Critter Environment 1.1

Documentation

Critter Entertainment

Contents

1. Introduction

Thank you for buying this asset! The *Critter Environment* contains tools for instancing foliage onto meshes and terrains, and three sub-shaders for Unity's URP Shader Graph: pixel outlines, natural cloud shadows, and customizable cell-shading effects. These are combined into an example shader that is provided with the asset, for objects, terrains and instanced meshes. You can use these to create your own more specific shaders or modify and use the existing ones.

These tools were originally designed for the 3D Pixel Art style, but many of the components work for many other styles as well. As the pixel outlines are 1 pixel wide, they work best with this art style.

We would love to hear your feature requests, feedback, and bug reports preferably on Discord or via email.

Email – critter.entertainment@gmail.com

Discord – https://discord.gg/bmzP8pG6en

1.1. Setup tutorial (complete this first)

Setup for instancing

1. Go to scripts folder and drag *MeshInstancingBehaviour.cs* or *TerrainInstancingBehaviour.cs* to an object that contains either a mesh or is terrain.
2. Set the parameters for the scripts. You can check the demo scene for reference.
3. **Done!** Now these scripts will spawn the defined instances on top of the mesh when you play.

Setup required for water and outlines to work

1. Navigate to your renderer
2. Click "*Add Renderer Feature*"
3. Select "*Pixel Outline Setup Feature*"
4. **Done!**

2. Instancing

2.1. User Guide

The instancing tools are meant for rendering meshes on objects in a performance friendly way. You can use these tools through the scripts *MeshInstancingBehaviour* and *TerrainInstancingBehaviour* or create your own implementation from *InstancingBehaviour*. In these scripts there are the following inputs:

Density
Density of instanced meshes.

Configurations
The configurations define an array of meshes and materials that are instanced. For example, a grass field can have: grass and flowers. In such case you might want to make grass larger and flowers rarer.

*NOTE: Remember to **set the scale to at least 1** manually. The scale changes the meshes size on all axis.*

Probability defines the chance of this configuration being chosen for a certain point. For example a grass with value 100 has a 100/101 chance compared to a flower with probability 1 and the flower a 1/101.

The distance offset pushes the points in the direction of their calculated normal, multiplied by its value. This can be useful, for example, for creating more fluffy trees by having the leaves off the mesh.

MeshInstancingBehaviour - Sub Mesh Details
The mesh can limit the points calculated to only cover a certain sub mesh. To only render points on a single sub mesh, enable the radio button and type index of the sub mesh. You can check which sub mesh has which material in the Mesh Renderer under Materials for reference.

TerrainInstancingBehaviour – Configuration Layers

Terrains have control textures that enable the painting of different textures on it. In the example terrain shader and instancing tool, we use 4 channels of one texture to provide the ability of rendering certain elements only on these parts of the terrain. For example, grass could be rendered on a grassy texture and tumbleweed on a desert texture. For example, In the editor you can use the First Layer input to define what gets instanced on the red channel of the control texture or first texture in the terrain editor.
(Control texture Color channels RGBA <-> Layers 1234)

TerrainInstancingBehaviour – Variance Parameters
The points on the terrain are calculated first evenly but inside their square they get a random offset depending on this parameter. In addition, the scale of these elements can be randomized as well.

2.2. How it works

The script first calculates points based on some input and logic defined for its use case. Second, the Instance configurations are defined for these points. Third, the data is moved to the GPU and finally the draw call is called in the update loop of InstancingBehaviour. The behaviour is enabled and disabled with Unity's OnEnable and OnDisable calls. You can try different input values at run time by changing input values and toggling the script on and off.

2.3. Extending the Implementation

InstancingBehaviour
This class handles the logic of interacting with Unity's instancing API with the use of our custom Instancing class. The behaviour of enabling and disabling can be found and changed here.

The InstancingBehaviour is an abstract class that needs the implementation of a function that provides it with InstancingData. You can extend a custom class where you want custom logic for calculating points. Use the existing ones as reference points if needed.

InstancingData
The instancing data is the result of the logic that calculates the points where things are instanced. The InstancingData should match the struct found on the GPU but can be changed if you need to pass some other data to the shaders.

Shader Graph Support

Shaders that are used for instanced objects need some setup especially in Shader Graph. An example of this can be found in our InstancedToon Shader. The setup provides access to the instances ID and with this it can access the InstanceData in the GPU memory. With this data it updates the object to world and its inverse matrices so the shader can draw the objects in correct positions. To create custom shaders for instanced objects you should use the ShaderGraphInstancingSetup at the beginning of the vertex shader.

Create Prefabs

This method of instancing is especially handy for creating prefabs of assets you like. For example, you could download 3d modeled trees off the asset store and apply the mesh script on them to create a more unique looking prefab and use it in your worlds.

3. Shaders

3.1. Clouds

The clouds use 3D Simplex noise to create a cloud-like texture. The third dimension allows us to move through the third space to create more natural illusion of cloud shadows changing over time as they would in real life. The cloud shadows are sampled with UV coordinates which usually are from the X and Z axis in Unity (as the Y-axis points up).

Inputs (6)

- *Cloud Density* - Scales the noise texture to allow different sized worlds.
- *Cloud Movement* - Defines the speed at which the clouds move over the world.
- *Cloud Strength* - Adjusts how “thick” or strong the clouds are.
- *Cloud Cover* - Defines how large of an area the clouds cover.
- *Cloud Change* - Define the speed it moves through the third noise axis.
- *Cloud Step* - Adjusts how the noise layers on top of each other.

