Critter 3D Pixel Camera 2.3

Documentation

Critter Entertainment

Contents

1. Introduction

Thank you for buying this asset! It is designed for 3D pixel art games that have an orthographic camera. This asset pixelates your game in a consistent manner and makes it look smooth when moving the camera.

Things in this asset are optimized so that calculations are cheap, and the game is rendered on low-resolution to boost performance.

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

2. Step-by-step setup guide

Setup instructions. Once you have imported the asset:

1. Unpackage the “BuiltIn” or “URP” unitypackage, depending on your project’s render pipeline.
2. Delete the existing main camera.
3. Drag the PixelCameraSystem prefab into your project.

The demo scene contains an example if you want to reference a working solution.

3. Controlling the Camera System

This asset makes the camera match the position and rotation of another game object: *pixel camera transform*. To move or rotate your camera, move this *pixel camera transform* object instead. The asset then applies small adjustments on the camera directly for visual consistency, so it is strongly recommended not to manipulate the camera’s transform directly for these adjustments to work. For example, if you have a camera controller, you should now place it on the *pixel camera transform* object instead of the camera directly. Similarly, Cinemachine components like CinemachineBrain should be placed on this *pixel camera transform* object instead of the camera.

Other properties provided by this asset such as pixel resolution, game camera zoom, and view camera zoom are recommended to be changed from the *PixelCameraManager.cs* script. Note: it is not recommended to be constantly updating the resolution during runtime. This can become costly for performance.

4. Features

Pixel Perfect

Stationary objects in 3D look stationary without jittering colors or outlines. Pixels are consistent regardless of camera angle or movement (like in retro pixel art games). Ensured by locking movement to a grid relative to the camera.

Smoothness

Subpixel adjustments make the game look smooth (like in modern pixel art games). The view camera adjusts to match the true target position instead of the grid.

Performance

Game is rendered at a target resolution which has a significant boost on performance. It is then upscaled to the native resolution to allow subpixel movement, without compromising performance.

Versatile

Works in any camera rotation, uses Unity's transform position as input and every variable. Including resolution, can be changed at runtime.

Interface

Provides a straight-forward interface for changing rotation, position, resolution and two different types of zooms. At runtime.

5. Basics

This section goes more in depth about things to consider when using the asset. You can instead play around with the *PixelCameraManager.cs* component on the prefab once you have completed the step-by-step setup guide. This works in Editor as well as in Game mode.

Basics

The center of this asset is *PixelCameraManager.cs* script. This monobehaviour updates the other ones at correct timings.

The system uses Unity's Transform component on the camera as input for movement and rotations. Check the previous chapter for more specifics on controls.

The camera is locked to a grid relative to its view and to reduce blocky movement another camera smoothes the movement between the space in this grid. The adjustments are done in LateUpdate. This ensures that the adjustments and voxel grid positions are updated after the input on the target transform.

Pointer Events

Pointer events work by aligning your UpscaleDisplay's camera perfectly with your game's world camera. Therefore, in order for your pointer events to work intuitively, remember to configure your UpscaleDisplay camera's Culling Mask.

Please note that because this script aligns the UpscaleDisplay to your camera, its distance from world origin is now dependent on your camera's position. See section '*3.2. Things to consider'* and there the *Floating-point precision error* for more information about what issues this can potentially introduce.

Variables

PixelatedCamera.cs has few public variables you can change at any time. They are described in Table 1. below. Modifying these variables is how the select features are intended to be used.

The best way to understand what effect these variables have is to play around with them, as they are quite simple and intuitive. The calculations of this asset have little to no impact on performance.

