xyz* bark” and “*xyz* leaf” as otherwise the importer will throw an error.



Add **vertex colors** to the leaf plane in order to apply **edge fluttering**: Apply green = 0.0 to the pivot and green = 0.2 to the outer vertices which will result in a fairly gentle edge flutter.

Please note: As soon as you add vertex color green leaf planes might get torn apart in case a vertex has several varying normals (aka hard edges). So please always make sure that you leaf planes use smoothed normals



First test of the tree and its rotation

As the overall rotation of the tree is very crucial when it comes to the calculation of the bending and different 3d apps handle rotation differently in case you export your model as fbx, we should do a first export very early on to get this right. So let’s export the tree as fbx file – using “simple_tree_afsTREE.fbx” as file name.

Please note: “afsTREE” is the magic keyword which triggers the tree importer script. You

always have to add it to your file name.

Switch to Unity, then drag the imported fbx into the scene view. The tree should appear as single flat mesh (if the importer did a good job — otherwise there will be some error messages in the console). Now we can check its orientation:

The tree should stand upright and the rotation of its transform should be 0, 0, 0.

Most likely however the tree will not stand upright or its transform's rotation is not set to 0, 0, 0. *Fix this by going back to your 3d app and rotating the locator at level 0. For Maya 2013 this would be: 90.0, 0.0, 0.0.*

In order to do so unparent all child objects from the locator, set its rotation to 90, 0, 0. Then parent trunk, branch and leaf plane again. Finally safe the fbx and check it in Unity.

Try different rotations until you get a properly aligned upright oriented tree.

Test bending

Once the rotation is fine we can check bending by adding a built in **Wind Zone** to our scene. Raise its ***Main*** parameter (main wind strength) and look how the tree behaves while you are in play mode. If everything worked out nicely, the tree should bend according to the given wind strength and direction.

If branches or leaf planes get disconnected:

- You most likely have forgotten to set the pivots to where they should be located.
  Fix this by moving the pivots to the intersection of trunk/branch and branch/leaf plane.
- You may have to freeze the transformations on all child objects of the tree depending on your 3d app.
  Fix this by freezing all transformations on all objects.
- Tessellation (especially on the trunk of our example) might be too low so that the (linearly) interpolated bending values between far away vertices on e.g. the trunk do not match the baked bending values of relatively "highly" tessellated branch, which are calculated using a pow function.
  Fix this by adding 2 more loops to the trunk and maybe even the branch – in case the leaf plane gets disconnected as well. Freeze all transformations as well.

Fixing single sided leaf geometry

As we simply use "afsTREE" in the file name the tree will be imported to be compatible with the tree creator shaders. These need double sided geometry on the leaf planes so leaves will be invisible when seen from the back.
Fix this by making the leaf plane double sided or make the tree import as LOD tree by adding "_xlod00" to the filename. Let's add "_xlod00" to keep things simple...

We also have to manually subdivide the leaf's quad into triangles to get a proper shape in Unity. Do so by using the suitable tool of your 3d app and add an edge along the spine of the leaf plane.

Adding some more leaves and branches

Next add some more leaf planes by simply duplicating the existing one. Then duplicate the

whole branch two times and reposition, scale and rotate the copies so you have a tree with 3 branches.

Importing this tree and examining it in the wind you will see, that all branches and leaf planes sway according to the main wind strength, but look somewhat boring, as they all swing at the same phase.

We will fix this by adding our first control tags:

Adding variation using control tags

Phase variation

Select the 2nd branch and add "_xp03" to its name. Then select the 3rd branch and add "_xp06" to its name. This will give us a phase of 0.3 on the 2nd, 0.6 on the 3rd and 0.0 on the first branch and will result in slightly different bending. Check it out by exporting the tree again.

Adding wind strength and turbulence

As thin branches at the outer parts of the tree might bend more than thick branches and branches in the center of the tree we will increase the wind strength and turbulence to the highest (and thinnest) branch of the tree:
Do so by adding "_xw15" and "_xt20" to the branch's name.

Enabling leave tumbling

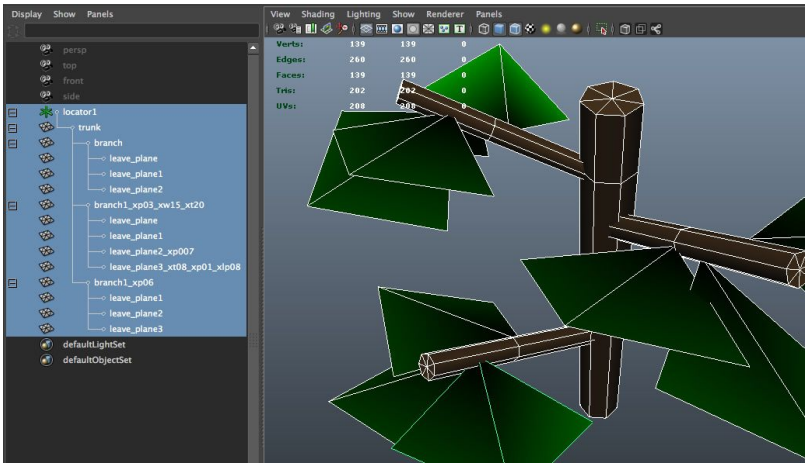
In order to enable leave tumbling add "_xlpr" to the tree's file name. Then raise "Leave tumbling" in the material inspector to "0.02".

Please note: Adding "_xlpr" or "_xlp" will make the importer script bake the leaf pivots into the mesh. If these are not baked raising "Leave tumbling" will break the tree.

After all the tweaks the final tree might look like shown in the screenshot to the right:

Please note the hierarchical structure and the control tags added to some of the levels.

Find the files in the folder "01 Quickstart Demo".



The result really is not a nice looking tree. But it is a nicely bending tree. And that is what the quickstart guide is all about. Now that you know, how to get rotation right, set up the hierarchy and add control tags, i hope, you can make much more complex and better looking trees :)

In depth information can be found in the next chapter.

In depth Guide

Assuming that you have either read the Quickstart guide or are already familiar with CTI this chapter will give you the most complete overview about the importer script, its needs and possibilities to tweak.

Hierarchical Setup

As mentioned in the section Requirements you will have to break up your tree into different and hierarchically nested objects, which will give the script a basic idea of your tree and its structure.

Level 0 – empty root or locator The root node of the tree must always be an empty transform. Make sure that its pivot is placed at 0, 0, 0.

Level 1 – trunk Next in hierarchy should be the main trunk. The trunk's pivot also should be placed at 0, 0, 0. The main trunk will not be affected by any turbulence — it will just be affected by the main wind making it sway in the direction of the given wind zone.

Level 2 – branches or leaf planes On the next level there are branches which are directly connected to the trunk. These may have different main wind settings, different main turbulence settings and also different phases.
Please note: In case you add different phases at upper levels of the tree, bending will most likely get corrupted. So always apply *main* phases on this level.

You may however add different phases directly to the last level (most likely leaf planes). In this case consider using only rather small values like 0.1–0.2. Otherwise the leaf plane might get heavily distorted as its pivot will pick up the phase from the levels below and only outer vertices will sway using the: (phase from the levels below) + (individual phase for this level).

You can also add leaf planes on the 2nd level of course.

Level 3 – small branches or leaf planes Next there might be small branches which are connected to the branches of the 2nd level and/or leaf planes.

Level 4 – branches or leaf planes

…

Adding more than 5 or 6 levels will most likely not be very suitable but should be handled by the script anyway.

Please note: The pivot of each element of the tree must match the position where it is connected to its parent as it is taken as origin for any calculation.

Most 3D apps need that you **freeze all transformations** before exporting your tree as .fbx (like modo 7 and maya 2013).

Fine tune Bending using Control Tags

In order to control main wind, main turbulence or per-leaf / per-branch phase variation you have to add **control tags** which defines the given parameter for the given level of the tree (and all its children). You may even add several different control tags to a single level.

Control tags (almost) always start with "_x".

Next there will be several letters which specify the parameter you would like to set.

Finally you add the value you want to set. Values have to be written as 2/3 digits without the period: "00" → 0.0f / "05" → 0.5f / "10" → 1.0f / "37" → 3.7f / “012” → 0.12f

So setting the wind strength of the trunk named e.g. "trunk" to 0.7 would be achieved by changing the trunk’s name to "trunk**_xw07**" (xw because it is the per level wind control tag).

Global Control Tags

Global control tags have to be added to the file name. They influence all nested objects and control e.g. overall bending and wind strength or enable special features like leaf bending, which needs additional data to be baked into the final mesh.

Main Wind

Adjust the overall wind force by adding "**_xmw**" + 2/3 digit value to the **filename.**
This will allow you to adjust the bending according to e.g. the size or overall bendability of the tree (as smaller trees should bend less in the wind).

Edge Flutter

Adjust the overall edge fluttering by adding "**_xef**" + 2/3 digit value to the **filename.**
This will allow you to scale the edge fluttering without having to rework vertex color green.

Leaf bending

The built in tree creator does not support secondary bending on leaf planes. But as this is needed on trees like palms (which only consists of a trunk and several leaf planes) the tree importer may also add secondary bending to all leaf planes.

Control the amount of leaf bending adding the corresponding "**_xt**" value to each leaf plane or set it globally by adding "**_xlb**" + 2/3 digit value to the **filename.**

Adding "**_xlb00**" disables leaf bending.

Leaf tumbling

Leaf tumbling will rotate each leave plane around its original pivot according to the wind’s direction and strength. This is a feature unique to the custom tree importer and will need the customized shaders to work correctly.

In order to activate leaf tumbling add "**_xlp**" to the to the **filename.** The script will automatically add UV3 and assign the needed shaders.

The general tumbling strength will be controlled by the material.

In case you want to have a more organic leaf tumbling you can alternatively add "**_xlprl**" to the **filename.** This will lower the tumbling on vertices close to the leaf plane’s pivot.

Leaf turbulence

Leaf turbulence will rotate each leave plane around its original pivot according to the main axis of the leaf plane.
Please note: As all transforms should most likely be freezed, we can’t rely on the object’s forward vector to get the main axis. So the script will calculate the main axis by taking the

pivot and the vertex, which is most far away from the pivot, into account. You may want to respect this when modelling your leave planes.

In order to activate leaf turbulence add "**_xlt**" to the to the **filename.** The script will automatically add UV3 and assign the needed shaders.

Adding leave turbulence will automatically enable leaf tumbling.

Mask bending

As rather thick branches tend to get disconnected from their parents — even if you have placed their pivots properly — you may mask the vertices around the fork or furcation using vertex color blue. So simply color the brnach with vertex color balck, then set the vertices, which should stay connected to their parent, to vertex color blue = 1.0.

In order to enable this feature you will have to add "**_xmvcb**" to the **filename**. Otherwise vertex color blue just gets ignored.

UV2

In case you want to use two uv channels on the bark material you have to add **"_uv2"** to the **filename**.

Level of detail

If you want to import different level of details of your tree which use the LOD shaders and support some basic transitions between the different levels you have to add **"_xlod"** and a 2 digit value (00, 01, 02) to the **filename** to define the level of detail the given mesh belongs to.

Find out more >

Please note: Adding "_xlod" will need you to add "_xlpr" or "_xlp" as well, as these control tags will make the importer script bake the leaf pivots into the mesh, which is needed to fade in and out leaf planes.

Merging Materials

CTI allows you to finally merge materials – which would reduce the rendering costs from 2 to just 1 draw call. In order to merge materials add **"_xmm"** to the **filename**.
Find out more >

Generate Mesh for AFS and ATG

In order to make the final mesh be compatible with the Advanced Foliage Shaders or the foliage shader of the Advanced Terrain Grass package you have to add **"_xafs"** to the **filename**. By default primary and secondary bending will be stored in UV4 so vertex color alpha can store ambient occlusion. Adding **"_xafs02"** instead will store all bending information in vertex colors only. Using "_xafs02" is mandatory in case you want to use the mesh with ATG.

Property	Tag	Description
Main Wind	**_xmw**	Add "_xmw" and a 2/3 digit value to the **file name**.
Edge Flutter	**_xef**	Add "_xef" and a 2/3 digit value to the **filename** to tweak the edge fluttering as stored in vertex color green.

