

# Table of Contents

[About PixelRender](#)

[PixelRender License Details](#)

[Camera Setup](#)

[Create a RenderTexture](#)

[Create a PixelRenderCamera](#)

[Camera FOV](#)

[Drawing to the Screen](#)

[Pixel Art Shader](#)

[Texture & Palettes](#)

[Palette Mixing](#)

[Light & Shadows](#)

[Pixel Outline Effect](#)

[Depth Threshold](#)

[Palette Editor](#)

[Palette Texture Layout](#)

[Edit Modes](#)

[Shade Generation](#)

[Palettize Texture Tool](#)

[Animation Quantizing](#)

[Mechanim Quantizing](#)

[Transform Quantizing](#)

# About PixelRender

PixelRender is a collection of shaders and tools for rendering 3D scenes as 2D pixel art. The key components are the PixelArt shader for rendering meshes using palette based shading, the PixelOutline image effect for drawing pixel-perfect outlines and the Palette Editor which provides several features for manipulating palettes used by the PixelArt shader directly within Unity3D.

To get the most out of PixelRender, please take a moment to read this manual and explore the included "Shooter" example game.

If you have any questions, get in touch!

email: [support@kode80.com](mailto:support@kode80.com)

twitter: [@kode80](https://twitter.com/kode80)

# PixelRender License Details

PixelRender is free to use for strictly **non-commercial** purposes. The full non-commercial license is included in the project root.

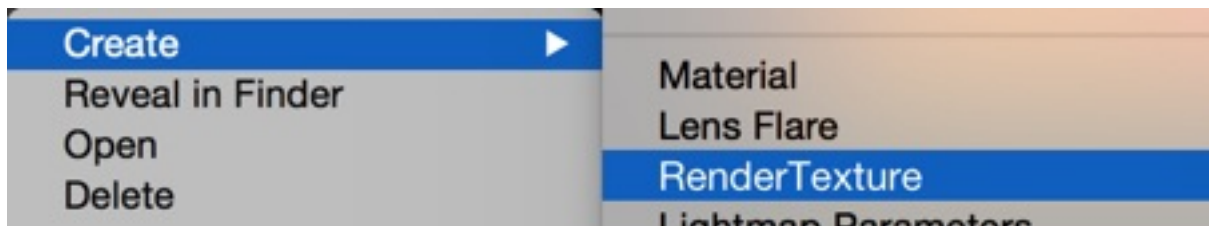
To use PixelRender for **commercial** purposes, or to support this & future kode80 development, **commercial licenses** are available for purchase from: <http://kode80.com/>

# Camera Setup

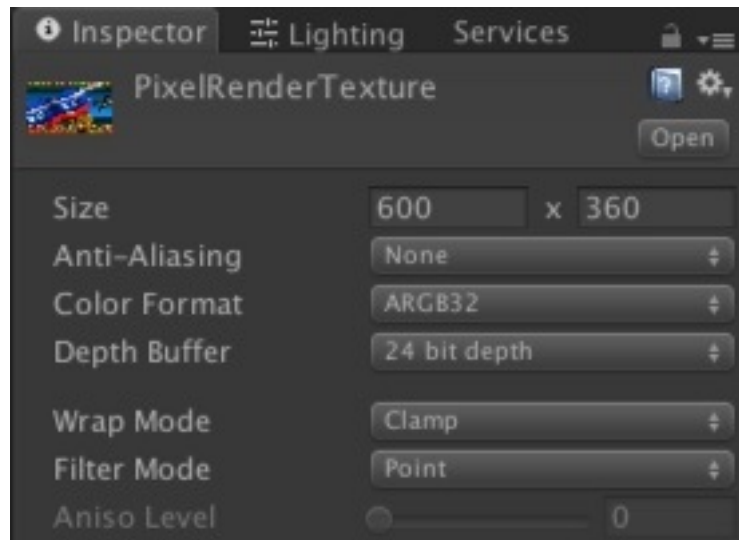
The first step to achieving a pixel art look is setting up your scene's cameras. All scenes that use PixelRender must have at least 2 cameras; one for rendering the scene itself to a lower resolution RenderTexture and the Main Camera included in all Unity3D scenes to draw the RenderTexture to the screen.

## Create a RenderTexture

When starting a new PixelRender project you must create a RenderTexture for PixelRender to use.

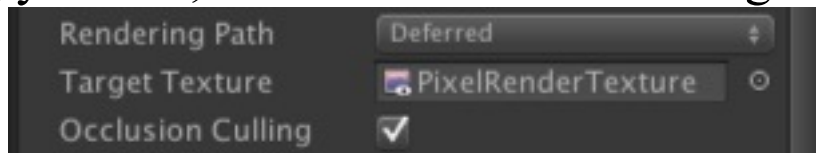


You can set the size of this RenderTexture to whatever you like, but generally a good resolution is half the width/height of your target screen resolution. You should also set the RenderTexture's Filter Mode to Point as this will maintain crisp pixels when drawn to the screen.



