Pixel Camera Pro

A smooth pixel-perfect 2D camera system for Unity URP with per-layer resolutions and built-in parallax. Cinemachine compatible.

Contents

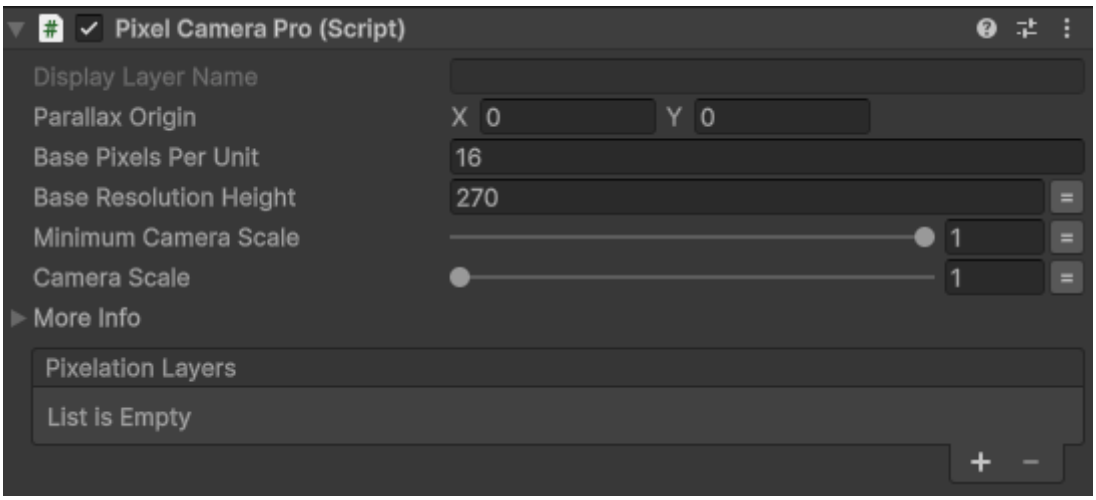
1. Overview

Pixel Camera Pro is a ***pixel-perfect*** camera system designed for 2D pixel art games. It delivers a smooth, ***jitter-free*** camera movement while allowing you to render different parts of your scenes at different resolutions. Each layer is pixelated on its own grid, making ***smooth parallax*** possible.

2. Setup Walkthrough

2.1. Create pixel camera

1. Add a camera to your scene, set the projection to orthographic, then add the `Pixel Camera Pro` component to the GameObject.
2. Set **Base Pixels Per Unit** to whatever PPU most of your sprites are imported at.
3. Set **Base Resolution Height** to the screen height your pixel art is designed for. Click the `=` button and pick from the recommended list. Every value there divides your current window height evenly, so it fills the screen with no letterboxing. Typing your own value works too, but the list guarantees this.