Leaf Bending	**_xlb**	Add "_xlb" and a 2/3 digit value to the **filename** to control leaf bending.
Leaf Tumbling	**_xlp** **_xlpr**	Add "_xlp" or "_xlpr" to the **filename** to activate leaf tumbling.
Mask Bending	**_xmvcb**	Add "_xmvcb" to the **filename** to enable mask bending.
UV2	**_uv2**	Add "_uv2" to the **filename** to add uv2 to the submesh, which uses the bark material.
LODs	**_xlod**	Add "_xlod" and a 2 digit value (00,01,02) to the **filename** to define the level of detail the given mesh belongs to.
Merged Materials	**_xmm**	Add "_xmm" to the **filename** in case you want to merge materials.
Prepare for tessellation	**_xds**	Add "_xds10" to the **filename** in case you want the importer to prepare the mesh for tessellation. In case you want to globally reduce displacement along uv seams you may also use e.g. "_xds02" which would reduce displacement around seams by factor 0.2.
AFS and ATG Support	**_xafs** **_xafs02**	Add "_xafs" to the **filename** to generate a AFS compatible mesh. Add "_xafs02" to make bending information stored in vertex colors only (needed by ATG).

Per level Control Tags

In order to add more variation to your tree, control tags may be assigned per level as well.

Property	Tag	Description
Main Wind	**_xw**	Add "_xw" and a 2/3 digit value to the **level** of your tree which you would like to change the main wind strength for.
Turbulence	**_xt**	Add "_xt" and a 2/3 digit value to the **level** of your tree which you would like to change the turbulence strength for. *This probably is the most needed tag to add some some variety to your leaf planes.*
Phase Variation	**_xp**	Add "_xp" and a 2/3 digit value to the **level** of your tree which you would like to change phase variation for. **Please note:** In case you add different phases at upper levels of the tree, bending will most likely get corrupted. So always apply phases on branches directly connected to the trunk only. *You may however add different phases directly to the last level (most likely leaf planes). In this case consider using only*

		rather small values like 0.05–0.3. Otherwise the leaf plane might get heavily distorted as its pivot will pick up the phase from the levels below and only outer vertices will sway using the: (phase from the levels below) + (individual phase for this level).
Tumble Strength	**_xlp**	Add "_xlp" and a 2/3 digit value to the **level** of your tree which you would like to change the tumble strength for.
LOD	**_xlod**	Mark the branches and leaf planes that should be skipped at the specified level by adding "_xlod" plus 2 digits to the **level name**.
Smooth Normal	**_xsn**	Add "_xsn" and a 2/3 digit value to the **level** name in case you want to smooth out its normals. The numerical value must be in a range of 0.00–1.00. You also have to add "**_xmvcb**" to the **file** name and apply proper vertex color blue values as otherwise this feature will be skipped. See Per Vertex Settings for more details.
Displace seam edges *tesellation*	**_xds**	In case you use tessellation you may set the factor for the displacement at uv seams by adding "_xds" and a 2/3 digit value to the **level** name. This will overwrite the global value. See "Tessellation" for more details.
Per level Displacement *tesellation*	**_xdp**	Adding "_xdp" and a 2/3 digit value to a **level's** name will let you manually specify the the dampening of the displacement for the given level. See "Tessellation" for more details.

You can always set up most of these parameters on any level of your tree. However parents will pass their wind and turbulence settings to their children, so values added to child objects will always be relative to their parents:

If you want to e.g. to make the trunk being affected less by wind and set its main wind to 0.5 (by adding "_xw05") you can only raise the main wind on child branches using values above 1.0 (e.g. by adding "_xw15"), as using a value of 1.0 would be 0.5 according to the parent's setting. Setting the main wind on child branches to 0.8 would even apply less main wind to these.

Per vertex settings

You may add special attributes even per vertex by manually adding vertex colors. In case you do so please always start with adding vertex color rgb = 0,0,0 = black to all vertices of the given mesh – as black means: no changes. Then start to add blue and green as described below.

Prevent Discontinuities

As thick branches tend to get disconnected from their parents even if you have placed their pivots properly you may

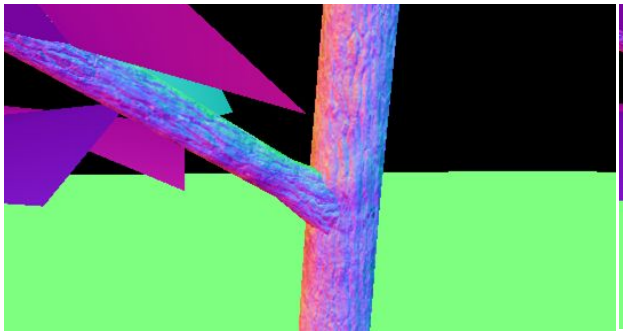
mask the vertices around the bifurcation using vertex color blue. Set the vertices which should stay connected to their parents to vertex color blue = 1.0 – all other should be set to black.

In order to enable this feature you will have to add "**_xmvcb**" to the **file name**. Otherwise vertex color blue just gets ignored.

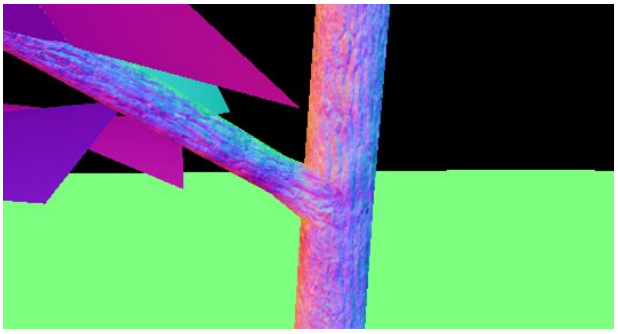
Smooth out Normals

Masking vertices around the bifurcation using vertex color blue also lets you smooth out the normals automatically. In order to activate this feature you have to add **"_xsn"** and a 2/3 digit value to the **level** name which in this case is the branch. The script will lerp the actual normal towards the closest normal of the parent mesh. The strength of the lerp is control by vertex color blue * the numerical value of the "_xsn" control tag. Not adding "_xsn" will make the script not touching the normals at all.

Please note: As the script simply looks for the closest vertex in the parent mesh you should make sure that you have some vertices on the parent mesh close to the bifurcation.



Regular normals



Smoothed normals

Dampen Displacement

In case you use tessellation and displacement you may add vertex color green manually to certain vertices in order to dampen the final displacement. Adding vertex color green = 1.0 would fully remove the displacement (and probably fix a seam) whereas adding vertex color green = 0.5 would reduce the displacement by 0.5.

Please note that the importer will add some dampening automatically according to UV seams and the given level. In case you want to disable latter you have to add "_xdp10" to the level. Then only your painted vertex color green + the automatic edge seam correction will be applied. Otherwise it sums up the per edge seam dampening + level dampening + your vertex color green value. In order to remove the automatic edge seam dampening you would have to add "_xds10" to the level's name. See "Tessellation" for more details.

Texture Settings

As already mentioned all CTI shaders rely on a number of combined textures — whose layout slightly differs between the advanced tree creator and the LOD shaders.

Advanced tree creator shaders

Leave Textures

Base (RGB) Alpha (A) Just like you would expect: Diffuse or albedo is stored in RGB, transparence in Alpha.

Normalmap (GA) Spec (R) Shadow Offset (B) This texture is the first combined one. Its channels should be set up like this:

R = Specular: A simple gray value
G = Green channel of the regular normal map
B = not used.
A = Red channel of the regular normal map

Please note: As this texture is NOT a regular diffuse texture nor a normal map you have to switch its import settings to "Advanced" and check "ByPass sRGB Sampling" – or uncheck “sRGB (Color Texture)”.

Trans (B) Gloss (A) This texture is the second combined one. Its channels should be set up like this:

B = Translucency
A = Gloss
All other channels are unused.

Please note: As this texture is NOT a regular diffuse texture nor a normal map you have to switch its import settings to "Advanced" and check "ByPass sRGB Sampling" – or uncheck “sRGB (Color Texture)”.

Bark Textures

Base (RGB) Alpha (A) Just like you would expect: Diffuse or albedo is stored in RGB, Alpha is not used.
Normalmap (GA) Spec (R) This texture is a combined one. Its channels should be set up like this:

R = Specular: A simple gray value
G = Green channel of the regular normal map
B = unused
A = Red channel of the regular normal map

Please note: As this texture is NOT a regular diffuse texture nor a normal map you have to switch its import settings to "Advanced" and check "ByPass sRGB Sampling" – or uncheck “sRGB (Color Texture)”..

Trans (RGB) Gloss (A) This texture is a combined one. Its channels should be set up like this:

RGB = not used
A = Gloss

LOD tree shaders

Leave Textures

Albedo (RGB) Alpha (A) Just like you would expect: Diffuse or albedo is stored in RGB, transparence in Alpha.
Normal Map (GA) Specular (B) This texture is a combined one. Its channels should be set up like this:

R = unused
G = Green channel of the regular normal map
B = Specular as simply grayscale as dielectric materials do not have a colored specular. You can specify different shades of gray in your texture but please note that the associated billboard shader simply uses a simple spec color value.

Please note: Spec color here is **linear**, so Unity's default 51,51,51 is way too bright. It should be RGB = 6,6,6 – but that seems rather dark. RGB around 10,10,10 however looks fine.

A = Red channel of the regular normal map

Please note: As this texture is NOT a regular diffuse texture nor a normal map you have to switch its import settings to "Advanced" and check "ByPass sRGB Sampling" – or uncheck "sRGB (Color Texture)". Trilinear filtering is recommended.

AO (G) Translucency (B) Smoothness (A) This texture is a combined one. Its channels should be set up like this:

R = unused
G = Ambient Occlusion
B = Translucency
A = Smoothness

Please note: As this texture is NOT a regular diffuse texture nor a normal map you have to switch its import settings to "Advanced" and check "ByPass sRGB Sampling" – or uncheck "sRGB (Color Texture)".

Bark Textures

Albedo (RGB) Smoothness (A) Albedo in RGB, alpha stores smoothness.
Normal Map (GA) Specular (R) AO (B) This texture is a combined one. Its channels should be set up like this:

R = Specular as simply grayscale as dielectric materials do not have a colored specular. Gets compressed pretty lossy... . You can specify different shades of gray in your texture but please note that the associated billboard shader simply uses a simple spec color value.
Please note: Spec color here is **linear**, so Unity's default 51,51,51 is way too bright. It should be RGB = 6,6,6 – but that seems rather dark. RGB around 10,10,10 however looks fine.

G = Green channel of the regular normal map
B = Ambient Occlusion
A = Red channel of the regular normal map

Please note: As this texture is NOT a regular diffuse texture nor a normal map you have to switch its import settings to "Advanced" and check "ByPass sRGB Sampling". Trilinear filtering is recommended.

Adding leaf tumbling and turbulence

All CTI tree shaders support tumbling and turbulence which are described in the chapter CTI Shaders >

Importing a custom tree using leaf tumbling and/or leaf turbulence

Add "**_xlp**" to the to the **file name** to activate **leaf tumbling.** The script will automatically add UV3 and assign the needed shaders.

The general strength will be controlled in the material.

In case you want to have a more organic leaf tumbling you can alternatively add "**_xlprl**" to the to the **file name.** This will lower the tumbling on vertices close to the leaf plane's pivot.

Please note: Meshes that are setup to use leaf tumbling should use unique materials and must not share materials with trees not set up for tumbling.

Adding "**_xlp**" plus 2 digits to either to a branch or a single leaf objects will let you control the leaf tumble strength per branch and leaf object/mesh.

In order to activate **leaf turbulence** add "**_xlt**" to the to the **file name.** The script will automatically add UV3 and assign the needed shaders.

Adding leave turbulence will also add leaf tumbling.

LOD Groups

The importer script and the provided LOD shaders support smoothly fading LODs using unity's built in LOD Group component.

To make LODs work you will need different meshes for the different LOD levels as well as a billboard asset and the billboard textures.

Creating LOD Meshes

The script won't optimize your meshes as 3d apps such as Maya, 3ds Max, Modo or Blender will most likely do a much better job and allow much more control over the final result.

The script may simply skip certain branches or leaves planes on importing certain LOD levels — and it will bake information into the mesh which allows the shader to smoothly fade in and out certain user defined leaf planes.

In order to be able to use the LOD import features add "**_xlod**" plus 2 digits to the **file name**. Use "_xlod00" for the highest LOD, "_xlod01" for the next and "_xlod02" for the last level. Currently only three mesh based LOD levels are supported.

Please note: Adding “_xlod” will need you to add “_xlpr” or “_xlp” as well, as these control tags will make the importer script bake the leaf pivots into the mesh, which is needed to fade in and out leaf planes.

Next you would have to mark the branches and leaf planes that should be skipped at the specified level. So add "_xlod" plus 2 digits to the **level name**.

Adding "_xlod01" to a **leaf plane** (using the leaf material) will make it fade out at the switch between LOD00 and LOD01.

Adding "_xlod02" to a **branch** (using the bark material) will make it be skipped on LOD02.