## Create a PixelRenderCamera

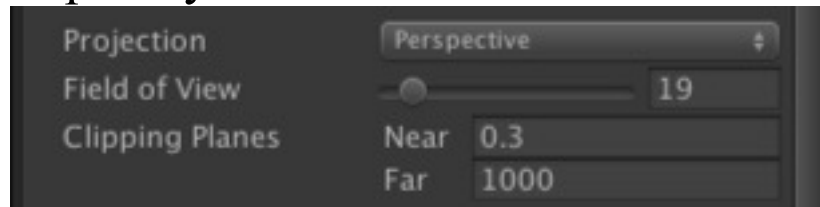
When starting a new scene you must create a 2nd camera for PixelRender to use. Add a standard Unity3D camera to your scene and set its Target texture to your PixelRenderTexture. You can name this camera anything you like, PixelRenderCamera is a good choice.



## Camera FOV

Aside from rendering at a lower resolution, another important factor for achieving a 2D pixel art look is minimizing the effect of perspective on your scene. By lowering the PixelRenderCamera's Field of View property, scene object's depth will appear compressed. The FOV setting you use is dependent on the look you

want for your project but a good range is between 20 and 1. An FOV setting of 20 will still have a subtle amount of perspective while an FOV setting of 1 will appear completely flat.



As you decrease the PixelRenderCamera's FOV setting, the relative distance between scene objects and the camera will also need to be *increased* to counteract the change in perspective. That is, the smaller the FOV, the larger the distance between the camera and an object will need to be to produce an equivalent image of the same scene with a larger FOV.

To visually pick appropriate camera settings, switch to a top down view, move the camera back until your scene fills the game view and then adjust the Near and Far Clip Planes so that they encompass your scene.



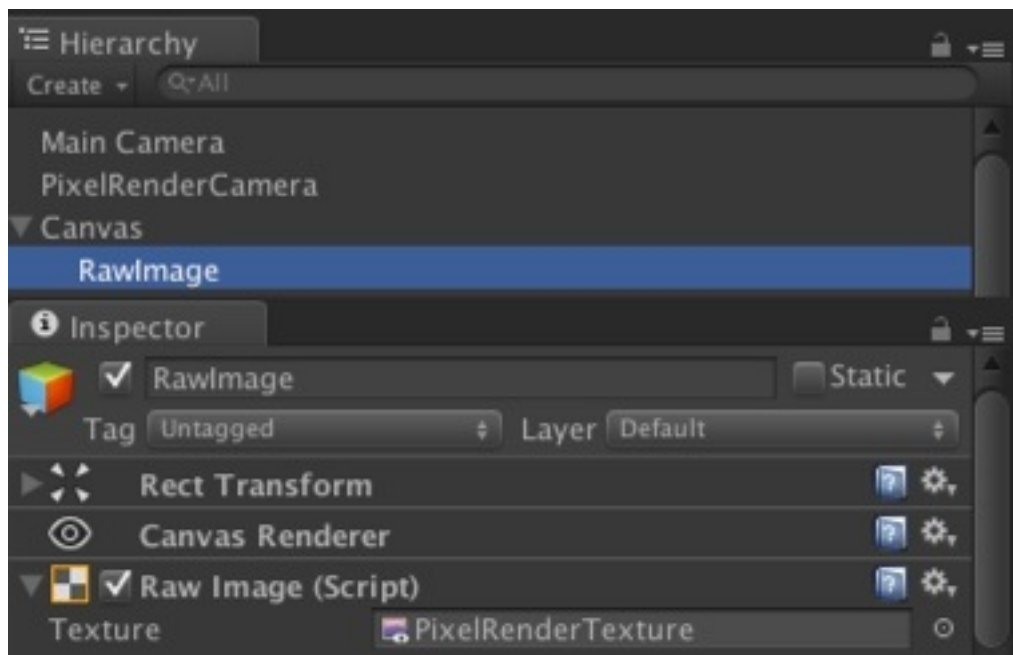
Note: while using an orthographic camera might seem like the first choice, this brings with it several

limitations such as incompatibility with deferred rendering and broken shadows. It also limits the control we have over perspective. Using the PixelRenderCamera's FOV setting we can choose to keep a small amount of perspective distortion which can provide subtle parallaxing for free.

