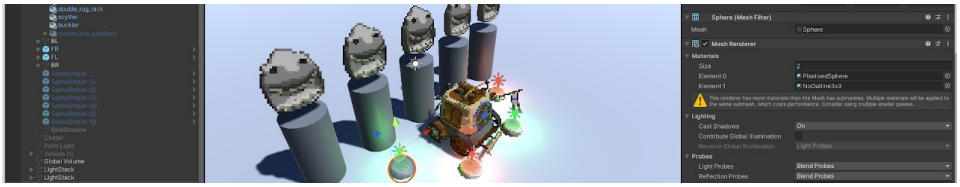
User Guide



Requirements

The package has the following minimum requirements*:

- Universal Render Pipeline 7.3.1

Please let me know if you find it working without these requirements, or not working with them!

I have tested using the following versions of Unity:

- ProPixelizer v1.3
  - 2019.3.0 + URP 7.4.x
  - 2020.1.2f1 + URP 8.2.0
  - 2020.2.1f1 + URP 10.2.22
- ProPixelizer v1.4
  - 2019.4.23f1 (LTS) + URP 7.3.1
  - 2020.2.1f1 + URP 10.2.22

Build Targets

I've tested:

- webGL ES 2.0 (firefox and chrome), running on a desktop.
- Windows PC, both Direct X and openGL.
- Android (works, but laggy on my low end phone, a Galaxy A10).

If you are interested in how ProPixelizer works, I give a brief description of the pixelization process in this article, under Attempt #3. I intend to write more technical articles in the future. Feel free to contact me on Discord or twitter with questions!

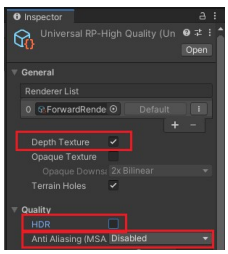
Getting Started

A really quick video tutorial can be found here: https://www.youtube.com/watch?v=S9tT6Fu1lRQ

Below is a more detailed step-by-step guide on how to set up a project to use ProPixelizer.

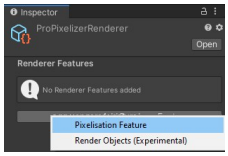
First, make sure you have Universal Render Pipeline package added to the project, then add the ProPixelizer package.

Setting up the Render Pipeline



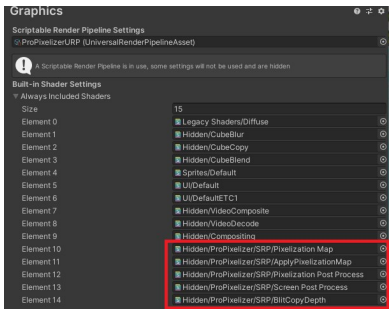
In the Universal Render Pipeline Asset, enable *Depth Texture* and disable *HDR*.

Disable *Anti Aliasing*.



Select the Renderer asset (here - ForwardRenderer, in the RendererList). Below *Render Features* click *Add Render Feature*, and select *Pixelise Feature* to add the render pass required for ProPixelizer.

Setting up the Project

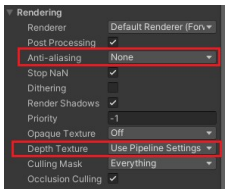


Under `Project Settings > Graphics`, add the following ProPixelizer shaders to the Project's *Always Included Shaders*:

- Hidden/ProPixelizer/SRP/Screen Post Process
- Hidden/ProPixelizer/SRP/Pixelization Post Process
- Hidden/ProPixelizer/SRP/BlitCopyDepth
- **v1.3+:** Hidden/ProPixelizer/SRP/Pixelization Map
- **v1.3+:** Hidden/ProPixelizer/SRP/ApplyPixelizationMap

This step is required because some versions of Unity have an issue detecting that these shaders are used by the Project - because they are only referenced in the code (specifically, in the Pixelise Feature*) - and so Unity mistakenly omits them from the build. It's only required for building your project, and you can skip it for the moment if you are in a rush to get started.*

Setting up the Camera



Configure the camera as follows:

- *Rendering/Anti-aliasing*: None
- *Rendering/Depth Texture*: Use Pipeline Settings
- *Output/HDR*: Off

Post processing can be used, eg *Bloom* and *Vignette*, and configured through a Volume as per the usual Unity workflow. Note that *Depth of Field* is currently unsupported (but will be added in a future release).

For best results, it is strongly recommended to **use orthographic rather than perspective** projection. This is because pixel creep can only be fully eradicated for cameras using orthographic projection; for orthographic projections, the size of an object does not change as it moves within the camera's view.

Add the **Camera Snap SRP** MonoBehaviour to the camera game object.

Note that the camera's world-space pixel size will be set by the **Camera Snap SRP** component during runtime.

Updating from previous versions

Updating to **v1.4**:

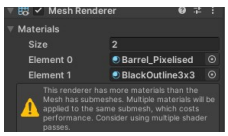
- I changed the name of some shader keywords to be consistent between outline and appearance materials. After you update, run the tool *Window/ProPixelizer/Verify Materials* to fix any broken materials in your project (this will update all materials to use the new keywords).

Pixelizing Objects

Objects can be pixelized into *macropixels* of size 1x, 2x, 3x, 4x, and 5x screen pixels.