Skipping Leaf Planes

If you want to make certain leaf planes to fade out towards the next LOD level add "_xlod" plus 2 digits (number of next LOD level) to the plane.

Skipping Branches

If you want to skip smaller branches (using the bark material) add "_xlod" plus 2 digits (number of next LOD level) to the branch.

This will simple remove the geometry from the final model. Child objects like leaf planes still will be included and bend correctly. Currently there is no fading for branches. They simply pop in and out.

Please note: Using "_xlod" on branches will give you a warning: "combine mesh instance xyz is null" but that is just fine.

Example

Let's have a look at a highres LOD00 model (like the "Jatoba_Tree1_afsTREE_xlprl_xlod00.fbx")

Find the leaf planes which do not contribute much to the outer shape of the tree or are pretty small and add "_xlod01" to their name in the hierarchy.

In case there are very tiny branches you may do the same to them.

Next find leaf planes which you think might be skipped on the next LOD level and add "xlod02" to those.

Do the same for small and mid size branches.

Finally export your tree three times using different file names:

- "my_afsTREE_otherControlTags_xlod00_xlp"
- "my_afsTREE_otherControlTags_xlod01_xlp"
- "my_afsTREE_otherControlTags_xlod02_xlp"

Switching to unity you will get 3 assets with different vertex and triangle counts.

And if you put those into a LOD Group component you will get nicely fading transitions.

Testing the LOD meshes

You should test your LOD meshes as soon as possible. So simply create an LOD Group and assign your meshes to it in order to get a preview in the editor.
Tip: *Test your LOD meshes with wind enabled and disabled.*

Bounds

If you manually optimize your meshes by merging vertices or even delete some leaf planes completely in lower LODs the bounds of the imported tree meshes might not fit over the the different LOD levels which would lead to slightly different bending values calculated by the importer script.

- In order to get around this parent a cube under the trunk and size it so it encloses ALL vertices of all LOD levels. Make the bounding box a bit bigger just to be safe.
- Make sure you name it "bounds" so it gets identified by the script.
- In case you have different files for your LODs you have to copy/paste the bounding box to all files.

The importer script will skip all objects named "bounds" when generating the final mesh but they will be taken into account by unity when it calculates the outer bounds of your tree which control the bending parameters baked into the final mesh.

Special shader inputs of the LOD tree shader

As soon as you add "_xlot" to the trees name the importer will automatically assign the LOD tree shaders which need some different inputs compared to the tree creator shaders:

- **Color Variation** lets you define a color that gets lerped with the given albedo based on the trees position. As the shaders use a very simple color lerp function you should use a pretty low alpha value on this color as otherwise your tree will most likely get washed out. **Please note:** You have to make sure that leaves and bark on all LODs as well as the billboard material use the same color variation color.
- **Normal Map (GA) Specular (B)** In order to support physically based shading we need a specular color — which in our case is a simple shade of gray as dielectric materials do not have a colored specular color.
- **AO (G) Translucency (B) Smoothness (A)** We also need smoothness and might want to add ambient occlusion.

- **Use Wind from Script** Check this if you want to place the prefab whose material you are editing within the terrain engine. **Please note:** You have to make sure that leaves and bark on all LODs as well as the billboard material use the same wind input.
- **Fade out wind** In case your billboard does not use wind animation you might check this in the material use on the lowest LOD to get a smoother transition. Higher LODs must use a material with this feature unchecked.

Creating Billboards

In order to create the needed billboard assets and textures should duplicate the provided *"Create Billboard Assets"* scene.

On opening the duplicated scene you will see that the scene only contains a camera and a tree but no directional light. The reason for this is that we just use a simple flat ambient color (white) to render all the needed textures as we do not want to get any lighting baked into our atlases as billboards will be lit in real time.

In order to create the new assets for your own tree remove the tree currently in the hierarchy, place your tree (you should use the last LOD mesh model for this) in the scene at 0,0,0 and select the camera in the hierarchy.

You will notice that the camera has the *"Create Billboard Assets"* script assigned to it which provides some inputs, buttons and a texture preview.

Tree to render: Start by assigning your tree to this slot.

Single sided leaves: In case your tree uses single sided geometry on the leaf planes check this checkbox.

Power for Translucency: Defines the exponent of the power function which is used to calculate the final translucency mask: Value > 1.0 will darken the mask, value < 1.0 will brighten it. Default is 2.0. So just leave it that way and come back later when you have setup your LOD group to fine tune this value.

Center Tree: Hit this button and your tree will be centered and the camera will be set up to capture the whole tree.

Now your tree should be nicely centered in the game view and show up in the texture preview slot of the script.

If not please check the console for any errors and debug information.

Please note: The "Create Billboard Assets" script only handles CTI trees with max. 2 materials applied to them which must be using the "CTI/LOD Bark" and the "CTI/LOD Leaves" shaders.

Create Billboard Assets: Hitting this button will create all needed billboard assets like the billboard textures, the billboard material and the billboard asset, which is needed by Unity's BillboardRenderer component.

The upcoming Safe File Panel will ask you for a location and name: The name acts as a prefix for all files that will be generated. So simply name it e.g. "My_XYZTree_Billboard".

Creating all assets might take some seconds. When done the script will automatically add a game object called "_BillboardRenderer" to the hierarchy, which lets you preview the generated billboard assets and compare them to the mesh tree.

Please note: The texture preview will have changed to the combined normal, translucency and smoothness atlas. In case you want the albedo preview back simply hit “Center Tree”.

Render Albedo Atlas, Render Normal Atlas and **Save Texture** lets you bypass the complete creation process and will just perform the action as described on the button. *Save Texture* will save the current render texture as shown in the preview.

Current Billboard Asset is just some kind of control which lets you see, which billboard asset the script currently is working on.

Texture Atlas: Preview of the rendered texture atlas containing 8 views or sprites of the billboard positioned in 4 columns and 2 rows.

Tip: *Do not forget to save the scene so you can come back to it later in case you have to do any changes*.

Optimizing Billboards

The generated billboard asset always has a size of 1:2 – just like the size of the per sprite reserved texture space. According to your tree’s shape some texture space might be simply wasted (which we can’t change) and so will be fill rate (which we can change).

Tip: *Preview the wasted fill rate by lowering the Cutoff value of the billboard material to 0.0: Most likely you will get a lot of space above the treetop which will simply be cut off by the pixel shader.*

In case you have used “Create Billboard Assets” the automatically added game object “_BillboardRenderer” should have the script “OptimizeBillboardAsset” attached to. Otherwise add it manually.

This script lets you adjust the vertices of the billboard to safe some fillrate.

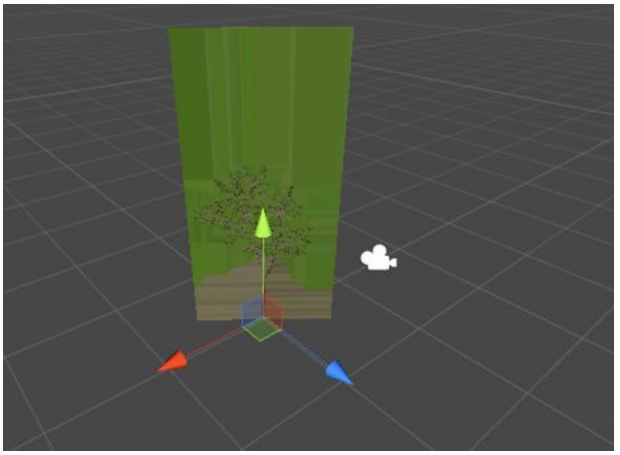
Get billboard asset: If no billboard asset is assigned to the field “Billboard Asset” simply check this once and the script will grab the billboard asset from the billboard renderer.

Billboard Asset: The billboard asset the script will work on. If this field is empty simply click the checkbox above.

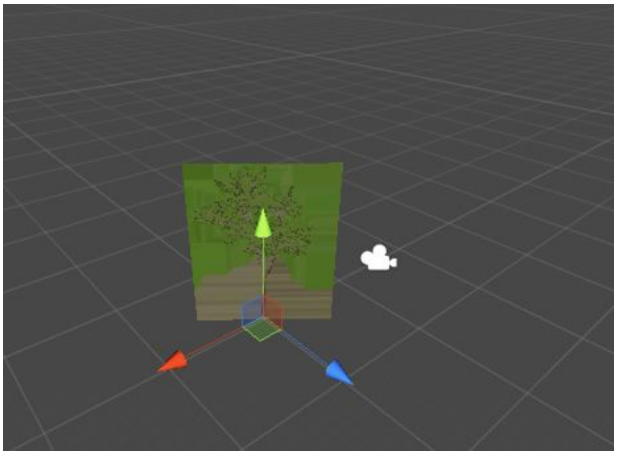
Tweak Vertices: Using the sliders provided here you can move the vertices of the billboard asset along the x and y axis in order to get the optimal shape like shown in the example below:

Start by lowering the Cutoff value of the billboard material to 0.0.

This will let you “see” how much of the fill rate is wasted – most likely at the upper parts.



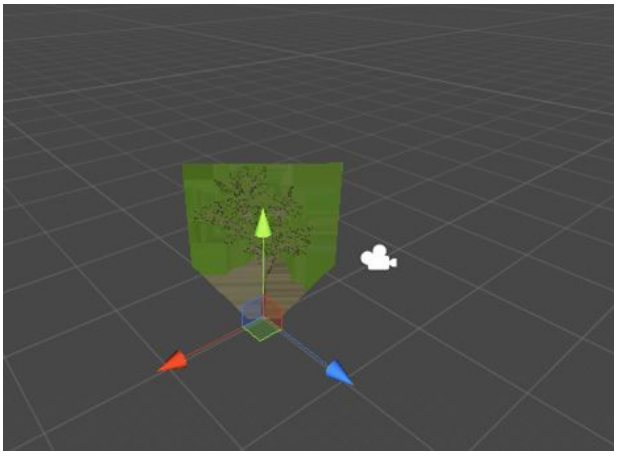
So lets fix this by lowering **Top_Y** until the upper edge hits the actual billboard texture. Top_Y would be something around 0.6 in this example.



Now lower **Middel_Y** to 0.3 so it is in the middle of Top_Y and Bottom_Y, then lower **Bottom_X** to something like 0.4 to cut away at least some of the wasted space below the treetop.

Important: Check the billboard from all angles and make sure that none of the sprites get cropped.

Make sure you save the scene when ready.



Please note: Although the scripts should have generated a pretty well fitting billboard asset height or width might be a little be off. You may always adjust these values manually by selecting the billboard asset in the project tab and then setting height and width in the corresponding inspector.

Setting up LOD Groups

In order to create a new LOD Group you can simply duplicate one of the provided LOD prefabs from the demos folder.

Otherwise you would have to setup it up from scratch:

- Create a new game object.
- Add a “Component” → "Rendering" → "LOD Group" component to it.
- Set the "Fade Mode" of the LOD group to "Speed Tree".
- You may or you may not tick "Animate Cross-fading"
- Now parent your LOD mesh trees under the new game object and make sure that they are all set to position 0,0,0 / scale 1,1,1 and rotation 0,0,0.
- Create the number of needed LOD levels in the "LOD Group" and add an additional one for the billboard.
- Assign the mesh trees to the according LOD levels within the LOD group.
- Finally you will have to add the billboard: Create an empty game object and name it “Billboard”, add a billboard renderer component and assign your custom billboard asset. Then parent the game object under the LOD group, position it at 0,0,0 and drag the “Billboard” to the last visible and not culled level of the LOD bar.

Using LOD Groups in the terrain engine

LOD Groups used in the terrain engine do not receive wind from built in wind zones. I guess this is caused by the fact that unity assumes to deal with a speed tree within that LOD group which comes with it's own wind.

So in order to have wind on those LOD Groups too you will have to add the "CTI_CustomWind" script to your wind zone. Or you can simply use the **"CTI Windzone" Prefab**.

The **Wind Multiplier** parameter lets you adjust the bending of your trees to make it fit the bending of any given Speed Trees on the terrain as these tend to bend quite heavily even at very moderate wind settings.

Next you will have to check "Use Wind from Script" in the bark and leaf materials. This will add support for directional wind. Radial wind zones are not supported.

Please note: As LOD trees in the terrain engine do not react to built in wind zones you may get rid of the "Tree" scripts on all mesh trees and the billboard on the prefab used in the terrain engine – as it is not used at all.

Optimize fading between mesh and billboard

The billboard shader supports wind which of course is a bit more expensive than not using wind. You may activate Wind on Billboards by editing the billboard material: Check **Enable Wind**.