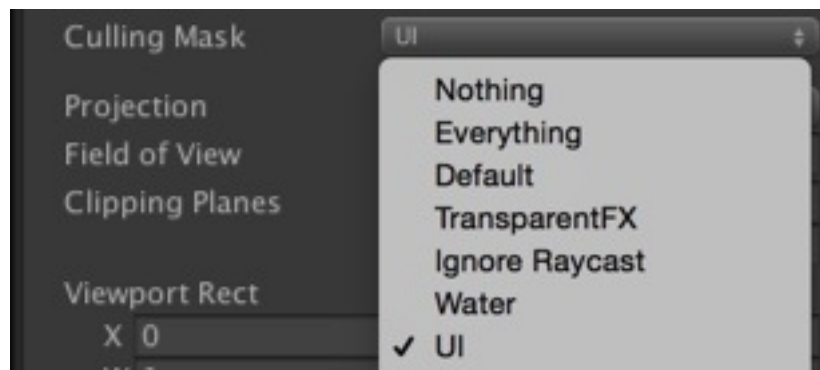
## **Drawing to the Screen**

With the PixelRenderCamera setup, your scene is now being rendered to the PixelRenderTexture. You are free to display this however you choose but the quickest approach is using Unity3D's UI system.

To display PixelRender's output using Unity3D's UI system, add a Canvas to your scene and then add a RawImage to this canvas, finally assign your PixelRenderTexture to the RawImage's Texture property and set the Width/Height you want it displayed as.



As a final step, set the Main Camera's Culling Mask to UI only. This will cause the Main Camera to only draw the PixelRenderTexture, rather than also rendering your scene a 2nd time.





# Pixel Art Shader

One of the most important aspects of the pixel art *look* is using a limited set of artist chosen colors. Assigning a PixelArtShader material to your 3D objects gives you full control over the palette used to render them, how much dithering is used and lighting/shadows.

## Texture & Palettes

Rather than using a typical diffuse/albedo texture, the PixelArtShader uses a special gray scale *index* texture. Each shade of gray used in this texture is interpreted by the PixelArtShader as the index of a color slot in the palette, the lighting calculations then pick the appropriate *shade* of this color for the output pixel.



Using index textures and palettes has several advantages. Palettes can be adjusted directly in the editor giving full artist control over the scene's appearance. Entirely new palettes can be created easily without having to change the object's main texture. And

not least, the colors used for realtime lighting are under full artist control.

**Want a different enemy variant?**

Just duplicate the material and palette then adjust the clothing colors in the new palette.

**Want shaded areas of an object to have a blue tint?**

Just adjust the hue of those shades in the palette.

To learn more about creating index textures and palettes take a look at the [Palette Editor](#) and [Palettize Texture Tool](#) sections.

## Palette Mixing

PixelArtShader materials can optionally include a second palette and mix between the 2 palettes via the Palette Mix property. When Palette Mix is set to 0 the first palette will be used, when Palette Mix is set to 1 the second palette will be used and values in between will result in a lerp between the 2 palettes.

**Want enemies to flash when hit?**

Create a 2nd *hit* palette and set Palette Mix to 1 when the enemy is hit.

**Want to emulate 2D colored lighting?**

Create a 2nd palette for when the object is under the light and adjust Palette Mix based on distance to the light.

## Light & Shadows

Each unique PixelArtShader material has it's own light source, this is separate from Unity3D's lighting system allowing for full artist control over the appearance of light on a per-material basis which is particularly important for achieving the pixel art look.

All lighting calculations are done in object space based on the PixelArtShader's Light Direction property. While lighting will always exclusively use the shades defined in the assigned palettes, it is also possible to optionally create in-between shades with dithering via the Dither property.

The PixelArtShader can optionally include Unity3D shadows on a per-material basis. To enable shadows; simply add a directional light to your scene and check the Shadows property on each PixelArtShader material you want to receive shadows.

# Pixel Outline Effect

Another aesthetic choice often seen in 2D pixel art is the use of pixel-wide, colored outlines. These outlines can range from subtle variations of the underlying color to fully opaque high contrast colors such as black or white.

With PixelRender you can also achieve this look in your 3D scene using the PixelOutlineEffect component. Simply add the PixelOutlineEffect component to your PixelRenderCamera and adjust the Outline Color and Depth Threshold properties.



## Depth Threshold

The PixelOutlineEffect works by comparing the depth of surrounding pixels and using the Depth Threshold property to determine if the current pixel is part of an *outline*. High Depth Thresholds will result in silhouette outlines, while lower Depth Thresholds will include internal outlines.

It's important to note that due to the way depth textures

























work, the `PixelRenderCamera`'s Near and Far Clip Plane settings will have an impact on what the `PixelOutlineEffect` considers an outline. If you change your `PixelRenderCamera`'s clip plane, you may also have to adjust your materials' Depth Threshold property to match the new settings.

