This is the offline version of the Wiki. For the most recent version, please look at our GitHub page.

Tasty Grass Shader v2.2.1

Tasty Grass Shader is a procedural framework for grass, vegetation and debris. Leveraging modern graphics techniques like Compute Shaders and Indirect Draw, it is very efficient at rendering hundreds of thousands of blades!

Links

- Watch the release trailer on YouTube
- Download the demo from itch.io
- Buy it on the Unity Asset Store

Contact

- Discord (Symmetry Break Studio)
- Email (support@symmetrybreak.com)

Features

□Powerful procedural generation

- Scriptable Object-based settings system
- Use any grayscale texture to control the distribution of parameters, such as: height, angle, bending, thickness, and color of grass blades
- Up to four customizable layers of variation

- Several ready-to-use noise textures included.

Fast and beautiful

- Separated baking and rendering pass at load time for maximum runtime performance.
- Control density over distance via template settings and graphics quality settings.
- Optional use of Alpha-Clipping for finer grass blade shapes.
- Support for Multi-Sampling Anti-Aliasing (MSAA) and Alpha-To-Coverage in URP Forward for maximum visual fidelity.
- Renders hundreds of thousands of blades, often under one millisecond, on a modern mid-range GPU.

Wind and collision system

- Simple and effective procedural wind.
- Tweakable wind speed, strength and patterns.
- Use sphere colliders to make objects interact with the grass.

Flexible

- Use custom textures for grass blades.
- Create your own plants and flowers.
- Also usable for ground debris, like fallen leaves and pebbles.
- Slope cutoff control.

Easy to get started

- 14 presets for grass, flowers and debris included
- Several grass textures included
- 4 wind presets included

Support for Unity Terrain

- Grass can be assigned to entire terrain, a single Texture Layer or painted..
- Built-in chunk system for optimal runtime performance on large terrains

Support for any Unity Mesh

- Height controllable via vertex color.
  - Vertex colors from mesh supported.
  - Vertex colors from Polybrush supported.

Supports 3D (Forward and Deferred) and VR (Single-Pass Instanced and Multipass)

Hack-friendly source code

- Support for custom rendering materials.
  - Via Amplify Shader Editor.
  - Via Shader-Code.
- API for loadscreens.
- Independent use from Unity components supported.
- Endpoints for custom terrain solutions included.
- All important functions are exposed.

Requirements

- Support for Shader-Compile Target 4.5. (DirectX 11 feature level 11+, OpenGL 4.3+, OpenGL ES 3.1+, Vulkan, Metal.)
- Unity 2021.3+
- Universal Rendering Pipeline 12.1+ OR High Definition Render Pipeline 12.1+ (Beta)
- (Burst 1.6+)

Support & Contact

If you encounter any issues or have questions, we are happy to help via:

- Discord: https://discord.gg/RgJxqkNYqR
- GitHub: https://github.com/SymmetryBreakStudio/TastyGrassShader
- Email: support@symmetrybreak.com

Further Notes

- The skybox which is used for presentation screenshots is not included in the package and is sourced from AllSky - 220+ Sky / Skybox Set.

Roadmap & Feature Requests

Please refer to our GitHub issues page.

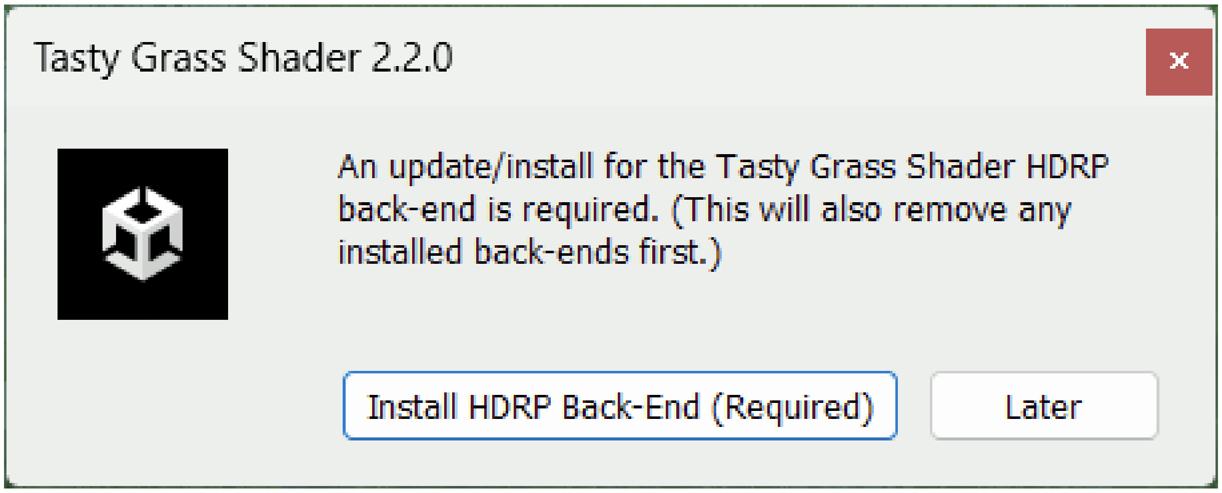
Before you start

As always, make sure you have the most recent version of Tasty Grass Shader installed.