In case you disabled wind for the billboard you should also edit the material of the last mesh LOD and check **Fade out Wind.**

Please note: This most likely means that you will need 2 bark and 2 leaf materials in case you have more than one mesh LOD because fading out wind between LOD containing mesh trees will look completely weird. So only the last mesh LOD should use materials which fade out wind.

CTI Shaders

CTI ships with two different shader families: the tree creator like CTI tree creator tumbling and CTI LOD tree shaders – both supporting leaf tumbling and turbulence.

Leaf tumbling and turbulence

Leaf tumbling will rotate each leave plane around its original pivot according to the **wind's direction** and strength.

This is a feature unique to the custom tree importer and will need CTI shaders to work correctly. Furthermore there will be additional information required that gets baked into UV3 of the generated mesh.

So using leaf tumbling will:

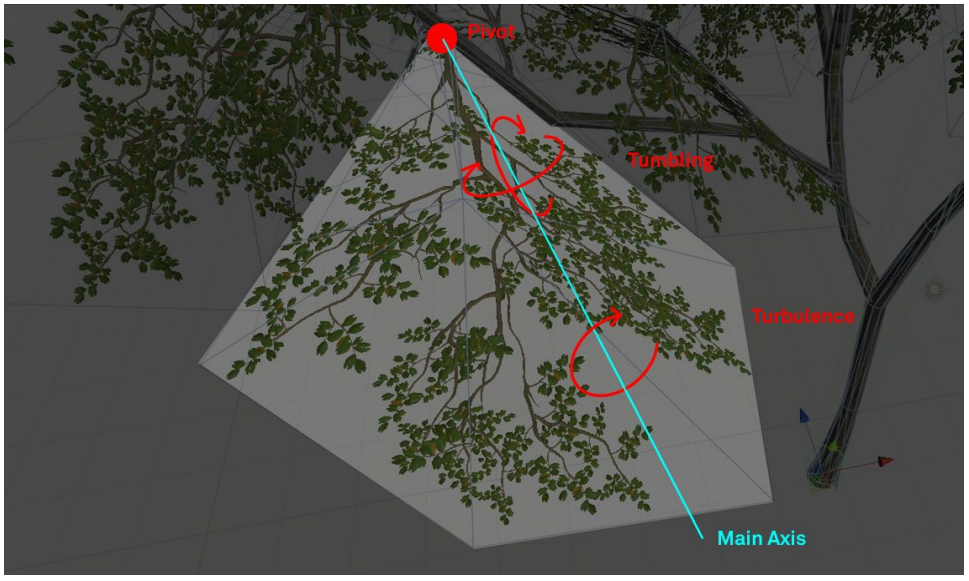
- increase the mesh's vertex count (+32bit per vertex).
- need a bit more expensive shader (which always uses Shader Model 3.0)

Please note: High leaf tumbling values (strength) might lead to leaves getting disconnected from the branches due to the fact that the method used is based on a packed Vector3.
Important: Due to the packed Vector3 you may NOT place any leave plane below the tree's pivot (y-position has to be positive).

Leaf turbulence will rotate each leaf plane around its original pivot according to its **main axis**.
Please note: The script will calculate the main axis by taking the pivot and the center of the bounding box into account.

Leaf turbulence will also take the edge flutter values (stored in vertex color green) into account and slightly distort or twist the leaf meshes unless you set the **Edge Flutter Influence** value in the material inspector to 0.0.

So while the direction of the tumbling animation is defined by the wind direction and therefore might change accordingly, the direction of the turbulence animation is more or less fixed as defined by the branch's main axis and only the strength of the of the wind influences turbulence:



Please note: Only tumbling and turbulence use rotations – which means, that only their animation may be reflected by the normal. Therefore checking: “Enable Normal Rotation” will only have an effect if either tumbling or turbulence or both are enabled. Primary and secondary bending as well as edge fluttering simply displace the vertices and do not change the normal.

CTI/LOD Bark shader

Shader Inputs

Color Variation Trees will be slightly different tinted according to their position in world space. RGB defines the tint color, alpha the strength of the tint. Always make sure that all shaders (leaves, bark and billboard) share the same color variation values.

Albedo (RGB) Smoothness (A) Diffuse texture which contains **smoothness** (unlike the leaf shader which expects transparency) in the alpha channel.

Normal Map (GA) Specular (R) AO (B) contains the combined normal, specular and ambient occlusion map. The channel layout of this texture differs from that of the leaf shader.

Secondary Maps (need UV2) In case you have created UV2 and made the importer to add it this drop down lets you specify how the shader should handle the secondary maps:

- **Disabled:** Secondary maps will simply be ignored.
- **Enabled:** Secondary maps will always be rendered. Use this e.g. on the material assigned to LOD00.
- **Fade Base Textures:** The base textures will be faded out towards the LOD switch. Use this e.g. on LOD01. Then use *Skip Base Textures* on LOD02.
- **Skip Base Textures:** Base textures will be totally skipped and only the secondary maps will be sampled and applied. Use this e.g. on LOD02.

  If you setup and assign the materials for the different LODs like described above LOD00 will use base and secondary textures, LOD01 will do too but fade out the base textures, while LOD03 will only use the secondary maps.
  Of course you can already fade out the base textures towards LOD01.

Swap UVS lets you swap UV0 and UV2.

Average Color (RGB) Smoothness (A) In case the shader skips the base textures on higher LODs, it still should add some “average” color and smoothness from the base texture in order to make the final result match the result from the LOD before to get smooth transitions. You may set this color manually or simply use the button *“Get average color”* at the bottom of the inspector.

Detail Albedo x2 (RGB) Smoothness (A) Secondary albedo and smoothness texture.

Normal Map (GA) Specular (R) AO (B) Secondary combined normal, specular and occlusion map.

Normal Strength Lets you adjust the strength of the secondary normal.

Use Wind from Script In case you place your LOD trees using the terrain engine you have to check this in order to make the trees receive at least proper directional wind.

Fade out wind In case your billboards do not use wind you may create an extra material just for the last LOD using mesh trees, assign it to the corresponding LOD and check this feature.

Doing so will make the final transition between the mesh tree and the billboard a little bit smoother. Do not check this on materials you use on LOD00 or LOD01.

CTI/LOD Bark Tessellation shader (beta)

Shader Inputs

The shader inputs are similar to those of the regular LOD Bark shader — except from the additional tessellation and displacement related inputs which are:

Tessellation Amount of tessellation or subdivisions. Usually values between 7 and 15 should be fine already.

Max Dist maximum Distance where the tessellation starts. You should keep this value pretty low e.g. 15 – 20m.

Min Dist Distance at which the maximum tessellation as defined by Tesellation will be reached. As the displacement will only happen if the Min Dist is reached you should keep this parameter pretty close to Max Dist (otherwise you would simply waste GPU power). So set Max Dist e.g. to 20 and Min Dist to 19.
Please note: I might drop Min Dist in future releases as i do not think that i really need it to get smooth transitions.

Extrusion Range: The range over wich the displacement or extrusion will fade in starting at the Min Dist.

Height Map (A): A regular height map (grayscale). Values will be read from the alpha channel. Black will push vertices downwards, medium grey won't displace them at all and white will pull them upwards along the given surface normal.

Displacement: The amount the vertices will displaced along the given surface normal in local space.

Please note: This shader does not support instancing on OpenGLCore right now. So instancing is not enabled (Unity 5.5). When using Unity >= 5.6. you would have to enable instancing manually anyway – doing so using OpenGLCore throws an error, however the shader will still work. Instancing and DX11 just works fine.

CTI/LOD Bark Array shader

Shader Inputs

The shader inputs are similar to those of the regular LOD Bark shader — except from the first two texture slots which expects texture arrays instead of simple image files with contain exactly two layers each:

Albedo Array (RGB) Smoothness (A) Texture array with two layers which contain the albedo color in RGB and smoothness in the alpha channel.

Normal Map Array (GA) Specular (R) AO (B) Texture array with two layers each containing the combined normal, specular and ambient occlusion map.

How to create texture arrays >

CTI/LOD Leaves shader

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…

Color Variation Trees will be slightly different tinted according to their position in world space. RGB defines the tint color, alpha the strength of the tint. Always make sure that all shaders (leaves, bark and billboard) share the same color variation values.

Albedo (RGB) Alpha (A) Diffuse texture which contains transparency in the alpha channel.

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.

Normalmap (GA) Specular (B) contains the combined normal and specular map.

AO (G) Translucency (b) Smoothness (A) contains the combined occlusion, translucency and smoothness texture

Translucency Strength acts as factor which gets multiplied with the translucency value sampled from the “AO (G) Translucency (b) Smoothness (A)” map and lets you fine adjust final translucency.

View Dependency determines when the translucent lighting effect will kick in depending on the view angle: Lower values will make translucent lighting appear already at rather flat viewing angles while high values will make it appear only if you look directly towards the sun. Values between 0.7 – 0.8 should be fine in case you want some kind of traditional thin layer translucency.

Fade out Translucency Lets you fade out translucency over distance as trees beyond the real time shadow distance tend to look a bit weird. If checked the leaf shader expects some global shader variables to be set as fade distance and fade range. You may set these manually from any script or use the provided `CTI.CTI_Utils.SetTranslucentLightingFade` function to so. Have a look at the LOD Group Demo to find out more.

Tumble Strength defines the strength of the tumbling animation.

Tumble Frequency lets you adjust the frequency of the tumbling.

Time Offset lets you shift the tumble animation in time so it comes slightly after the main wind animation.

Enable Leaf Turbulence You have to check this to enable leaf turbulence.
You may use turbulence even on meshes not having any UV3 in case you want to improve bending. If so make sure that "Tumble Strength" is set to 0 (to make the shader skip the whole tumble animation).

Leaf Turbulence lets you adjust the strength of the turbulence.

Edge Flutter Influence lets you adjust the strength of the edge flutter (stored in vertex color green) affecting the leaf turbulence. Using edge flutter influence values above 0.0 will most likely add some distortion to the leaf meshes – which in fact looks really nice.

Use Wind from Script In case you place your LOD trees using the terrain engine you have to check this in order to make the trees receive at least proper directional wind.

Enable normal rotation Checking this will make the vertex shader rotate the vertex normal according to tumbling. This is a bit more expensive but will improve lighting tremendously.

Fade out wind In case your billboards do not use wind you may create an extra material just for the last LOD using mesh trees, assign it to the corresponding LOD and check this feature. Doing so will make the final transition between the mesh tree and the billboard a little bit smoother. Do not check this on materials you use on LOD00 or LOD01.

CTI/LOD Billboard shader

Shader inputs

Color Variation (RGB) Strength (A) Make sure that the color fits the one you have added to the mesh trees.

Albedo (RGB) Alpha/Occlusion (A) This slot should contain the created albedo texture atlas.

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. A value of 0.45 should just be fine.

Alpha Leak Suppression: As the alpha channel of the Albedo textures stores both: **Alpha** and **Occlusion** dark pixels from the alpha mask might leak into the occlusion texture (caused by bilinear filtering) which would end up in full occlusion at the outer parts of the billboard. But if you set it about 0.6 all pixels darker than that will be set to white so you will get simply no occlusion on outer pixels – which in fact makes much more sense.

Normal (AG) Translucency (R) Smoothness (B) This slot should contain the created texture atlas.

Specular Specular Color as simple solid color which you most likely should set to the default value of dielectric materials which is RGB = 51,51,51.

Translucency Strength might be a bit different from the one set in the leaf shader as the billboard's translucency map contains some kind of depth map in order to reduce artefacts due to missing self shadowing.

View Dependency determines when the translucent lighting effect will kick in depending on the view angle: Lower values will make translucent lighting appear already at rather flat viewing angles while high values will make it appear only if you look directly towards the sun. The value here should match your settings in the LOD leaves shader.

Tree Height Limit (legacy) lets you optimize fill rate in case the used billboard asset does not fit the actual shape of your tree. Keep this at 1.0 in case you have optimized your billboards.

Enable Wind lets you enable or disable wind animations on billboards.
Please note: Only Wind from script is supported.

Wind Strength As Billboards do not have any baked wind information you may use this parameter to make the bending of the billboard better match the bending of the mesh tree.

CTI/LOD Debug shader

In order to visualize the data baked into the mesh by the script you can switch to the "CTI LOD Debug" shader.
Please note: The debug shader currently only handles leaves properly.

Shader Inputs

Vertex Color Channels Lets you view the different vertex color channels

Bending Lets you view the data baked to the UVs. Darker values mean less, brighter values mean more bending.
Please note: Bending has to be set to: "None" in case you want to have a look at the vertex color channels.

In order to preview and debug each of the accumulated bending animations you may enable/disable them using the proper check boxes.

Adjustments you make to the bending parameters will be stored of course even if you switch back to the original shader.