# Palette Editor

Editing palettes used by PixelArtShader materials is an important and powerful part of PixelRender. The included Palette Editor tool (Window->kode80->PixelRender->Palette Editor) allows you to quickly alter the overall look of objects in your scene from directly within Unity3D. Once you've loaded a palette texture into the Palette Editor, you can adjust the number of shades the palette uses, directly edit each color/shade and automatically generate shades for each color slot using HSL shifting.

## Palette Texture Layout

The order of colors in PixelRender palette textures has special relevance. Each column represents a unique color, while each row represents a specific shade of the colors, with brightest at the top and darkest at the bottom.

		Colors					
		1	2	3	4	5	6
Shades	1						
	2						
	3						
	4						

Put another way the width of the palette texture is the number of unique colors accessible by the index texture, while the height is the number of shades for each unique color when the shader calculates lighting.

## Edit Modes

To speed up palette editing, the Palette Editor provides several different edit modes that will affect all palette operations.

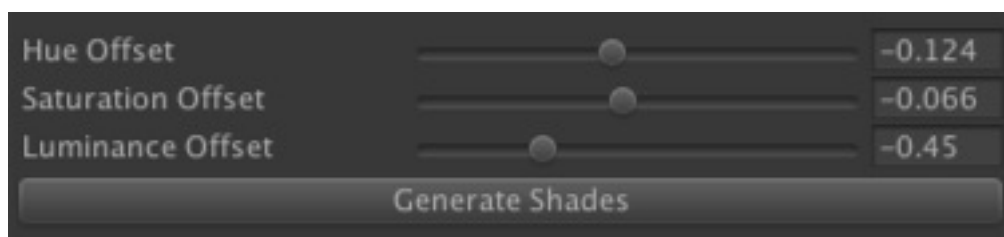
- **Single:** the operation will be performed on the selected color
- **Row:** the operation will be performed on all colors in the selected color's row
- **Column:** the operation will be performed on all colors in the selected color's column
- **All:** the operation will be performed on *all* colors

For example to change all colors in a row at once, make

sure Edit Mode is set to Row then select and change any color in that row.

## Shade Generation

While you are free to hand pick every shade in the palette manually, the Palette Editor can also generate them for you by gradually shifting hue, saturation and luminance of the top row's colors over the full shade range.



To generate shades for the full palette; make sure Edit Mode is set to All, use the Hue/Saturation/Luminance sliders to pick how the much the final shade will be shifted and then click the Generate Shades button.



# Palettize Texture Tool

The preferred method for creating the index textures needed by PixelArtShader materials is the Palettize Texture Tool (Window->kode80->PixelRender->Palettize Texture) included with PixelRender. Using this tool, simply pick your input diffuse texture and then click the Save Palette & Texture button to save both an index texture and a palette texture next to the input diffuse texture.

The Palettize Texture Tool works by creating a list of *all* unique colors found in the input diffuse texture and then generating the output index texture and palette texture accordingly. This means care must be taken to only include colors in the diffuse texture that you wish included as a *unique* color in the output palette.

Note: you should not include shading in the input diffuse texture *unless you specifically require that shade to be a unique color in the palette.*

# Animation Quantizing

Animations in traditional 2D pixel art are by their nature restricted to a relatively small number of hand drawn frames. 3D animations don't suffer this limitation as every keyframe in an animation can be smoothly interpolated. While this is usually desirable, if your project aims to fully emulate a 2D look you'll want to quantize at least some of your animations.

## Mechanim Quantizing

To quantize Mechanim animations simply add the `QuantizeAnimation` component to a `GameObject` that has an `Animator` component. Setting the `FPS` property controls the quantized framerate of the animation. Setting the `Animation Name` property controls the currently playing animation.

## Transform Quantizing

To quantize transform properties such as rotation, `PixelRender` includes the `QuantizedValue` family of helper classes. These classes allow you to manipulate a *real* value while accessing it's *quantized* value.

For example, to quantize an object's local rotation to 8

steps:

1. Create a `QuantizedVector3` member variable in your script called `quantizedRotation`
2. Set `quantizedRotation.quantizeStep` to `Vector3(360.0f/8.0f, 360.0f/8.0f, 360.0f/8.0f)`
3. Each frame update, perform any calculations on `quantizedRotation.realValue` and then assign `quantizedRotation.quantizedValue` to `transform.localEulerAngles`

# Table of Contents

About PixelRender	2
PixelRender License Details	3
Camera Setup	4
Create a RenderTexture	4
Create a PixelRenderCamera	5
Camera FOV	5
Drawing to the Screen	7
Pixel Art Shader	9
Texture & Palettes	9
Palette Mixing	10
Light & Shadows	11
Pixel Outline Effect	12
Depth Threshold	12
Palette Editor	14
Palette Texture Layout	14
Edit Modes	15
Shade Generation	16
Palettize Texture Tool	17
Animation Quantizing	18
Mechanim Quantizing	18
Transform Quantizing	18