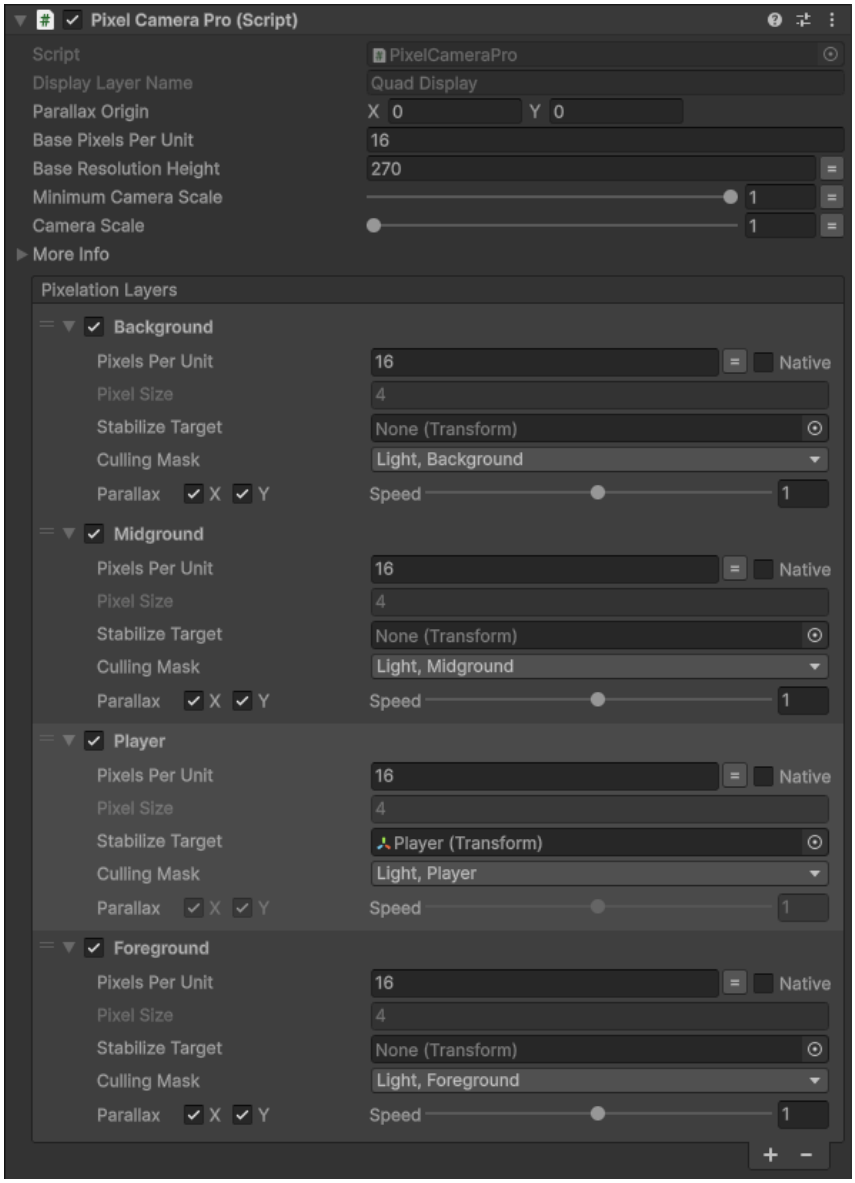


***Note**: Pixel Camera behaves like a normal camera. Supports camera stack and cinemachine*

2.2. Add Pixelation Layers

1. In the Pixelation Layers list, click `+`. This creates a pixelation layer. Double-click a layer's name to rename it.
2. Set the layer's **Culling Mask** to the Unity Layer(s) containing the objects you want to render. A pixelation layer's culling mask should not contain Quad Display layers.
3. Set **Pixels Per Unit** to the PPU you want for this layer. Click the `=` button to display all PPU values that produce square, whole-number pixels at the current window size. Or toggle **Native** instead when you want the layer to render at full screen resolution.
4. Repeat for each additional layer.
5. Layers are rendered from top to bottom in this list. Drag layers to change their render order.
6. You can click the triangle button to collapse a layer.

***Quick tip**: Give your lights their own Unity Layer, and add it to every Pixelation Layer's Culling Mask. Otherwise a light only affects the layers whose Culling Mask includes it.*