Further Shaders

CTI_Internal-DeferredReflections and CTI_Internal-DeferredReflections shaders

As CTI ships with its own custom deferred translucent lighting the package provides the needed deferred lighting and deferred reflection shaders, which can be found in: “CTI Runtime Components” → “Resources”.

CTI_Camera-DepthNormalTexture Shader

In case you use forward rendering image effects which rely on the depth normal texture such as SSAO need a proper depth normal texture, which will only be produced in case you have assigned the “CTI_Camer-DepthNormalTexture”.

This is caused by the fact that CTI shaders do vertex animations.

Please note: Make sure that your project does not contain any other "Camera-DepthNormalTexture" shader as this might overwrite the one, that ships with the custom tree importer. In case you use a package with also needs to modify the built in shader, you will have to manually merge both.

Advanced bark rendering

The CTI LOD bark shaders support different methods which allow you to increase the variety of the bark’s shading at relatively low costs: Texture Arrays (as provided by the LOD bark array shader) and adding a secondary texture set which uses UV2. You can even combine both methods.

Last but not least the bark shader also comes with a version which supports tessellation – which compared to the methods mentioned above – can’t be really called to be cheap.

Using texture arrays

The LOD Bark Array shader allows you to use 2 different texture sets on the bark material without having to draw the geometry twice – which usually is the case if you assign 2 different materials.

Actually the importer script detects if there is a 2nd bark material or if any of the bark meshes contains submeshes. If so it will flatten the materials and submeshes but store the original material's ID in vertex color blue. The pixel shader then will pick up the proper texture layer from the provided texture array according the given ID.

The shader always only reads set 0 or set 1, thus it will do only 2 texture reads per pixel (one for the albedo, one for the combined normal map) – unlike e.g. SpeedTree's branch detail shader, which does not support detail normals, smoothness or specular color but performs three texture lookups per pixel. Except from the increased amount of used texture memory using texture arrays more or less is for free.

Trunk and Branches

The easiest use case would be to have one texture set for the trunk and another for the branches.

- Create 2 bark materials named "xyz_bark" and "xyz_bark2nd".
  Please note: "bark2nd" is mandatory and needed by the importer script to detect the 2nd bark material. "xyz_bark" will map to ID = 0 → Texture Layer 0. "xyz_bark2nd" will map to ID = 1 → Texture Layer 1.
- Assign the "xyz_bark" material to the trunk and the "xyz_bark2nd" material to the branches.
- Save and import your tree, then drag it into the scene view.
  Please note: You have to save the tree adding "_xlod00" to its filename as texture arrays only are supported by the LOD shaders.
- If everything worked out well, you should see the processed and flattened tree mesh with only 2 materials assigned to: leaves and bark. The bark material should be set up to use the LOD Bark Array shader. Unfortunately you will not see any textures yet because you have to create and assign the need array first.
- So create the needed Albedo/Smoothness and Normal/Specular/AO arrays and assign them to the bark material. Creating Texture Arrays >
- Done.

Adding Details

Another use case might be adding details.

Just imagine a birch tree: Its trunk is rather rough and dark towards the roots whereas it is pretty bright on the upper parts.

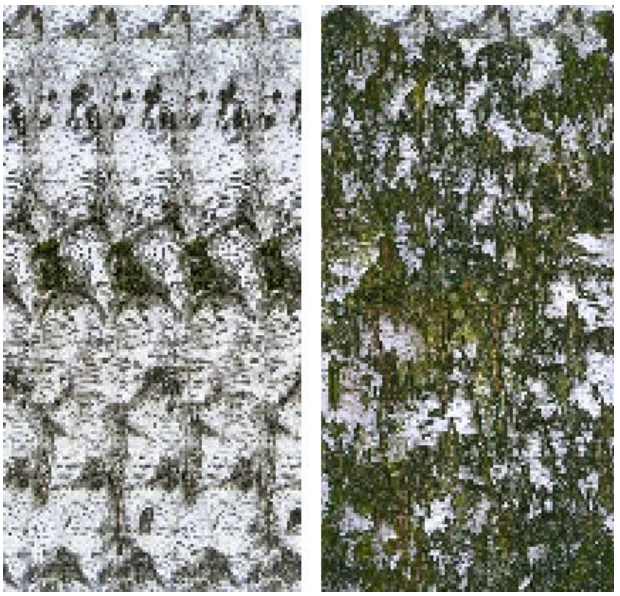
In order to create such texturing you would:

- either need a very large non tiling texture (like 512 x 4096px)
- or may use a dark detail texture on the lower parts and blend it with a bright tiling texture for the upper parts.

In this example we chose the second possibility. So we have 2 texture sets both sharing the same size of 512 x 1024 px: one for the upper (left), one for the lower parts (right).

The bright texture already tiles horizontally – but i simply did not have a better one, sorry.

The dark detail texture to the right contains a blend at the upper parts towards the bright texture created as described below.



As the array shader either reads texture set 0 **or** set 1 we can't blend between both sets in the pixel shader. So in order to get a smooth transition we have to create the blending in photoshop.

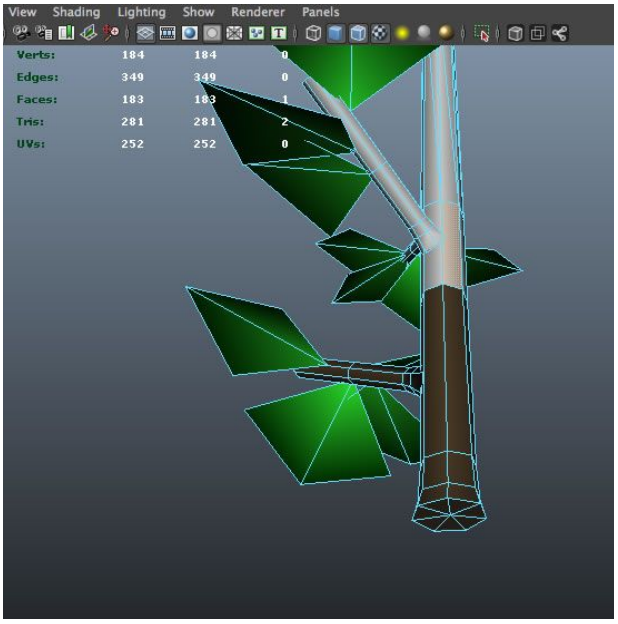
Simply take the detail texture for the lower parts of the trunk and copy it to a higher layer. Then copy the bright base texture for the upper parts into to background layer. Finally create the blend using the eraser tool or whatever you think will do the job.

Unfortunately you have to do this for all textures…

In case you want to add the detail texture somewhere in the middle of the trunk, you would have to add a blend at the bottom of the texture as well.

In your 3d app create 2 bark materials named “xyz_bark” and “xyz_bark2nd”.
Please note: “bark2nd” is mandatory and needed by the importer script to detect the 2nd bark material. “xyz_bark” will map to ID = 0 → Texture Layer 0. “xyz_bark2nd” will map to ID = 1 → Texture Layer 1.

Assign the “xyz_bark” material to all faces of the trunk. Then select the lower faces and assign the “xyz_bark2nd” material.

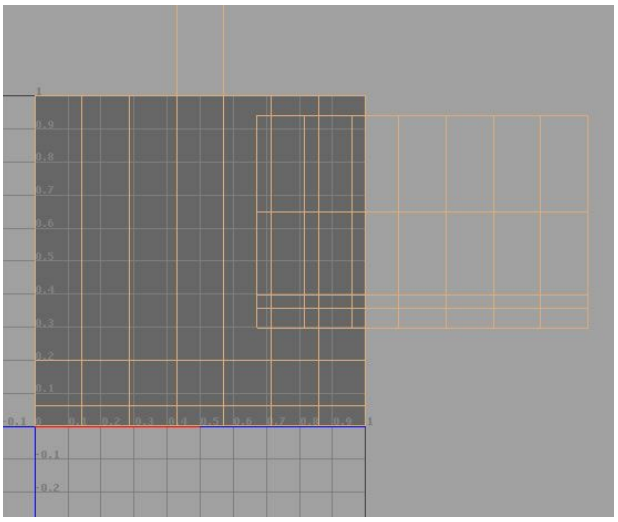


Brown faces have the “xyz_bark2nd” material assigned, white faces use the “xyz_bark” material.

The lower brown faces of the trunk should be mapped to v = 0.0 – 1.0. to create a seamless blending.

The highlighted face from the image above is represented by the single rectangle which starts at v = 1.0.

The uv shell on the right shows the uvs of the lower brown branch.



After importing the tree and having created and assigned the needed texture array the result in Unity might look like shown on the right.

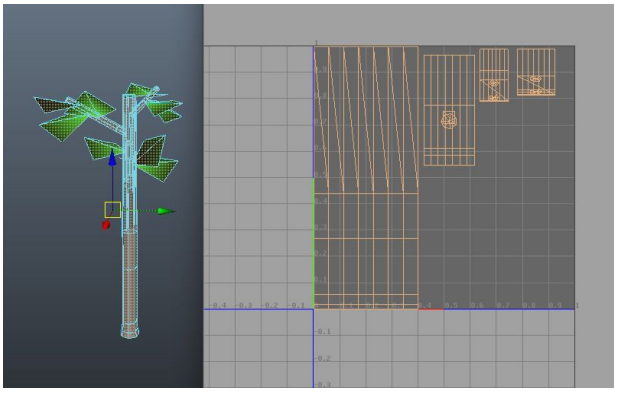
Creating Texture Arrays >

Using UV2

Now let's assume we want to texture a pine tree which has a brownish bark at the bottom and some kind of reddish at the top. This gradient usually would be too big to be represented using texture arrays.

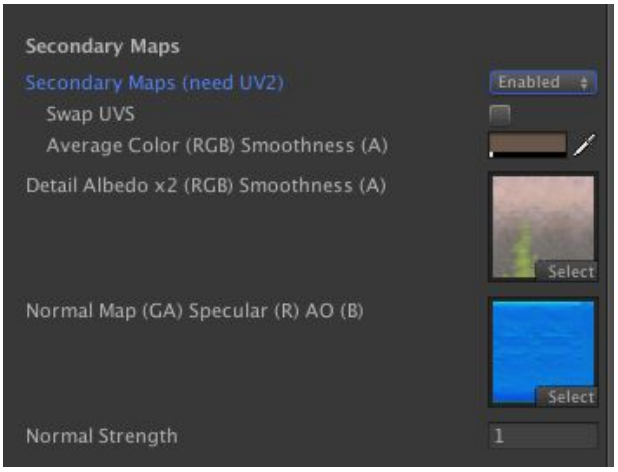
So for this case we can assign a secondary texture set which consists of albedo/smoothness and a normal/spec/ao map and uses UV2. This secondary set works a bit like you know from the standard shader — except for the fact that the secondary set support all pbs relevant parameters and most likely will not not add small scale but large scale details.

In your 3d app create a 2nd UV channel and map the trunk and the branches according to your secondary map.



Save the tree and add **"_uv2"** to it filename. This will tell the importer to keep uv2 on all meshes using the bark material.

Import the tree into Unity and select its bark material. Find "Secondary Maps (need UV2)" in the material inspector and set it to "Enabled".



Finally assign the secondary maps, done.

Pine trees using secondary maps (left) and without secondary maps (right).

Tessellation

Using tessellation and displacement (as latter is what you actually want) on the bark is a nice but pretty expensive feature — as tessellation itself is rather expensive on the GPU.

But as it just looks super nice, i have added this option to CTI as well. Just make sure you use it wisely.

Please note: The LOD Tessellation bark shader is currently in beta. It does not support instancing on OpenGLCore (some nasty errors) – DX11 however works fine.

Forward rendering is only partly supported – and probably way too heavy anyway – and as the shader is written as surface shader tessellation combined with bending is pretty expensive. So i will look into writing a CG shader.

When it comes to tessellation of the submesh using the bark material (leaves do not support tessellation) we are focussing several problems:

- **cracks** in the geometry.
- displacement on **different levels** of the tree like trunk, brach_1stlevel and branch_2ndlevel.
- a nice **transition** between tessellated and not tessellated geometry.
- **performance** costs on the GPU.

So let's address them one by one:

Cracks

In case you use tessellation and displacement (which we do, as it is all about this: we want to add more details to the trunk and branches) you may spot a lot of cracks in the geometry.

Whenever there are any discontinuities in your mesh und Unity has to split vertices – be it because of normals, uv seams or whatever – a crack will appear, which is a pretty common problem.

Normals should not be that big problem as we are talking about tree trunks and branches:
Simply make sure that everything is smoothes by raising the smoothing angle accordingly.