This can be combined with your shaders by multiplying the main light's strength.

3.2. Toon Ramp

The toon ramp divides a linear value into distinct cells. This sub shader has three different values to control how the division is done.

Inputs (3)

- *Shades* - Defines into how many distinct cells the values are output into.
- *Brightness* - Adds additional light. Makes objects brighter.
- *MinimumDarkness* - This value limits the minimum values the ramp will output.

In our experience a value for '*Shades*' ranging between 4-8 have looked good. Play around with it to find the values that suit your art style. The Lighting value is the value that gets divided (this should be a value between 0 and 1). You can check the example or demo shaders for how it can be used. Usually a strength value from the main lighting source which is calculated by its normal and the main light direction.

3.3. Pixel Outlines

To see pixel outlines you need to add the Renderer Feature for it (done in the setup guide). The pixel outlines are intended for low resolution 3D rendering, for example, by using with the *Critter3DPixelCamera* asset. This ensures that the outlines on objects are one-pixel thick which results in a stylistic look. This is done by calculating outlines from differences on the depth texture and normal texture. When a large enough difference is found above a threshold, it outputs an outline value, otherwise it outputs 0.

One pixel thick outlines can be found by sampling the texture 1 pixel away from the analyzed point. For outlines from the normal texture, you need to choose a direction into which the outlines are chosen.

Inputs (5)

- *DepthThreshold* - defines the difference needed to show an outline based on depth. *Normal Threshold* - defines the difference needed to show an outline based on the distance of 2 normals.
- *Normal Threshold* - defines the difference needed to show an outline based on the distance of 2 normals.
- *NormalBias* - the direction into which the normal outlines are preferred.

- *DepthEdgeStrength* - The output value when a depth outline is found.
- *NormalEdgeStrength* - The output value when a normal outline is found.

We recommend keeping Normal Bias at (1, 1, 1) and only changing the thresholds from the defaults if you don't see outlines. Check that your textures are enabled first. For output, you should play with the Strength values. Usually, a smaller normal value looks better and a stronger depth outline respectively.

Note
If you want to manually enable the needed textures, you can find the "Pixel Outlines_Renderer" render feature directly under the root folder. If you have issues getting outlines to render, you can use the pixel outlines renderer feature provided, by adding that to your pipeline asset's renderer list and making sure your camera uses that renderer.

3.4. Toon Shader

The example toon shader combines these shaders with a custom way of calculating lighting. It's provided to provide a reference on how these sub shaders could be utilized. You can try it out and see if it works for your project or modify it.

Here's a more in-depth description how I used the sub graphs in the example:

- The Outlines and Clouds are possible to be disabled through shader properties. This was done by using Shader Graphs Boolean Keywords.
- The outlines are used by blending the light-calculated color with an outline color. This blend opacity is defined by the output value of the PixelOutline node.

- The clouds are used by inverting the values and multiplying the attenuation of the main light. This creates the illusion of the clouds blocking the main light depending on how "thick" they are.
- This attenuation value is put through the Toon Ramp which creates the stylized cell shaded look. This can't be disabled as it is a toon shader.
- Unique to this shader it has 2 colors between which the color is interpolated between with the ramped values. In addition to this, additional lights affect the ramping of colors.

4. Water

The water shader provides a way to create stylized water on surfaces when used with orthographic camera. There is a water prefab that can be dragged into a scene. This requires the setup Render Feature to be on.

The reflection camera has some overhead and can impact performance you can disable it but disabling its game object and setting the boolean property off on the shader. In game mode, this reflection script creates an instance of the material so if you want to try parameters be sure to turn it off or edit the material instance instead of the main material.

4.1. Foam

The foam gives outlines to the water and are animated with voronoi noise. This can be controlled with parameters set in the shader.

Inputs (4)

- *FoamAmount* - how far the foam comes from the edges of the water.
- *FoamScale* – the scale of the Voronoi noise impacting the foam
- *FoamSpeed* – how fast time affects the noise texture sampling
- *ParticleThreshold* – how many and large particles of Voronoi noise is layered on

4.2. Tint and Opacity

Inputs (4)

- *TintStrength* – how transparent the water is or how strong the tint is.
- *Bands* – how many distinct bands the depth is split into
- TintBrightness – height where the tint starts to be bright
- *MinimumTint* – minimum value for tint not dependant on depth

4.3. Distortion

Inputs (5)

- *DistortionTexture* – texture that defines how the distortions form

- *DistortionTiling* – how zoomed into the texture the sampling is
- DistortionDirection – which direction the distortion moves from the sample
- *DistortionThreshold – threshold to discard that distortions do not sample above water*
- DistortionStrength – how strong the effect is into each direction

4.4. Reflections

Inputs (2)

- *ReflectionStrength* – how mirror-like the reflection is
- *ReflectionDistortion* – how strong the reflections are affected by distortion

5. Additional material and credits

Thanks to Cyanilux and his great material on URP Custom Lighting.
Thanks to Catlike Coding for great resources on water shaders.
Thanks to Unity for providing built-in shaders with good reference on how to instance with Shader Graph

Dev-log on creating 3D Pixel Art tools
Cyanilux's github on custom lighting in URP
Catlike Coding's website

6. Final Words

Thank you for purchasing this asset! Please contact us if you have any questions or feedback, as we would love to help you bring your vision to life. We would also greatly appreciate it if you submitted a review on the asset store as it helps us to improve the asset. Hope you achieve the look in your game you were looking for!