Warning

Before you install a newer version, remove the old installation folder found in "Assets/Symmetry Break Studio/Tasty Grass Shader".

After the installation of the Package from the Asset Store, TGS will ask you to install a back-end for the current render pipeline (URP or HDRP).


Tasty Grass Shader 2.2.0

An update/install for the Tasty Grass Shader HDRP back-end is required. (This will also remove any installed back-ends first.)

After clicking `Install`, Tasty Grass Shader is ready for usage.

For Level-Artists

Adding Grass to a Mesh / MeshFilter and MeshRenderer

1. Add the "Tasty Grass Shader For Mesh" component to the Game Object you want to use.
2. Add a new layer.
3. Click on the new layer, open the Settings section and select a preset.

Note

The mesh must be readable. In most cases, Tasty Grass Shader can assist you in importing your Mesh as readable. See the Unity documentation for more details.

Paint grass using vertex colors (requires the PolyBrush Package)

1. Select the layer you want to paint grass on.
2. Set the distribution to a color that should mask the grass, for example to "Red".
3. Open the Polybrush window and start painting on the mesh, using the color you previously selected.

Note

Some meshes might have a white vertex color by default, leading to grass growing everywhere. You need to use a black brush then to remove grass.

Adding grass to a Terrain / Unity Terrain

1. Add the "Tasty Grass Shader For Terrain" Component to your Game Object.
2. Add a new layer.
3. Click on the new layer, open the Settings section and select a preset.

Grow grass on only a specific terrain layer

1. Set the target terrain layer.
2. Under the layer's Distribution settings, choose the option "By Terrain Layer".

Blend the grass with the color of the ground (new in 2.0)

1. Select the target terrain layer.
2. In the settings, adjust the Camouflage slider.

Adjust the slope cut-off

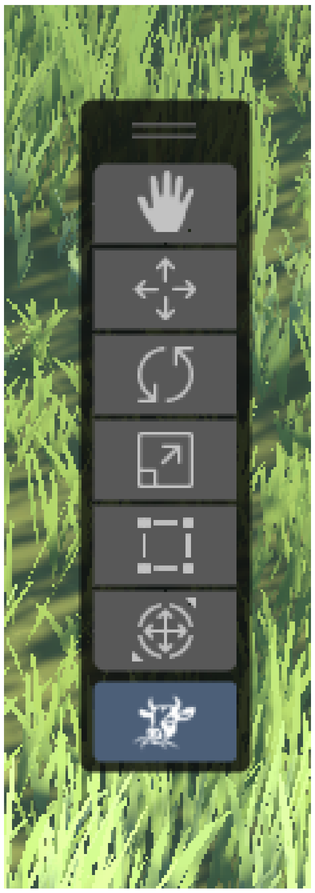
1. Select the target terrain layer.
2. In the settings, adjust the Slope Bias slider.

Paint Grass by Hand (new in 2.0)

Warning

The terrain must have a Terrain Collider Component for the painting tool to work.

1. Under the layer's Distribution settings, choose the option "Tasty Grass Shader Paint Tool".
2. Open the paint tool by clicking the Tasty Grass Shader Icon in the Unity Toolbox. (Note that this tool will only show if you select a terrain)



3. Make sure to select the right terrain layer in the "Target" section.
4. Start painting by dragging the mouse over the terrain while holding the left mouse button.
5. Erase grass by holding Shift and dragging the left mouse button.

Tip

You can use the "+", "-" and "=" buttons on the Terrain Splatmap section to add, remove or clone from the terrain's splatmap. For example, the clone ("=") function can be very useful as a starting point.

Note

Currently, you can not paint/place grass between triangles of the terrain. (The internal resolution of the distribution map is tied to the heightmap resolution of the terrain.)

Adding a Collider

1. Add the `Symmetry Break Studio/Tasty Grass Shader/Collider` Component to the Game Object.
2. Adjust the size to your desire.

Tip

For best results, make the collider radius slightly bigger than the object itself.