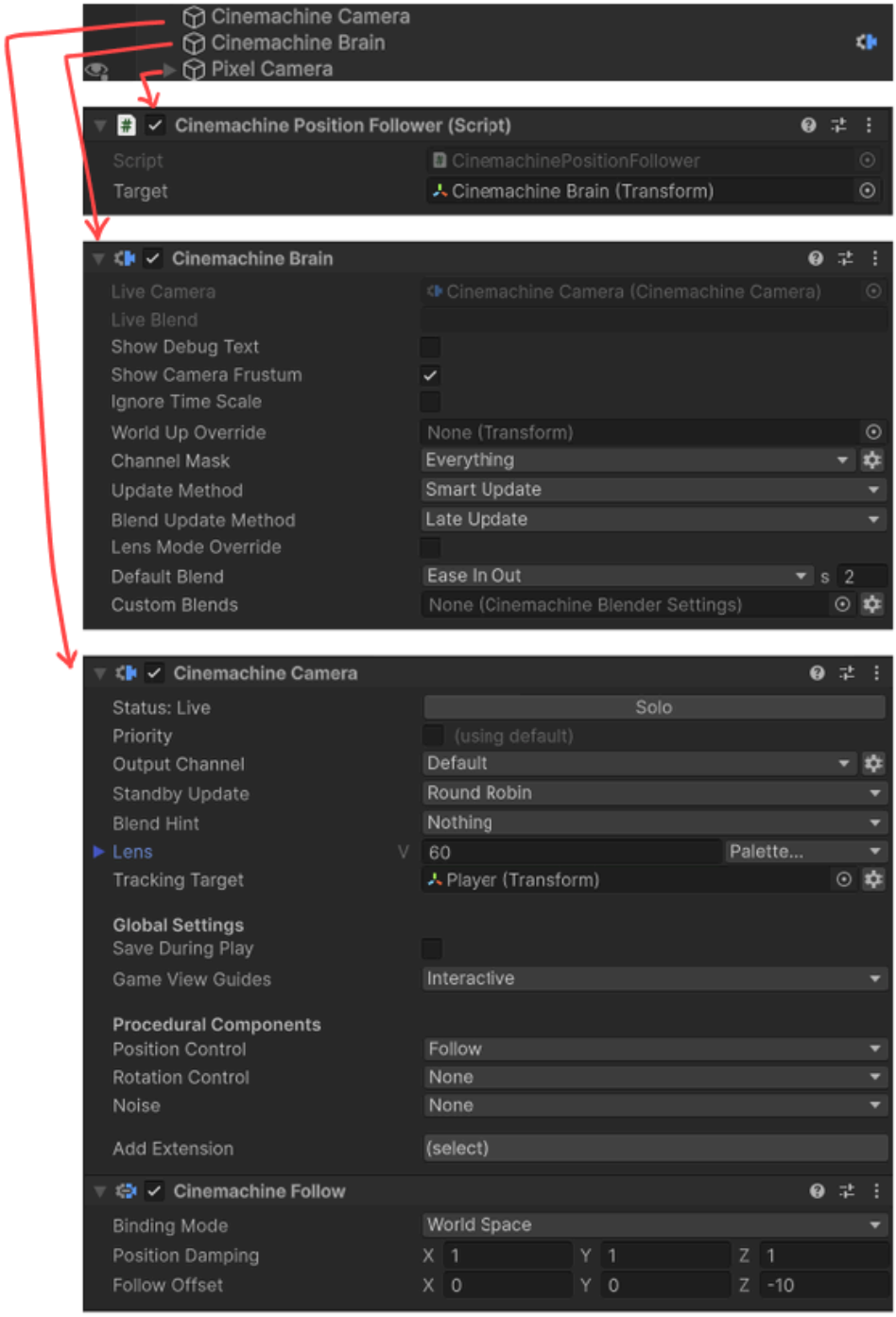
2.3. Set the camera's zoom

1. Set **Minimum Camera Scale** to the most zoomed-out view your game will ever use. Use its `=` button to pick a value every one of your layers stays clean at. It takes each layer's Pixels Per Unit into account and finds a scale that keeps pixels square and aligned to whole numbers across all layers.
2. Set **Camera Scale** to your camera's zoom level. 1 is the normal view, and higher numbers zoom in. This is also the value you'd change at runtime to zoom the camera in and out during play.

   You can adjust Camera Scale smoothly at runtime (for example, by lerping toward a target value) to create smooth zoom transitions. Pixels may look slightly uneven mid-transition. This is expected, and usually unnoticeable when the camera is changing quickly. Just make sure that once the zoom finishes, Camera Scale settles on a value that doesn't trigger a Pixel Size warning on any layer. A whole-number multiple of Minimum Camera Scale or an integer value (such as 1, 2, 3, etc.) is always guaranteed to work, but depending on your layers' Pixels Per Unit, other values may work too.