Knots may be a problem just like broken branches where you want a harsh edges – but in case you add a uv break to these the importer script will handle them automatically.

Otherwise you may simply add vertex color green = 1.0 to the vertices which show up cracks in order to eliminate these cracks.

UVs usually cause most problems. Assuming that your height map perfectly tiles (mandatory) as well as your UVs, the UVs on the trunk e.g. will go from 0 – 1 (along U) and then start back from 0. So you will have that break at 1/0 which will lead to slightly different samples of the displacement texture and cause cracks.

The importer may detect vertices with equal positions but different UVs and reduces displacement for these vertices and try to stitch the UV seams by adding a 2nd uv set.

In case stitching does not remove all seams – which may occur on very low tessellated geometry like super simple branches which have very long edges – you may simply set the displacement along the seam to 0.0 by adding **"_xds00"** to the name of the **level** which shows up cracks: Visually you will get one single edge (as shown in the screenshot below) if you apply "_xds00" e.g. right to the trunk, as vertices along this edge simply do not get displaced. This of course will be visible from certain viewing angles but should be just fine in case you lay out your geometry accordingly.

*Please note the **flat edge** at the left side of the trunk which marks the uv seam where displacement is set to 0.0.*

In order to break this up you may just add some more "real" vertices along that edge, add branches, ...

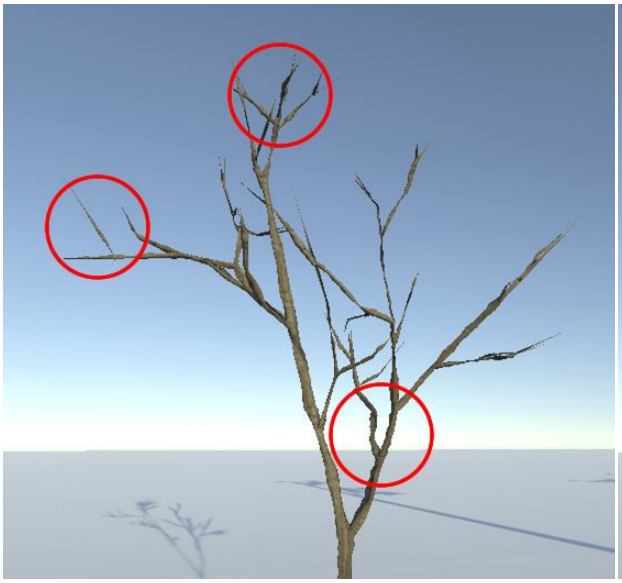
In fact the trunk used in this example is just super simple.

In order to make the importer automatically detect UV seams and apply the desired improvements you have to add **"_xds10"** to the **file name**.

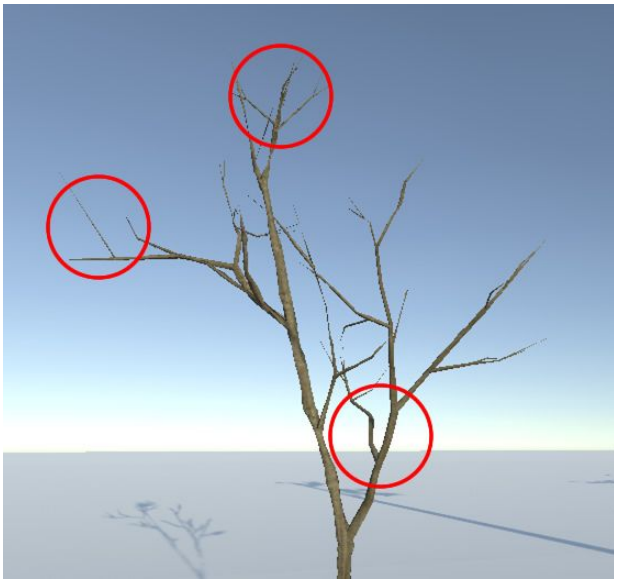
In case you want to globally reduce displacement at all seams you may also add a proper value here like "_xds02" which would reduce displacement at about factor 0.2 along all uv seams. You may always add an "_xds" tag to any **level** of the trunk and branches to get more or less displacement.

Displacement on different Levels of the branches

A displacement value of 0.1 would displace the vertices of the trunk about 10cm – which might be fine in case the trunk has a diameter of e.g. 1m. But now imagine a tiny branch with a diameter of only 5cm – its vertices would also be displaced by 10cm, so in the end the tiny branch would become a giant, fat and jagged something.



No damping at higher levels. Small branches tend to get jagged.



Regular damping at higher levels. Some small branches directly connected to the trunk at the top of the tree even have been set manually to use "_xdp00".

For this reason the importer bakes displacement factors into the mesh – according to the level of the given mesh:

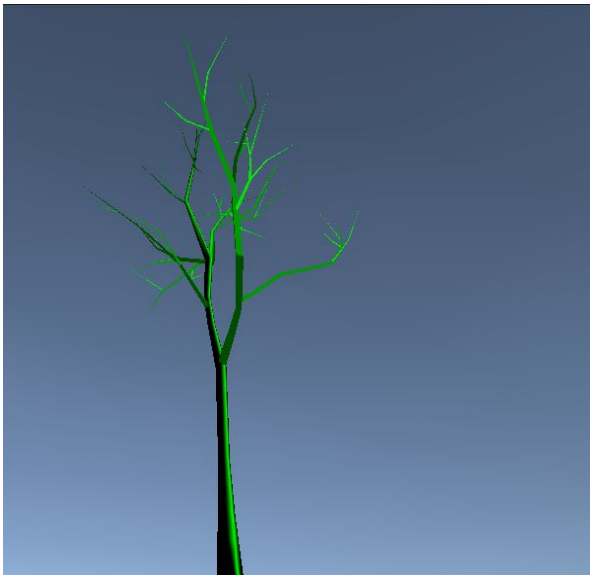
While the trunk's vertices will be displaced by the original displacement value as defined in the material, branches on the 1st level by default will use the displacement value * 0.3f, branches on the 2nd level will use the displacement value * 0.09f and so on.

You may however overwrite the displacement factor per level by adding **"_xdp"** and 2/3 digits to the **level's name**. Adding "_xdp02" to a branch would dampen the displacement factor from 1.0 to 0.2.

Displacement factors are baked to vertex color green. The image on the right lets you see the UV seam of the trunk, which is in this case is fully masked by green, as well as the the different shades of green added to the upper levels of branches: More green means less displacement.

Tip: *You may preview the baked colors doing the following:*

- *deactivate the wind zone,*
- *assign the "CTI LOD Debug" shader*
- *select: "Vertex Color Channels" = "Green"*



You may also **paint displacement factors** into your mesh. The script will simply add the calculated values on top of the vertex color green values already in the mesh. This may help you to fix some smaller cracks or add more variation to the displacement.

Transition

As you want to get rid of any tessellation shader as soon as you may (most likely from LOD0 to LOD1), we have to smoothly fade tessellation and displacement. In order to achieve a smooth transition the tessellation shader uses distance based tessellation and will fade in displacement only after the tree is fully tessellated.

Performance

Tessellation is quite expensive on the GPU. So do not use it over a large distance. 15–20m might already be enough. Next you should get rid of the tessellation shader as soon as possible, so make sure that you switch to LOD1 early which should use a regular LOD bark shader.

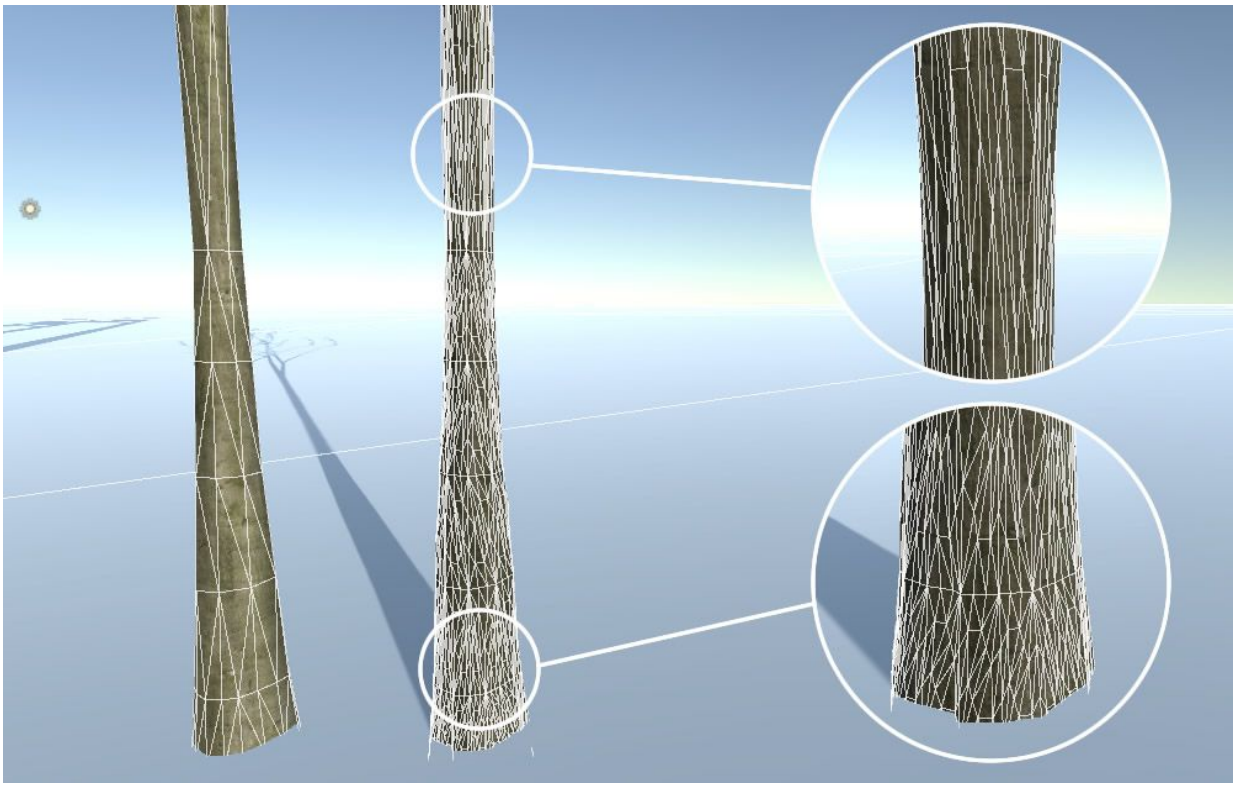
Also do not use super high tessellation values. Something between 9–15 should be fine.

Geometry

As tree trunks and branches usually use a rather high tessellation on the xz axis, but rather low tessellation along the y axis, edge based tessellation would be the best choice here. CTI instead uses distance based tessellation just because we want a stable shape and smoothly fading tree.

Basic modeling Guidelines

Using a distance based tessellation instead of an edge base one however needs you to tessellate the mesh more or less evenly along xz and y. At least you should avoid very “high” or stretched triangles along the y axis and add some more edge loops.



The right trunk shows the original geometry, which has rather dense loops towards the roots but only a few ones in the upper region. This will lead to a nicely tessellated mesh around the roots and only some very thin but high triangles in the upper parts – which will not let us do a nice displacement.

That being said you may already realize: The geometry you use with the tessellation shader might look different to that you use on lower LODs. At least it should have some more edge loops.

Bifurcations

While a branch may perfectly fit to the trunk using the regular LOD bark shaders displacement might add small gaps. You may fix this by scaling down the branch, or lowering its displacement factor – at least around the bifurcation.

Simple Roundup

If you want to use the tessellation shader start by:

- adding **"_xds10"** to the **file** name. This will make the importer add the needed fixes for UV seams and assign the proper shader.
- Assign a height map and raise the displacement so possible seams will become visible.
- If already the trunk shows up seams change the control tag of the file name to e.g. "_xds05". If only certain branches crack edit the model in your 3d app and add the control tag "_xds" to that branch only. Start by assigning "_xds05".
- Check the displacement of the branches. If they get too fat add **"_xdp"** and 2/3 digits to the **level's name** to lower the displacement on the given level only.
- If there are still some cracks you may manually add vertex color green = 1 to your model.

Next you should check the quality of the tessellation and

- switch to the "shaded wireframe" view mode in the scene view so you get a good picture of the tessellated geometry.
- In case you spot very thin triangles on rather exposed parts of the tree you may consider adding some more loops here in order to get more evenly distributed tessellation.

Last but not least check the bifurcations and if they still look fine when using displacement. If not lower the displacement of the child branch or simply scale it down a bit.

Examples

The Bark Tessellation Demo contains two different trees both using tessellation.
While the simple Pine Tree is a fully setup LOD group the Jatoba is just a single game object.