Note

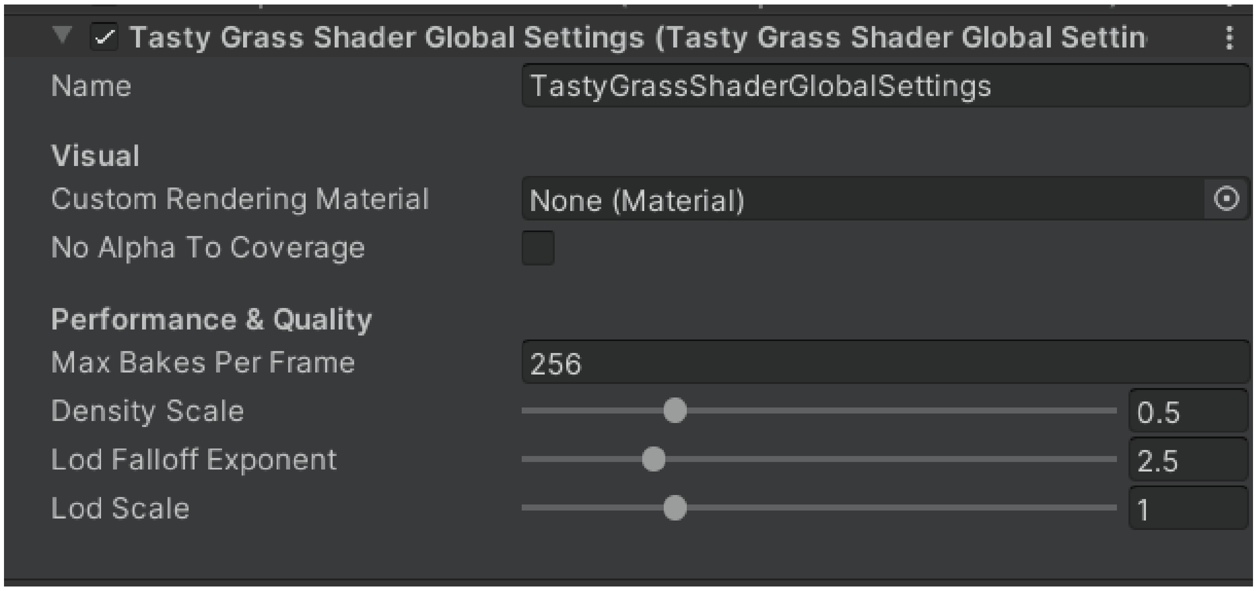
You can only have up to 8 colliders per Tasty Grass Shader Instance (= a mesh or a chunk on a Terrain).

□ For Technical-Artists

Adding Global Quality Settings (URP)

1. Navigate to your `Universal Renderer Data Asset` (often called `ForwardRenderer.asset`), used by your current `Universal Render Pipeline Asset`.
2. Click `Add Renderer Feature` and select `Tasty Grass Shader Global Settings`.

The general idea is to have multiple `Universal Renderer Data` assets for different levels of fidelity.



Tasty Grass Shader Global Settings (Tasty Grass Shader Global Settin

Name: TastyGrassShaderGlobalSettings

Visual

Custom Rendering Material: None (Material)

No Alpha To Coverage: ☐

Performance & Quality

Max Bakes Per Frame: 256

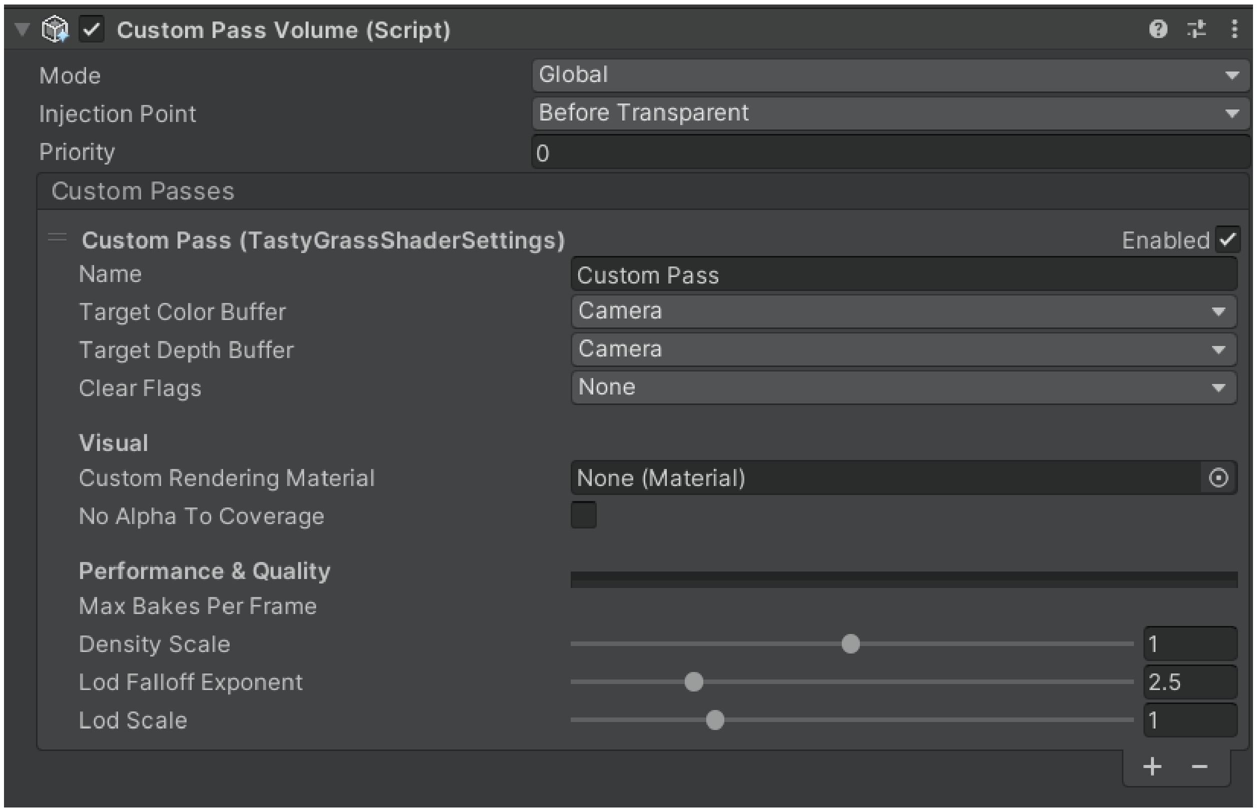
Density Scale: 0.5

Lod Falloff Exponent: 2.5

Lod Scale: 1

Adding Global Quality Settings (HDRP) (BETA)