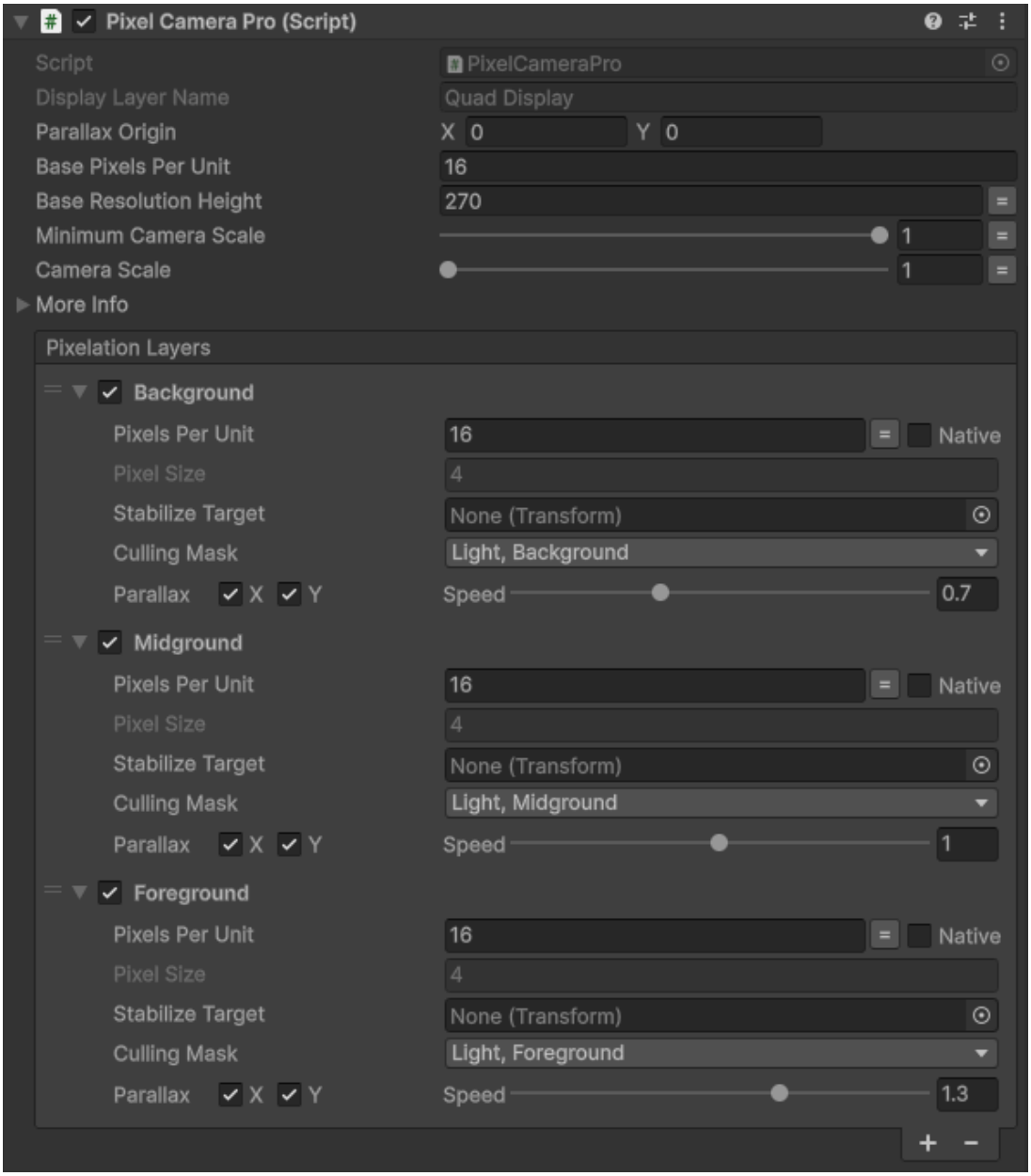
2.4. Set up Cinemachine Camera

1. Skip this part if you're not using Cinemachine Cameras.
2. Set up your Cinemachine Camera as usual, but place it on a separate empty GameObject from your Pixel Camera. The Pixel Camera should control the zoom, while Cinemachine handles the camera movement.
3. Control zoom through **Camera Scale** directly instead of your Cinemachine Camera's Lens.