Table 1. public variables

Type	**Name**	**Description**
Transform	Followed Transform	Transform that this camera follow while doing pixel perfect corrections. Very useful for camera controllers. Allows for full control of a transform.
bool	Voxel Grid Movement	Should the camera be locked to a grid in world units defined by the pixel size of the texture. Reduces jittering of static objects.
bool	Subpixel Adjustments	Should the view camera smoothen out the blocky movement caused from being locked on a grid?
bool	Follow Rotation	Should the camera match the followed transforms rotation.
ResolutionSynchronizationMode	Resolution Synchronization	If your display's aspect ratio changes suddenly, how should *Pixel Resolution* be calculated? Define both or scale them with aspect ratio and set one.
Vector2Int	Pixel Resolution	Your render resolution. This is later upscaled to fit the upscaled canvas and then the view camera renders it to screen. Lower values significantly improve performance.
bool	Control Game Zoom	Should this script control the orthographic size of the game camera. Can be turned off to give

		control to another controller like Cinemachine.
float	Game Camera Zoom	Changes the game cameras orthographic size. Zooming in shows more detail.
float	View Camera Zoom	Normal zoom effect. How much of the canvas does the view camera see. Value should be [0, 1] for normal behavior, but can be in the range [-1, 1].

6. Considerations

General tips for better performance in 3D pixel-art games

Having lower *Pixel Resolution* means your game is rendered in lower resolution. FX effects and post-processing can be especially costly for GPU at full resolution in bad hardware like mobile. Also, having a higher *Game Zoom* means your camera looks at a larger area of the world, which naturally requires more rendering and results in lower performance respectively.

Floating-point precision error

Because this asset performs precise calculations and adjustments on positions, it is advised to keep your world camera near world origin to avoid floating point precision error. In this context near means “not thousands of units away”.

Floating point precision error arises when dealing with so large floating numbers that the computer memory loses precision on them (not enough memory for decimals, because too many significant digits). In Unity and many other game engines, objects’ positions are defined by three floats (x, y, z). If your object is far away from world origin, your game-engine has hard time determining the **exact** (x, y, z) values deterministically. This error is highlighted in logic that requires precise numbers, like this asset.

Microsoft documentation - https://learn.microsoft.com/en-us/cpp/build/why-floating-point-numbers-may-lose-precision?view=msvc-170

Visual demonstration - https://www.reddit.com/r/Unity3D/comments/ijwfec/the_further_you_are_from_000_the_messier_stuff/?utm_source=share&utm_medium=web2x&context=3

7. FAQ

Why are the materials pink?

You are probably using wrong render pipeline's materials. If you have URP project, use files from the URP folder. If you have built-in render pipeline project, use files from the Built In Render Pipeline folder.

Can I use Cinemachine with this camera system?

With the 2.1 update you can use some features by attaching the controlling CinemachineBrain component to the followed transform. However, while rotating, the sampling is not consistent and might look jittery. To reduce this, we recommend rotating less often and for example not using the rotation part of Cinemachine's noise system. Here is an answer by Unity Technologies Gregoryl on how to create a custom noise profile to only use the position. (https://forum.unity.com/threads/freeze-the-z-translation-and-rotation-while-using-noise-module.520303/)

Can I lock moving other objects to not have jittery movement?

Yes, you can by using the VoxelGridAdjuster and setting another transform to be its target. However, unlike the camera that can be adjusted "subpixel" the moving objects on the grid can experience blocky movement.

How can I setup raycasts?

Ray casts can be simply set up by following the ray cast demo in the demo scene. The basics are to use the view camera as the source for the camera and the ray casts should end up in the correct positions. If you are culling objects on different layers, the view camera cannot hit them with ray casts.

The camera is jittering while rotating?

This is unavoidable to fix completely as it is impossible to constantly sample a rotating world but to minimize it, make sure you are disabling the smoothing and grid adjusting while your camera controller rotates. Check the example for reference.

8. Final Words

Thank you for buying this asset. I hope it helps you with your project and saves you from some headache. If so, please give it a good rating! If it doesn't help you or it's not what you thought it would be, then give it bad rating and provide us feedback to improve the asset.

- Developers

https://www.youtube.com/@JobeYT

https://www.youtube.com/@Voyageonyt