1. Add or locate a Custom Pass Volume. (The Injection Point doesn't matter.)
2. Add a `Tasty Grass Shader Settings` pass.
3. Configure the settings to your liking.



Custom Pass Volume (Script)

Mode	Global
Injection Point	Before Transparent
Priority	0

Custom Passes

Custom Pass (TastyGrassShaderSettings) Enabled ✓

Name	Custom Pass
Target Color Buffer	Camera
Target Depth Buffer	Camera
Clear Flags	None

Visual

Custom Rendering Material	None (Material)
No Alpha To Coverage	☐

Performance & Quality

Max Bakes Per Frame	
Density Scale	1
Lod Falloff Exponent	2.5
Lod Scale	1

Control settings via Code

Alternatively, you can also change the global settings via code.

Note

You still need the `Tasty Grass Shader Global Settings` to be added to the `Universal Renderer Data` as shown above.

Warning

Currently, this modifies the active Render Pipeline asset. A better API for controlling the settings might be added on demand.

```
//...
TastyGrassShaderGlobalSettings settings = TastyGrassShaderGlobalSettings.lastActiveInstance;
if(settings != null){
    settings.densityScale = 0.5f;
}
//...
```

□ For Programmers

Implementing a load screen (New in 2.0)

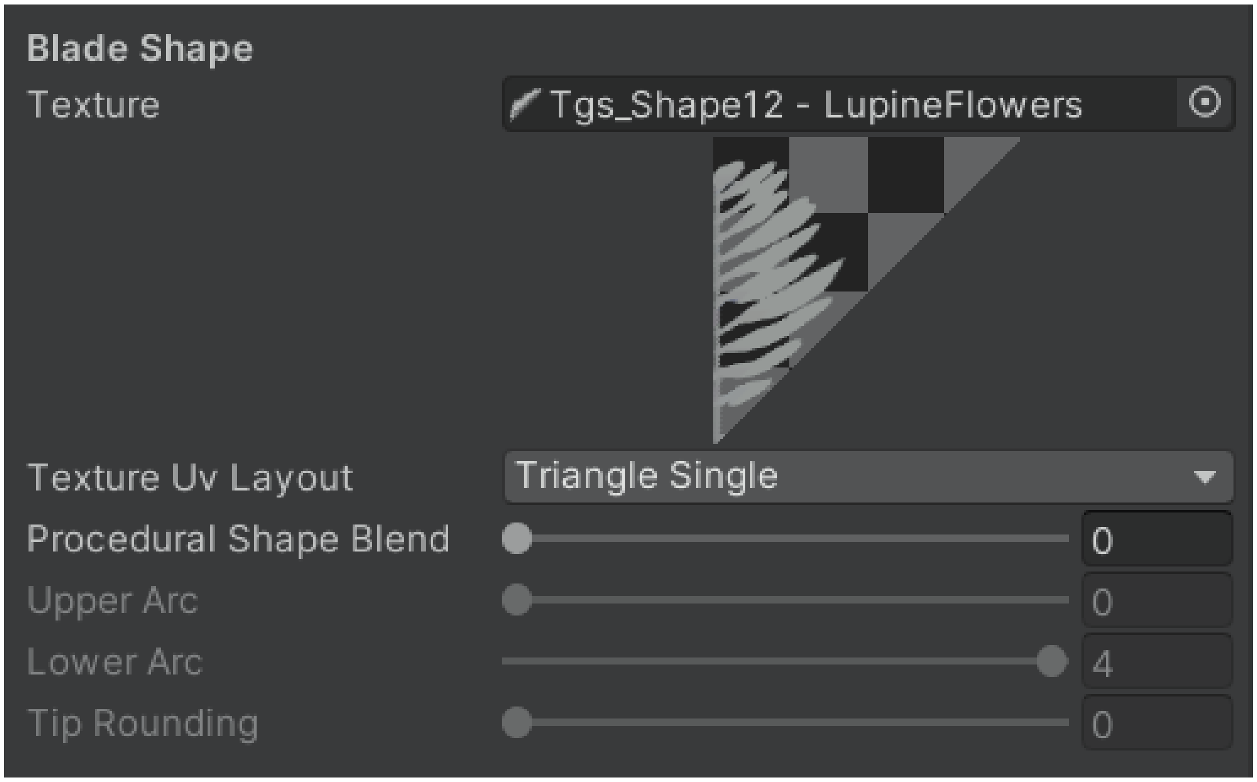
When a level is loaded, Tasty Grass Shader may not finish baking all chunks yet, resulting in unpleasant pop-ins.