2.5. Set up Parallax

1. Skip this part if your game doesn't have parallax.
2. Add a Pixelation Layer for each parallax layer you want to move at its own speed.
3. **Speed** controls how fast a layer moves relative to the camera. Below `1`, the layer moves slower than the camera and appears farther away. Above `1`, it moves faster and appears closer. at `1`, it tracks the camera normally.
4. Use **Parallax X** and **Parallax Y** to limit parallax to a single axis.
5. Set **Parallax Origin** (near the top of the Inspector) to wherever your parallax art should line up with the rest of the scene, usually your camera's starting position.



2.6. Snap your sprites to the grid

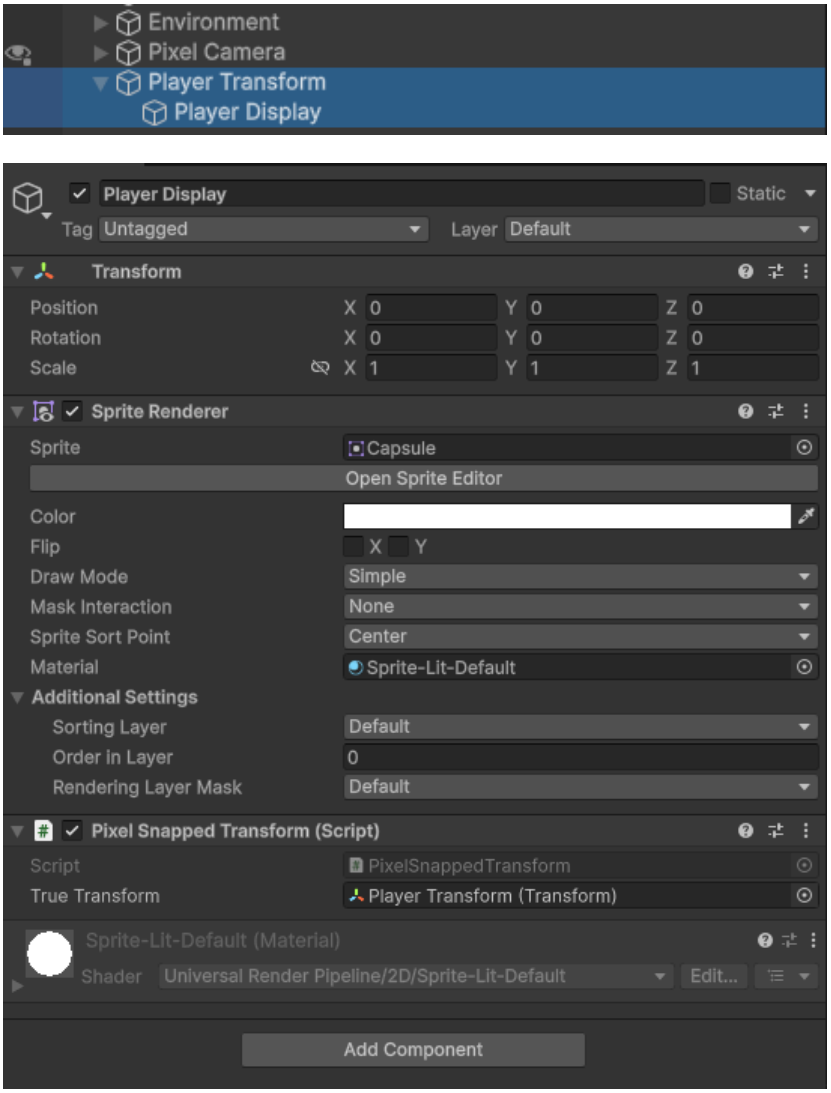
2.6.1. Static pixel art

1. Select the pixel-art sprites already placed in your scene (or their common parent), right-click, and choose **Pixel Camera Pro > Snap to Pixel Grid**.
2. Do this before entering Play Mode. An un-snapped pixel-art sprite may look unstable as the camera moves.