Simple pine tree

LOD group

- **LOD 00** uses a slightly more complex mesh than LOD 01 and the tessellation bark shader, which uses uv2 and fades out the base or detail textures towards the LOD switch. Some leaf planes fade out as well.

- **LOD 01** uses a more simple mesh and the regular bark shader, which skips all base textures. Leaf and bark material are set to fade out wind in order to make the transition between mesh and billboards less harsh.
- **LOD 03** is the billboard.

Please note: Even if LOD 01 shall not use any tessellation it must contain the same "_xds" control tag in the file name as LOD 00 in order to get the exact same bending on the leaves. The importer will automatically skip the tessellation processing on all LODs > 00.

Base mesh

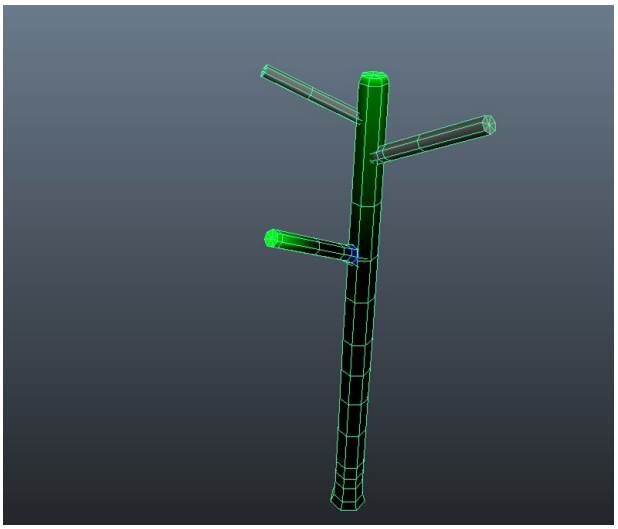
Most interesting of course is the base bark mesh of the pine tree which gets tessellated.

As its trunk as well as all its branches have sharp edges at the caps which simply create cracks, you will find the control tag "_xds00" in its file name: This reduces displacement on all UV edges to 0.0 and fixes all seams, but at the same time would break immersion – at least on the trunk.

So the trunk as well as the lowest branch have "_xds10" assigned, which brings back displacement at the UV edges.

In order to get rid of the cracks both use custom painted vertex colors green which reduce displacement towards the caps.

Please note: I also added one additional loop near the caps to get a bit more control.



Jatoba Tree

Base Mesh

The Jatoba comes with a much more complex geometry and does not include any sharp edges. So adding "_xds10" globally to the file name does not produce any cracks.

Nevertheless some branches uses special "_xdp" tags to reduce displacement in a custom manner: Especially thin branches in the upper regions which are directly connected to the trunk and therefore would get only a litte automatic displacement dampening.

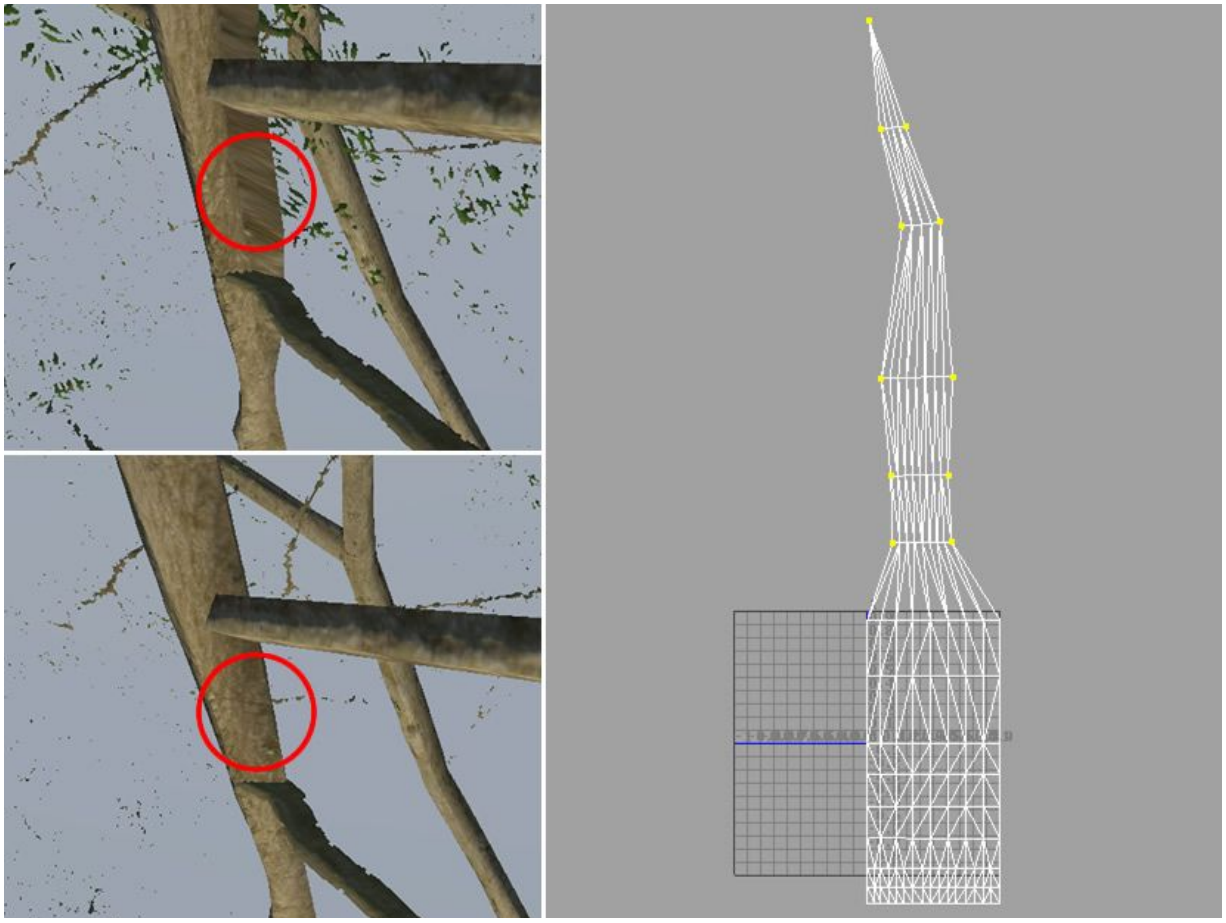
As the trunk is rather fat at the bottom and super thin at the top i added a custom gradient from vertex color green = 0.3 – 0.0 from top to bottom to reduce displacement towards the top.

Fixing not tiling UVs

Unfortunately the UVs of the trunk do not tile seamlessly on the upper parts which does not leads to cracks but something like steps or a sharp fin – like shown in the upper left screen shoot below.

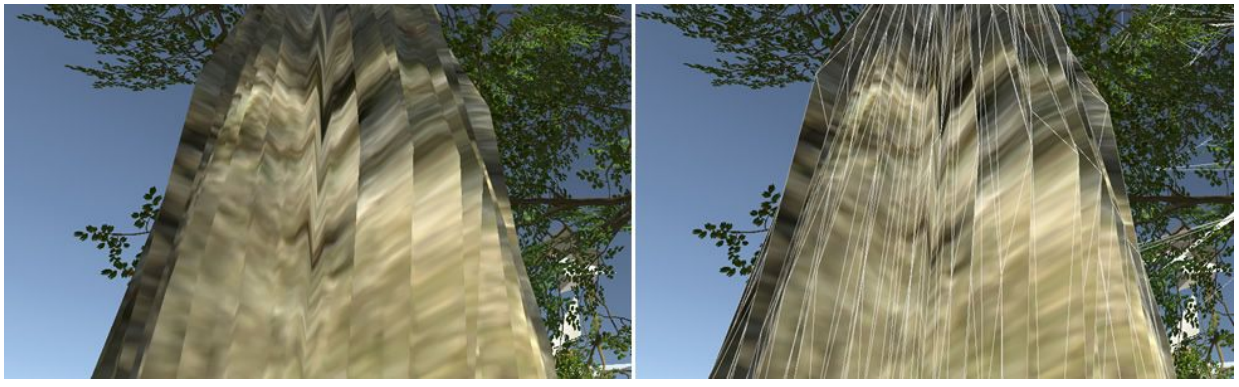
So here we actually have to set the displacement to 0.0 in order to get a nicely shaped trunk: Simply select the not matching UVs as shown on the right, then set vertex color green to 1.0.

The result is shown in the lower left.



Alternatively of course you may make the UVs simply tile.

I also added vertex color green = 0.5 –1.0 on some very long and stretched triangles as the displacement here simply created a totally jagged surface (due to missing loops):



Merging Materials

Usually the importer will expect and create 2 materials and submeshes leading to 2 draw calls for each tree, bush or whatsoever.

But imagine a small bush which might just have a thin and tiny stem followed by a vast number of leaf planes: Using a 2nd draw call just for this stem would be a waste of resources.

So for this case the importer lets you flatten the materials which will result in just a single draw call per instance.

In order to make the importer merge materials add **"_xmm"** to the **filename**.

In order to use merged materials, both bark and leaf have to share the same texture atlas as finally they will be rendered using the same material → textures. So you will have to create a proper texture and lay out your UVs accordingly.

Nevertheless you should still assign the "bark" material to all geometry, which should bend like bark – which in our example of the small bush would be the stem:
Using the bark material on the stem finally will make that part of your tree or bush just bend like if it was using the bark material and shader: No tumbling or turbulence will be applied and baked into the final mesh.

The importer will always assign the leaf shader by default.

Please note: I only recommend to merge materials on meshes like small bushes — where the trunk does not cover that much screen space (as the leaf shader will most likely do not do any backface culling).

Nesting Leaf Geometry

Nesting leaf geometry really is a nice feature which enables you to e.g. add some green leaves at the tips of a dry and alpha tested branch.

But in case you do so please keep in mind that:

- only the last child in the hierarchy will support tumbling and turbulence.
- you should not use "_xlp" on the parent leaf geometry as it would lead to discontinuities. Always only use "_xlp" on the second (main branch connected to the trunk) and the very last level.

Please note: In order to make 2nd level leaf geometry always stick to the upper level you should make sure, that the 2nd level geometry's pivot is stick to a vertex in the parent's geometry – or at least is located pretty close to a vertex.

LOD transitions

When it comes to nested leaves and LOD transitions you have to be a little bit careful.

As each leaf plane only know its own pivot nested leaves will fade into “nothing” if the parent leaf planes fades ot towards the same level of detail.

In case this looks too weird, you may consider fading out the leaf children towards LOD 01, whereas you make the parent leaf plane fade out towards LOD 02.

Please note: LOD control tags are not casted from parents to their children. So if the parent object uses “_xlod01” you will most likely have to add “_xlod01” to all child objects manually.

Texture Arrays

CTI ships with the **LOD Bark Array Shader** which lets you assign texture arrays instead of just a single albedo and combined normal texture.

Creating Texture Arrays

In order to create a texture array simply:

- select the textures in the project tab.
- click the right mouse button to bring up the context menu.
- choose: *CTI/Create 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.

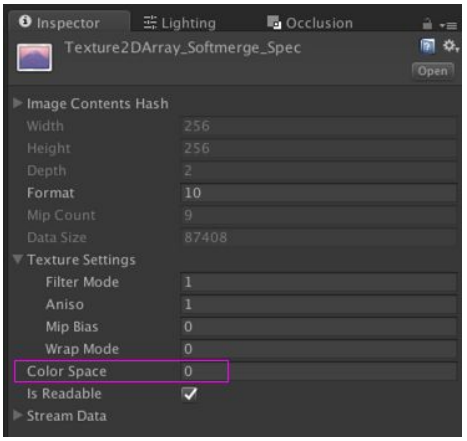
In case everything is fine the script will bring up the safe dialog and lets you choose a location and filename.

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 **Normal (GA) Specular (R) AO (B)** 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.



Importing Plants for AFS and ATG

You may use CTI to import plants (I am not talking about trees here btw.) which are compatible with the Advanced Foliage Shaders or the Advanced Terrain Grass package. You can not expect them to look as good as they do when using the CTI LOD shaders as they do not support tumbling nor turbulence but it should make it easier to set up complex plants for these shaders and iterate quickly.

As both shader families only support one material you have to lay out bark and leaves in a texture atlas. But in order to make life easier you may assign two different materials to your model: bark and leaf. Adding the bark material to e.g. the trunk of a small bush will tell CTI how to handle bending on this part of the geometry properly. So by assigning bark and leaf materials you will simply define how the corresponding parts of your model will bend – as finally the importer script will merge both materials into one.

Please note: As neither AFS nor ATG support leaf tumbling or turbulence it is worthless to use the corresponding control tags.