To know for how long you need to display a load screen, you can use the `TgsGlobalStatus` class. Use the `areAllInstancesReady` field as an indicator if the scene is ready to be presented yet.

□ For 2D-Artists

Creating Textures (New in 2.0)

Textures in Tasty Grass Shader may only use a triangle shaped section of the triangle and not the full texture. For reference, it is best to look at the provided textures found in ”Assets/SymmetryBreakStudio/Tasty-GrassShader/Textures/Shapes”. Alternatively, textures that were already created using the full texture size, may use the `Triangle Center` layout in the Preset, which uses an alternative shapes, intended for such cases.


Blade Shape
Texture: Tgs_Shape12 - LupineFlowers
Texture Uv Layout: Triangle Single
Procedural Shape Blend: 0
Upper Arc: 0
Lower Arc: 4
Tip Rounding: 0

Glossary

Preset (`TgsPreset.cs`)

A Scriptable Object, containing all settings required to bake grass. It is rather advanced and intended for Technical Artists.

Layers

A layer refers to an automatically-managed instance of Tasty Grass Shader, defined by Settings and Preset and Terrain/Mesh specific options.

Instance (`TgsInstance.cs`)

An instance refers to a single mesh of grass, controlled by a preset. Technically, the ”Tasty Grass Shader for Terrain” and ”Tasty Grass Shader for Mesh” are wrappers over one or multiple instances.

Chunk

In the case of Tasty Grass Shader, a chunk describes a square-shaped part of a Unity Terrain. Diving the terrain into chunks is necessary to reach acceptable performance levels. A small chunk size is usually better for the GPU, because the level-of-detail can be adjusted more granular. However, it increases the amount of CPU overhead. For the best performance, a careful balance must be chosen.

Baking (`TgsInstance.cs`, `TastyGrassShaderCompute.compute`)

Describes the process of placing grass on a given mesh (or terrain chunk) and storing it in a GPU-Buffer using a preset. This process is rather computationally heavy, so it is executed as few times as possible. It is executed on the GPU as an compute-shader, found in `TastyGrassShaderCompute.compute`.

Making Your Own Presets

How Tasty Grass Shader Works

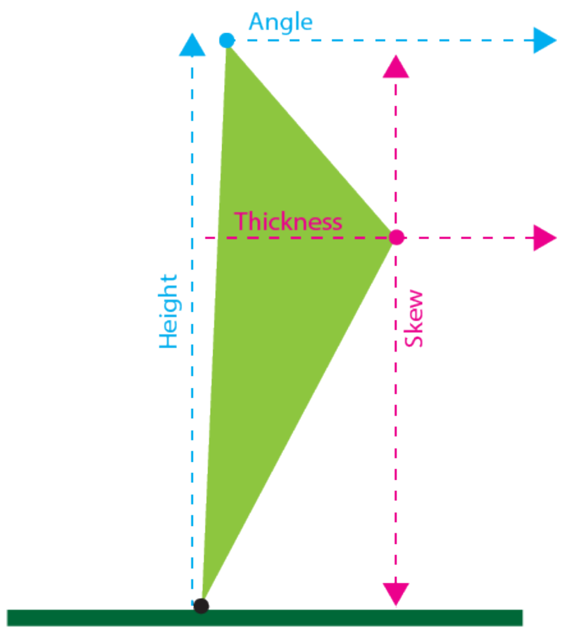
> Tasty Grass Shader can be seen as a *glorified geometry spamer*. Given a mesh or terrain, Tasty Grass Shader will procedually generate thoundsands of triangles on top of it. With the help of textures and coloring, these triangles create the ilussion of grass, dandelions or other clutter.

Note

The choice of triangles over quads or other shapes is for both performance and aestetical reasons. Triangles are the most minimal way to represent a surface, making it suitable for usages in high quantities. Also, they only have a singular intersection point with the ground, with avoids issues like grass growing out of the air, reducing head-ache for artists.

Each blade (or triangle) is internally defined by these properties:

- Height: How tall the blade grows.
- Angle: How wide the blade reaches to the side.
- Thickness: How thick a blade is.
- Skew: How skewed the blade is.
- Offset: How much the blade is lifted of the ground.
- Color: The color of the blade.



To add variation, these values are controlled via a texture is used. This texture is called the **noise texture**. For each grass blade that is spawned, the noise texture is sampled at the location of the blade. The texture value is then used to control these properties. For better control, each of the properties is **remapped** from the texture value of 0->1 to Min->Max. This package of noise texture and remapping ranges is called a **noise layer**, of which there are four in total.

After the blade has been generated, various effects can be applied, such as twirl or flatten. For many things, twirl is useful to add perceived volume to the grass.

We recommend copying existing presets as starting points, and from there on to play around with different values to get a feeling for the tool.