2.6.2. Objects that move at runtime

The step above only fixes the position once, in the editor. It won't work for characters or other script-driven objects whose position changes every frame during gameplay. For those objects:

1. Create a child GameObject under the moving object, and move its `SpriteRenderer` (and any `Animator`) onto that child.
2. Add a **Pixel Snapped Transform** component to the child, and set its **True Transform** field to the actual transform of the object (Usually its parent).
3. The child now snaps to the grid every frame, following its true transform, while the object itself keeps moving exactly as your own script or physics expects.

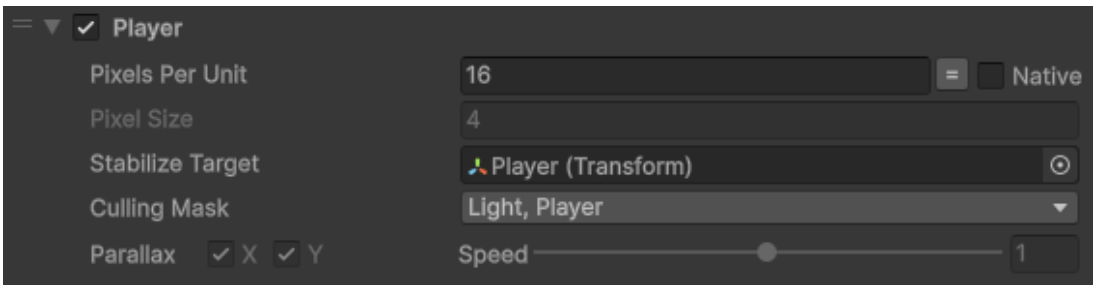


2.7. Pixelation Layers for Objects that Move

Objects moving in a pixelation layer may appear to jump between pixels instead of moving smoothly visually. This is especially noticeable with slow-moving objects and camera-followed objects. `Stabilize Target` prevents this by making the layer's pixel grid follow the object.

To stabilize a moving object:

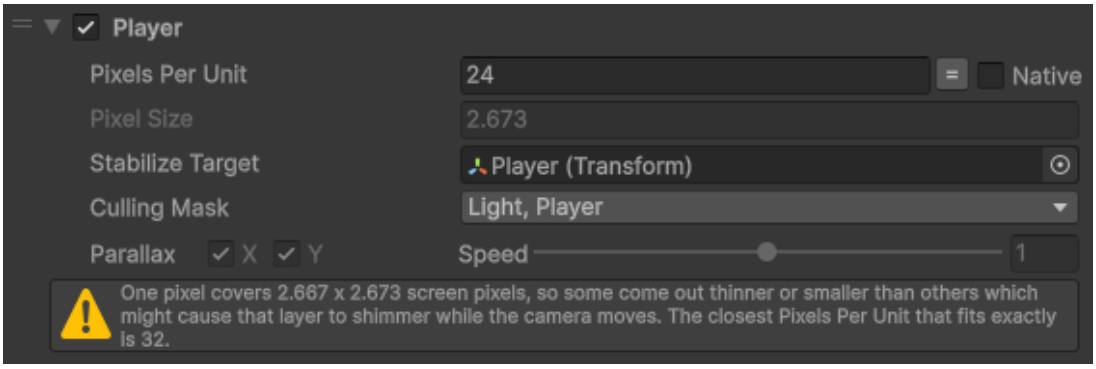
1. Give those objects their own Pixelation Layer, if they didn't have one already.
2. Set that layer's `Stabilize Target` to the object's Transform.



***Note**: Enabling `Stabilize Target` disables `Parallax` on that pixelation layer. See [Stabilize Target] for more information.*

2.8. Verify Your Setup

Every Pixelation Layer shows a live **Pixel Size** readout, with a warning if that layer's pixels aren't coming out whole and square. That warning usually means this layer's Pixels Per Unit doesn't fit your current window size cleanly, which can shimmer as the camera moves. If you see a warning, try adjusting **Base Pixels Per Unit**, or use the `=` button next to **Base Resolution Height**, **Minimum Camera Scale**, **Camera Scale**, or the layer's own **Pixels Per Unit** to choose a recommended value.