In order to tell the importer to convert the final mesh to be compatible with AFS or ATG simply add **“_xafs”** to the **file name.** This would create a mesh which uses UV4 to store main and secondary bending as supported by AFS when using its “UV4 and vertex colors” bending mode, and allow you to bake ambient occlusion into the vertex color alpha channel. In order to get a less fat vertex use **“_xafs02”**, which will make the importer to store all bending information in the vertex colors only.

In case you want to import meshes for ATG using “_xafs02” is mandatory.

Compatibility

APIs

DX9 and OpenGL are only partly supported as both APIs do not support instancing. I highly recommend to use DX11, OpenGLCore or Metal instead.

CTI and other packages

The CTI shaders should be compatible with almost all other packages — however and as CTI overwrites some built in shaders — there may be some conflicts with packages that do the same.

CTI also adds some custom shader passes to the built in "Camera-DepthNormalTexture" shader to make depth based image effects work correctly.

So it ships with a modified version which is located at: "CTI Runtime Components" → "Resources" → "Camera-DepthNormalTexture".

Please note: You must not move the shader as it relies on several includes.

Please note: Make sure that your project does not contain any other "Camera-DepthNormalTexture" shader as this might overwrite the one that ships with CTI. In case you use a package with also needs to modify this shader you will have to manually merge both.

CTI and the Advanced Foliage shaders

The advanced tree shaders of the AFS package are prepared to work with trees imported using the "Custom Tree Importer" package from the asset store and may even support tumbling.

Simply assign the “Afs Tree Creator Leaves Optimized CTI” shader to the leaf material.
The bark material can just use the regular “Afs Tree Creator Bark Optimized” shader – but you have to make sure that “Extra Leaf Bending” is set to 0.0.

Using the “Afs Tree Creator” shaders with CTI trees will make you benefit from advanced billboards blending and fading in shadows.

Alternatively you may edit the standard Afs tree shaders manually which would need you to tweak the following shaders:

- Resources/Camera-DepthNormalTexture.shader
- AfsTreeCreatorBarkOptimized.shader
- AfsTreeCreatorLeavesOptimized.shader

Please find and comment:
```
#define XLEAFBENDING
```

Then uncomment:
```
//#define LEAFTUMBLING
```

CTI and Lux Plus, Alloy or UBER

In order to implement deferred translucent lighting CTI ships with its own deferred light and deferred reflection shaders.

Lux Plus is simply supported out of the box. But as Alloy and UBER both bring their own deferred lighting (and 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 “CTI_TranslucentLighting.cginc” and simply uncomment either:
```
// #define USEALLOY
```
or:
```
// #define USEUBER
```
Then reimport the CTI folder.

In case you use UBER please note that CTI’s 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 Lux’s lighting.

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

The Demos

CTI ships with various demos which show you how to setup and use certain features.

By default all cameras used in these scenes are set to render in deferred. So in order to get proper visual results you first have to e to make sure that your project is setup properly.

Prepare your project

- Before starting please make sure that your project is set to use the **linear color space** in *Edit -> Project Settings -> Player.*
- In case your camera uses **deferred rendering** you also have to assign the **CTI deferred lighting shaders** in *Edit -> Project Settings -> Graphics*:
    - Under the *Built-in shader settings* change *Deferred* to *custom*, then assign the *CTI_Internal-DeferredShading* shader.
    - Also change *Deferred Reflections* to *Custom* and assign the *CTI_Internal-DeferredReflections* shader.

Quickstart Demo

This demo is linked to the chapter Quickstart Guide and contains the final tree created in this tutorial.

Base Demo

This scene shows various trees using both the advanced tree creator tumble shaders as well as the LOD shaders. You may explore the differences both shaders and get a picture of what using leaf tumbling and turbulence may add to the final wind animation.

LOD Group Demo

The demo contains a simple terrain and just the Jatoba tree which is placed manually as game object as well as using the terrain engine.

For latter reason it contains a “CTI Wind Zone” prefab which has the “CTI Custom Wind” script attached, which simply grabs the values from the wind zone and writes them to some global shader variables as needed by the CTI LOD shaders when used inside the terrain engine.

Manually placed trees use the **“Jatoba LOD Group”** prefab. As all child objects (mesh LODs and billboard) have the “Tree” script assigned, these trees react to Unity’s built in wind zone directly and may even be affected by radial wind zones. Billboards however do only receive wind from script.

Trees placed using the terrain engine use the **“Jatoba LOD Group for Terrain”** Prefab: All its child object do *not* have any “Tree” script attached and all materials are set to “Use Wind from Script”.

Translucent Lighting

As translucent lighting looks pretty weird in case there are no real time shadows, you will find a game object called “_SetShadowDistance” in the hierarchy, which has the “SetRealTimeShadowDistance” script assign and will raise the shadow distance to 300m on start.

This of course is not very suitable for a game. So you may insert any other value here.
The script then will call the:

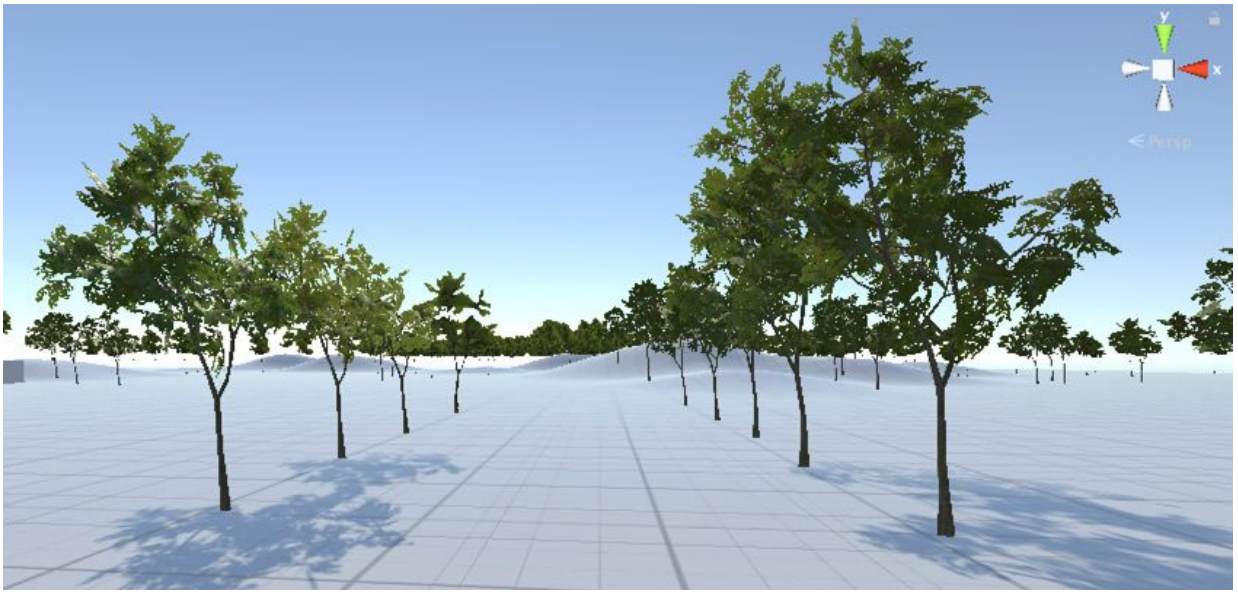
`CTI.CTI_Utils.SetTranslucentLightingFade` function, which will calculate proper fading values (distance and fade length) for the translucent lighting and send it to the shaders as global variable.

The function expect two float values as input:

- float TranslucentLightingRange: The range or distance over which translucent lighting will be applied – which most likely matches the given shadow distance in the quality setting.
- float FadeLengthFactor: Factor determining the actual fade range. If set to e.f. 0.2f translucent lighting will fade out over TranslucentLightingRange * 0.2f. So if TranslucentLightingRange

Please note: All leaf materials, that should use these fade global values, must have "Fade out Translucency" checked. Fading out translucency means also fading out smoothness.

Tip: *Instead of using "Fade out Translucency" you may e.g. make sure that your billboards kick in right before the shadows fade out.*



Trees on the left do not use "Fade out Translucency" on the leaf material. So already the 2nd tree in the row appears too bright – whereas the billboard looks rather dark.

Trees on the right do use "Fade out Translucency". Trees which aren't casting any shadows due to the given shadow distance fit way better.

Bark Texturing Demo

This demo lets you get a quick impression of the advanced bark texturing methods CTI offers as described in the chapter Advanced bark rendering.

Nested Leaves Demo

This is a super simple demo using a super simple model to demonstrate the possibilities of nesting leaf planes.

When opening the scene you will see just a single and pretty ugly bush – which actually has been made as simple as it is in order to show how nested leaves will behave.

Bending

- **Upper branch** uses nested leaf planes. The parent leaf plane does not support tumbling not turbulence – only the child leaves do.
- **Middle branch** consists of combined geometry therefore it tumbles as a whole. Variation only comes because the global control tag "_xlb" (enable leaf bending) and some vertex color green have been added.
- **Lower branch** uses nested leaf planes just like the upper does.

LOD fading

- **Upper branch** uses nested leaf planes which all fade out towards LOD 01. As it contains child leaves these will fade out into "nothing" – which may look a little bit odd.
- **Middle branch** consists of combined geometry. As it is set to fade out towards LOD 01 the whole geometry will shrink and fade, which usually just should be fine.
- **Lower branch** uses nested leaf planes just like the upper does. But unlike the upper branch only its children are set to fade out towards LOD 01 – while the parent leaf plane may fade out towards LOD 02.

Tessellation Demo

This demo shows how to use tessellation, get crack free geometry and optimize your meshes. Find out more in the chapter Tessellation > Examples

Change Log

Changes

- **Importer:** Pow value for main wind lowered from 2.0 to 1.5 which might lead to slightly different bending.
- **Billboards:** In order to create proper bounds for the LOD group i had to rework the billboards and the billboard shader. Billboards created with CTI < 3.1. therefore will be twice as tall as expected. Fix this either:
  - by editing the billboard asset and setting its *height* to: height * 0.5.
  - recreate the billboard asset using the *Create Billboard Asset* script
- **Shaders:** Variable for "Color Variation" renamed from _Color to _HueVariation: You may have to adjust your materials.
- **Translucent Lighting** reworked to make it match Lux Plus, AFS and ATG lighting. Use the new *View Dependency* parameter to adjust it. Translucent lighting may fade out over distance.
- **Render Billboard Atlas** script: Renamed to "Create Billboard Assets".

Improvements

- **Bark materials** using the LOD shaders **support UV2** which lets you mix an overall "color" and normal map along with a detail albedo and normal texture – just like you are used from Unity's standard shader: Using UV2 >
- **Bark materials** using the LOD shaders now **support up to 2 different texture sets** — all drawn within a single draw call and with just a single texture lookup: Using texture arrays >
- **Bark materials** may use tessellation (beta): Tessellation >
- **Leaf materials** may fade out translucency over distance when using the LOD leaf shader. LOD Group Demo >
- **Billboards** may be created with just a few clicks: Create Billboards >
- **Billboards** let you define a more precise shape which helps to save some **fillrate**: Optimize Billboards >
- **Shaders:** The material inspector of the LOD shaders will assign default combined maps in case you haven't assigned any in order to make rendering look less corrupted.
- **Shaders:** Wind Strength parameter added to CTI Billboard shader.
- **Shaders:** Some minor performance improvements and instancing support added.
- **Importer:** Control tags now support **2 digit and 3 digit values** to give you more fine grain control.
- **Importer** automatically sets *Material Naming* to *From Models's Material* – finally :)
- **Importer:** Support for nested leaf geometry added: If leaves contain further leaf objects only the last child objects in the hierarchy of the given branch will support tumbling and turbulence. Nested Leaves Demo >

- **Importer:** New global control tag added: Add "_xmm" to the filename and the importer will merge bark and leaf materials and submeshes and assign the leaf material by default. Merging Materials >
- **Importer:** Masking tree branches using vertex color blue at the bifurcation now produces way better results and makes almost all branches stick to their parent geometry – as calculations now are done per vertex.
  It also lets you automatically smooth out the normals towards the bifurcation. Smooth out Normals >
- **Importer:** When working on LODs the importer will try to calculate the main branch axis (needed by leaf turbulence) always based on LOD 00 which in case the bounds of a leave plane changes between LODs will lead to less popping.
- **Importer** lets you create meshes which are compatible with the Advanced Foliage Shaders and The Advanced Terrain Grass packages. Importing Plants for AFS and ATG >

Fixes

- **Create Billboard Assets** script: Small fix for asymmetrical trees which are now better centered automatically.