Change the Lighting Model / Customize the Rendering Material (new in v2.1)

If you have specific stylistic or other rendering requirements for your project, you might want to replace the rendering material/shader.

Create a custom shader for Tasty Grass Shader

Via Amplify Shader Editor

1. Duplicate the template/reference shader file found in "Assets/SymmetryBreakStudio/Tasty-GrassShader/Shaders/AmplifyShaderEditor/TgsCustomAmplifyShaderExample.shader" into a folder of your own.
2. Create a material from your new shader.
3. Open the Universal Rendering Data. You may need to add a "Tasty Grass Shader Global Settings" rendering feature if not present.
4. Set the `Custom Rendering Material` to the new material you have created.

Via Shader Code This is more advanced, but useful if you want total control over the shader. There isn't a clear path from here, but there are a few notes and starting points:

- `TastyGrassShaderCommon.hlsl` is the rendering pipeline independent core. It contains all functions required to handle the data that is created from the Tasty Grass Shader, as well as all variable names that are used by the Tasty Grass Shader rendering system.
- `TastyGrassShaderURP.shader` is a great starting point, as it only contains code from Tasty Grass Shader that is used for the Universal Rendering Pipeline. This also a great example to see how to use the functions in `TastyGrassShaderCommon.hlsl` are used in practice.

Use The Low Level API

You can use the Tasty Grass Shader API directly, without the built-in components. This is usefully if you want to use TGS on geometry, that is not directly exposed via Unity Components (`MeshFilter`, `MeshRenderer` or `Terrain`), such as for the Unity ECS system or a custom level creation pipeline.

Internally, there are very few dependencies on Unity classes. If you want to bake for the Unity Terrain for example, the low level API doesn't even know what that is, it just takes a `Texture2D` and a few coordinates as an argument.

Minimal Example For Mesh

```
using System;
using SymmetryBreakStudio.TastyGrassShader;
using UnityEngine;

/// <summary>
/// Minimal example of a custom TgsInstance.
/// </summary>
[ExecuteInEditMode, RequireComponent(typeof(MeshFilter), typeof(MeshRenderer))] // This is not te
public class TgsCustomInstanceExample : MonoBehaviour
{
    /// <summary>
    /// Instances represent a "slot" for a individual block of grass.
    /// You only need to create an instance once, it is intended to be "recyclable" for minimal G
    /// </summary>
    TgsInstance _instance = new TgsInstance();

    /// <summary>
    /// Settings contain all information of *what* kind of grass to grow.
    /// </summary>
    public TgsPreset.Settings settings = TgsPreset.Settings.GetDefault();

    /// <summary>
    /// The wind settings to use. Since they are often shared across the entire scene, they are s
    /// </summary>
    public TgsWindSettings windSettings;

    private void OnEnable()
    {
        _instance.Hide = false;
        BakeInstance();
    }

    private void OnValidate()
    {
        // For editor only.
        BakeInstance();
    }

    void BakeInstance()
    {
        if (settings.preset == null)
        {
            // We must have a preset defined.
            return;
        }

        if (windSettings == null)
        {
            // We must have a wind setting.
            return;
        }

        // The mesh that is used to grow grass on.
        Mesh sharedMesh = GetComponent<MeshFilter>().sharedMesh;
        // This is the matrix that is used to transform the mesh vertices into world-space.
        Matrix4x4 localToWorldMatrix = transform.localToWorldMatrix;
        // The bounding box that is used to frustum cull and also store the grass.
        // (Grass blades are stored relative to the bounding box of the mesh for smaller memory f
        Bounds boundingBox = GetComponent<MeshRenderer>().bounds;

        // Create the recipe, a container for *what* kind of grass to grow and *where* (Mesh, chu
        TgsInstance.TgsInstanceRecipe recipe =
            TgsInstance.TgsInstanceRecipe.BakeFromMesh(localToWorldMatrix, settings, sharedMesh,

        // Apply the recipe.
        _instance.SetBakeParameters(recipe);

        // Mark the geometry of this instance "dirty". This will tell TGS that something about th
        // either something about the grass it self, or something about the geometry that the gra
        _instance.MarkGeometryDirty();

        // Don't forget to setup a Wind Setting, or the grass won't show up.
        _instance.UsedWindSettings = windSettings;

        // Marks the "look" of the grass dirty.
        // This is used when only indirect aspects of the grass have changed, such as the wind.
        _instance.MarkMaterialDirty();
    }

    private void OnDisable()
    {
        // We can also hide instances. This will make them invisible, but still keep them alive.
        _instance.Hide = true;
    }

    private void OnDestroy()
    {
        // Don't forget to release the instance, or it will leak internally!
        _instance.Release();
    }
}
```

FAQ

How does Tasty Grass Shader work?

Tasty Grass Shader can be seen as a *glorified triangle spammer*. Given a mesh or terrain, Tasty Grass Shader will procedurally generate thousands of triangles on top of it. With the help of textures and coloring, these triangles create the illusion of grass, dandelions or other clutter.