***Important**: It's highly recommended to use the `=` button whenever possible. It's specifically designed to find values that produce square, whole-number pixels with your current settings.*

3. Core Concepts

3.1. Pixelation Layer

Each Pixelation Layer renders part of your scene with its own camera and its own resolution, then stacks that result onto the display camera. Because layers are independent (Each pixelated on its own grid), each one can have its own Pixels Per Unit.

3.2. Stabilize Target

By default, a pixelation layer's pixel grid is fixed to the world. As an object moves, it renders at whichever grid cell its actual position is closest to. So as it crosses the midpoint between two cells, it visibly jumps to the neighboring pixel instead of sliding smoothly between them.

Stabilize Target makes the layer's pixel grid follow a target Transform, so the target always sits exactly on the grid instead of rounding to the nearest cell. This lets it move smoothly while staying pixelated.

3.3. Letterbox

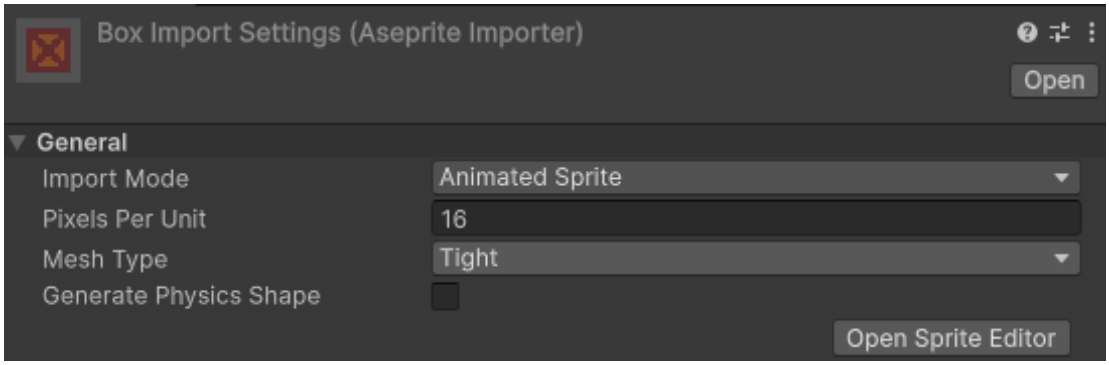
Letterbox is the black bars shown at the top and bottom of the screen if the screen height isn't an exact multiple of **Base Resolution Height**. That's because content can only scale up in whole numbers to keep every pixel a clean, uniform size. Letterbox bars paint that space black instead of stretching content unevenly to cover it. A value from **Base Resolution Height**'s `=` button avoids this for your current window size.

3.4. The `=` Button

Every `=` button next to a numeric field opens a menu of recommended values, though what that recommendation actually checks differs by field. On **Minimum Camera Scale**, **Camera Scale**, and a layer's own **Pixels Per Unit**, every value listed is checked against your current layers and guaranteed to keep them clean. On **Base Resolution Height**, it only guarantees no letterboxing and an even Base Pixel Size; whether a specific layer's own Pixel Size also comes out clean still depends on that layer's PPU.

3.5. Pixels Per Unit (PPU)

Sets the resolution of the Pixelation Layer. For pixel-art sprites, this should match the PPU they were imported with. Using a different PPU can cause pixels to shift or jitter as the camera moves. Non-pixel-art sprites can use any PPU value.



3.6. Camera Scale & Minimum Camera Scale

Camera Scale is the camera's live zoom (`1` is normal, above `1` zooms in). **Minimum Camera Scale** is the most zoomed-out value you'll ever set Camera Scale to.

Set Minimum Camera Scale to the smallest headroom you'll actually need, not the smallest number possible: the lower it is, the more world a layer pre-renders.

3.7. Snap to Pixel Grid

Pixel Camera Pro > Snap to Pixel Grid and **Pixel Snapped Transform** work the same way. Each looks at the sprite's own Unity Layer, finds the Pixelation Layer whose Culling Mask covers it, and uses that layer's PPU to snap the sprite to the grid. Because of this, any Unity Layer your snapped sprites live on must be covered by exactly one Pixelation Layer.