Tasty Grass Shader strictly separates creating the grass from rendering it. This gives a lot of room to increase visual richness, as users don't need to worry about how procedural parameters affecting the performance. Everything expensive is done once at loading and then baked into a single mesh. At runtime, only effects such as wind are applied per-frame. Only the amount of grass and its size will affect performance.

How good is the performance?

A list of tested devices can be found here.

How is Tasty Grass Shader (TGS) different from other grass shaders?

TL,DR:

Compared to solutions that rely on **geometry shaders** (meaning that the geometry is generated on-the-fly with a special GPU feature), Tasty Grass Shader benefits from this separation performance wise. (Not to mention that geometry shaders have a upfront performance penalty and are considered legacy by most GPU manufactures and are only kept around for backwards-compatibility reasons.)

Compared to solutions that rely on **GPU-Instancing** (that means repeating one or multiple meshes many times), Tasty Grass Shader benefits from its procedural approach, allowing for more visual variation. Using the mesh approach may create issues with steep terrain, as these meshes may clip through geometry.

Longer Answer:

Tasty Grass Shader is procedural-driven, each grass blade is virtually unique, while other similar assets may use prefabs and pre-made meshes. That approach might be easier to understand, but creates CPU overhead for Unity. With TGS, it's just a single draw call per mesh/chunk + preset.

We optimized for rendering lots of grass. For example, each blade is just a single triangle instead of a quad. (This also has the nice side effect that you never see TGS grass clip into the ground, since there is just a single point of intersection with it). To minimize bandwidth and GPU-Ram usage, the grass is also stored in a custom format, which reduced the size by 600%(!) compared to the full format. (Our format uses 16 bytes for triangle/blade. A Unity mesh would be 108 bytes for a triangle, given: position + normal + color = 36 bytes * 3 vertices = 108 bytes per triangle.)

Finally, with TGS you are not just limited to grass, you can also generate flowers or ground clutter. In fact, TGS might be even called a "glorified triangle spammer", so everything that needs a lot of individual triangles can be built with it, to some extent. Not to mention that we have a tool for placing grass on terrain and support vertex colors + PolyBrush for meshes.

How does Tasty Grass Shader integrate into Unity?

TGS is added to a Unity Terrain or a Mesh by a single Component. No additional setup is required.

For global settings, a render feature may be added to the current render pipeline, which exposes settings such as LOD factor or global density.

Can you add feature X?

We would be happy to hear from you about features you are missing. After all, we can not imagine all possible use cases or how important they are for you. Please get in touch with us via Discord, email, or create an issue here.

Intro

To test the expected performance of TGS, we sampled the average frame rate over ~5 seconds (after all baking was done) on the included demo scene:

All devices were tested with the following settings:

- Resolution: 1920x1080
- Global Density: 0.5
- LOD Scale: 1.0
- LOD Falloff: 2.5
- No MSAA, No SSAO.

Results

Device	Total Avg. FPS	TGS Version	GPU	API	CPU	OS
Google Pixel 6a	**36 FPS**	2.1.3.1 (23.09.2024)	Mali-G78 MP20	Vulkan	Google Tensor	An
Asus Rog Strix G16	**236 FPS**	2.1.3.1 (23.09.2024)	Nvidia RTX 4070 (Laptop)	DirectX 11	Intel i9-13980HX	W

This is an auto-generated listing of all tooltips used within the asset.

Tgs For Mesh

- `Wind Settings`: Wind setting used for this object.

Tgs For Unity Terrain

- `Wind Settings`: Wind setting used for this object.

Debugging

- `Show Chunks Bounds`: Shows the bounds of all terrain chunks in editor.

Settings

- `Preset`: The preset used. Must not be null.

Common

- `Amount`: Final scaling factor for the amount of the grass blades.
- `Height`: Additional scaling factor for the height of the grass blades.
- `Thickness`: Additional scaling factor for the thickness of the grass blades.
- `Angle`: Additional scaling factor for the thickness of the grass blades.
- `Tiling`: The additional tiling factor for noise textures of the grass.
- `Slope Bias`: Additional bias for the maximal allowed slope/angle in degree.

Color

- `Tint`: The final color tint for any grass blades.
- `Hue Shift`: Shifts the hue for all colors. Useful to alter the look.
- `Saturation`: Controls the saturation of the grass. Useful to alter the look.
- `Camouflage`: How much the color of the ground should be adapted. Currently only works with terrain.

Tasty Grass Shader Global Settings

Visual

- `Custom Rendering Material`: Optional custom material for rendering. If None/Null the default internal material will be used. Helpful, when using custom lighting models or other assets/effects that affect the global rendering are used. See the TgsAmplify
- `No Alpha To Coverage`: Fixes alpha issues with XR by disabling alpha to coverage and using simple alpha clipping instead. Note that this prevents MSAA from working with the grass. May only work with the default TGS shader/customRenderingMaterial is set to null.

Performance & Quality

- `Max Bakes Per Frame`: The maximum amount of instances that are baked per frame.
- `Density Scale`: Global multiplication for the amount value.
- `Lod Falloff Exponent`: The exponent for the internal LOD factor. Higher values will reduce the amount of blades visible at distance. This can be used to improve performance.
- `Lod Scale`: Global multiplication for of the LOD.

Noise Setting

- `Disable`: If set to true, this layer will not be applied.
- `Solo`: Isolate this layer.
- `Texture`: The noise texture used. Must not be null.
- `Tiling`: Tiling factor of the noise texture. Larger values will lead to more repetition. Note that the final values gets divided by 100.
- `Value Scale`: Scale applied to the raw noise value texture, before being passed to the modifiers. Practically the same as a contrast modifier in image editing.
- `Value Offset`: Offset applied to the raw noise value texture, before being passed to the modifiers. Practically the same as a brightness modifier in image editing.

Modifiers

- `Height`: The height of a grass blade. Negative values will internally be clamped to 0.0, so you can have negative numbers here for masking.
- `Offset`: How far the blade should be created from the geometry. Negative values will internally be clamped to > 0.0, so you can have negative numbers here for masking. You can use this settings to get experimental, for example to create simple flowers (see the flowers preset for that).
- `Angle`: The side-extending width of a blade. Negative values will internally be clamped to 0.0, so you can have negative numbers here for masking.
- `Thickness`: The thickness of a blade. Negative values will lead to inwards bended blades, while positive to outwards bended.
- `Skew`: The apex or mid-point of the thickness.
- `Color Influence`: How much the color property will blend in color to the blade. Negative values will internally be clamped to 0.0, so you can have negative numbers here for masking.
- `Color`: The color to blend in on top of the grass blade, controlled by Color Influence.

Tgs Preset

Grow Settings

- `Density`: How much grass to create approximately per square meter.
- `Angle Limit`: Maximum allowed angle between normal and grow direction.
- `Grow Direction`: The growing direction of the grass. Other values than (0, 1, 0) can be used to create ground clutter.
- `Grow Direction By Surface Normal`: Whether to grow in the upwards direction (at 0.0) or the surface normals (at 1.0).
- `Minimal Height`: If a blade is smaller than this threshold, it will be removed.

Clumping

- `Clump Count`: How many blades should be created at the same position. Note that blades can't be created across multiple triangles.
- `Clump Percentage`: How strong the clumping is.
- `Clump Limit Noise Variation`: If active, noise layers will used the clumped position. This is usefully for objects that are formed from multiple blades, such as some flowers.

Stem

- `Stem Generate`: Generates a stem. Useful for flowers or other plants. Will create one stem per clump.
- `Stem Color`: The color of the stem.
- `Stem Thickness Ratio`: The thickness of the stem in relation to the blade thickness.

Detail Layers

- `Noise Texture 0`: The noise texture used for Layer 0.

Detail Layers

- `Noise Layer 0`: The settings used for Layer 0.
- `Noise Texture 1`: The noise texture used for Layer 1.
- `Noise Layer 1`: The settings used for Layer 1.
- `Noise Texture 2`: The noise texture used for Layer 2.
- `Noise Layer 2`: The settings used for Layer 2.
- `Noise Texture 3`: The noise texture used for Layer 3.
- `Noise Layer 3`: The settings used for Layer 3.

Effects

- `Twirl`: How much the blade is twisted around its center. Even small values greatly enhance the voluminousness of a grass field.
- `Flatten`: How much the blade vertices are flattened, based on the ground normal. Can be used to create ground clutter.
- `Scruffiness`: Randomization of growing direction. Also affects shading normals.
- `Limpness`: How much the tip should be moved back to the ground. Moderate values are good to create ground clutter.

Blade Shape

- `Texture`: The optional texture to use.
- `Texture Uv Layout`: What UV layout to use.
- `Procedural Shape Blend`: Controls the mix between the procedural shape or the texture.

- `Upper Arc`: The strength of the upper arc. Lower values lead to stronger arcs.
- `Lower Arc`: The strength of the lower arc. Higher values lead to stronger arcs.
- `Tip Rounding`: Values greater than 0 will introduce a more rounded shape. Helpful for stylization.

Wind

- `Wind Intensity Scale`: Scaling factor for the current wind settings. A value of 0 will disable wind interaction entirely, which is useful for ground clutter.
- `Wind Pivot Offset`: The offset of the pivot along the growing direction, used for rotating the grass blades in the wind. If you create things like flowers that have height offset applied to them, use this value to make the rotation look correct again.

Shading

- `Smoothness`: The PBR smoothness used for shading.
- `Occlusion By Height`: Strength of the occlusion factor that is guessed by the height of the grass blade.
- `Collider Influence`: How strongly the blades should be pushed away by colliders.

Runtime Quality

- `Base Lod Factor`: The base LOD height for this bake setting. For settings that create smaller blades, also use smaller values here.
- `Cast Shadows`: Whether to cast shadows.

Tgs Wind Settings

- `Direction`: The direction of the wind in degrees.
- `Strength`: The strength of the wind.
- `Patch Size`: The size of a wind patch. Smaller values may create more believable settings.
- `Speed`: The speed of how fast the wind patches move.

Tgs Collider

- `Radius`: The radius of the collider.