4. Clean Up Unused Quad Display Layers

Each Pixel Camera Pro claims its own Unity Layer slot for its Quad Display objects, so multiple cameras in the same scene don't render into each other's display. This runs automatically whenever your scene hierarchy changes, freeing any slot no live camera is using anymore, so deleting a camera frees its Layer on its own. **Tools > Pixel Camera Pro > Clean Up Unused Quad Display Layers** does the same cleanup.

5. Scripting API

Get the Pixel Camera Pro component and set its properties directly,

for example `GetComponent<PixelCameraPro>().CameraScale = 2f;`.

A layer's own properties live on its Pixelation Layer Camera component instead, reachable through `Layers`, for example `pixelCamera.Layers[0].Speed = 0.5f;`.

Property	Component	Notes
`CameraScale`	Pixel Camera Pro	Clamped between `MinimumCameraScale` and `5`.
`ParallaxOrigin`	Pixel Camera Pro	Parallax reference point.
`BasePixelsPerUnit`	Pixel Camera Pro	Clamped to a minimum of `1`.
`BaseResolutionHeight`	Pixel Camera Pro	Clamped to a minimum of `1`.
`MinimumCameraScale`	Pixel Camera Pro	Snapped to the nearest `0.1`, clamped between `0.5` and `1`.
`Speed`	Pixelation Layer Camera	Per-layer parallax speed.
`ParallaxX`	Pixelation Layer Camera	Camera parallax movement in X axis.
`ParallaxY`	Pixelation Layer Camera	Camera parallax movement in Y axis.
`PixelsPerUnit`	Pixelation Layer Camera	Clamped to a minimum of `1`.
`IsNativeResolution`	Pixelation Layer Camera	Switches between pixelated and native, full-resolution rendering.
`StabilizeTarget`	Pixelation Layer Camera	Anchors this layer's pixel grid to a Transform instead of world origin.

A layer's Culling Mask, enabled state, and name are just plain Unity `Camera` and `GameObject` members, not anything Pixel Camera Pro adds itself, so `GetComponent<Camera>().cullingMask` and `gameObject.SetActive(…)` work exactly like they would anywhere else. You can't add or remove a layer at runtime, creating one relies on Editor-only tooling.

6. Troubleshooting / FAQ

6.1. Lighting flickers as the camera moves.

Please make sure the `Light Render Texture`'s `Render Scale` is set to `1` in your Renderer Data.

6.2. Pixelated sprites look unstable while the camera moves.

6.2.1. Pixelated sprite's shape is not consistent.

This usually means the sprite's own position isn't snapped to the pixel grid yet. See Snap your sprite to the grid in the walkthrough. If the sprite is already snapped, check that layer's **Pixel Size** readout instead. An uneven value causes the sprite to shimmer too.

6.2.2. Camera-followed objects shimmer as camera moves.

This can happen if no stabilize target is applied in the pixelation layer. See Layers for objects that move for the recommended setup.

6.3. A layer's Pixel Size warning won't go away.

Try to adjust the config using the `=` button. It's specifically designed to find values that produce square, whole-number pixels with your current settings.

6.4. Cinemachine fights my camera's zoom.

See Cinemachine. Don't put a Cinemachine Brain on the same camera as Pixel Camera Pro, use **Cinemachine Position Follower** to track a separate proxy instead.

6.5. Black bars appear at the top and bottom of my screen.

See Letterbox. the `=` button next to Base Resolution Height avoids this for your current window size.

6.6. "No empty Layer slot" warning in the console.

Unity only gives you 32 Layers total, and Pixel Camera Pro needs one per camera. Free one up in **Edit > Project Settings > Tags and Layers**, or use **Tools > Pixel Camera Pro > Clean Up Unused Quad Display Layers** first to cleanup unused Quad Display layers.

7. Supports

- Discord: https://discord.gg/9jVjbSnpcH
- Gmail: greed.jesse.business@gmail.com