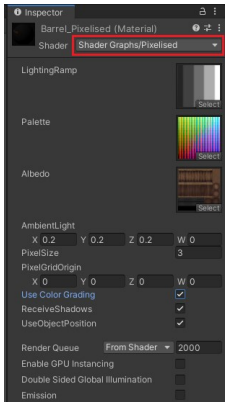
Objects are pixelised by applying two render passes to them.

- For **ProPixelizer v1.0-v1.3**, two materials must be added (with each material controlling one pass):
  - A material to determine appearance.
  - A material to determine the outline and object ID. This must be added even if no outline is desired - just set the outline color to an alpha of 0.
- In **ProPixelizer v1.4+**, you may either use the previous two-material approach, or you can use a single *ProPixelizer/SRP/PixelizedWithOutline* material. The properties of this material are the same as before - it just combines the two passes into one material for convenience. An example can be found here.



Example material setup for ProPixelizer v1.0-v1.3

The image on the left shows an example setup for correctly render an object as pixelated. The warning can be safely ignored - the outline material only defines a shadowcasting pass and a pass used for rendering metadata and does not lead to performance loss during normal rendering passes.

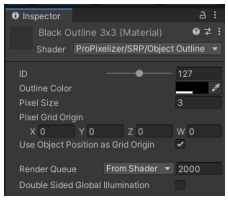


Appearance material

The package contains an example ShaderGraph material that implements the pixelisation effect, along with cell shading and color grading. Materials using this ShaderGraph can be used for the appearance of the object. The properties are:

- **LightingRamp:** A texture ramp used for cell shading, which helps give a pixel-art aesthetic.
- **Palette:** A texture look-up table (LUT) used for color grading when the 'Use Color Grading' option is enabled. A number of LUTs from well-known devices (eg NES, PAL, GameBoy) are shipped with **ProPixelizer**, and a tool (Window -> ProPixelizer-> Palette Builder) can be used to create new LUTs from palettes. The LUT can also include dither patterns as demonstrated here.
- **Albedo:** Texture to use for the object color
- **Ambient Light:** A background level of lighting. This can be helpful to tweak the appearance, in conjunction with the color grading.
- **Use Object Position:** Whether to use the object's world space position, or the *PixelGridOrigin* variable, when determining how to align pixels. Setting this value to off, and setting *PixelGridOrigin* can be useful to ensure that different objects use pixel grids that are aligned. For instance, you would use this to ensure that equipment on a character is drawn with pixels aligned to those of the character itself (so that it looks like a single sprite).

New in v1.3: *Appearance materials now have a custom ShaderGUI, which prevents the same information as above but ordered by category.*



Outline material

The outline material has the following properties:

- **ID:** A number. Outlines are drawn when pixels have an ID different to those around them. If two objects should have outlines when they meet, give them different IDs (eg, two enemies). If they should not have outlines (eg, a character and their equipment), give them the same ID. The value should be an integer in the range (0, 255).
- **Outline Color:** The color to use for the outline. The alpha value controls blending with the scene color. Alpha values of 0 can be used for invisible outlines, 1.0 can be used for block color, and fractions to blend with the appearance material color.
- **Pixel Size/Pixel Grid Origin:** These are the same as in the appearance material. **For almost all situations, you should make sure these values match those in the appearance material.**

An outline material must be added to the object even if the outline is set to invisible; the outline material is used to render metadata required for outlines and the pixelisation of objects.

Technical note for advanced users: If you are writing your own ShaderLab shaders, you can perform the pixelisation and outline within a single material shader. In the fragment shader, you should `clip()` the output of `PixelClipAlpha_float` in `PixelUtils.hlsl`, and also add a separate Pass with Tags `"LightMode" = "Outlines"`, which you can copy directly from the `ObjectOutline.shader`. The **only** reason a separate material is used by default for the outlines in ProPixelizer is to maintain support for ShaderGraph materials, which currently do not allow me to add a separate pass to the shader. In ProPixelizer v1.4 I added a multi-pass shader that uses passes from the ShaderGraph shader and the outline shader.

Eradicating Pixel Creep

Pixel creep is a problem that occurs frequently when rendering 3D objects as pixel art - the object appears to shimmer as it moves across the screen. An example can be seen in this video: https://www.youtube.com/watch?v=lO8O5tY-wZI . The creep occurs because the number of pixels that an object occludes changes as it moves across the screen.

Pixel creep can be removed by aligning objects to the pixels of the screen before rendering them. **Note that pixel creep can only ever be solved for orthographic projections**; in a perspective projection, the object's size will change as it moves on the screen.

For orthographic projections, ProPixelizer provides functionality to handle this for you, through two MonoBehaviours:

- A `CameraSnapSRP` MonoBehaviour, which should be attached to your camera. The `PixelSize` property determines the size of one camera pixel in world units.
- An `ObjectRenderSnapable` MonoBehaviour, which should be attached to meshes that you are rendering.

These MonoBehaviours will snap object positions before rendering and restore them afterwards. The implementation respects transform hierarchies.

Snapping Angles

The `ObjectRenderSnapable` MonoBehaviour can also snap the angles of objects being rendered. Set `ShouldSnapAngles` to true if object angles should also be snapped, by the desired `Angle Resolution`. If you have nested transformations (eg for equipment), I recommend you only do this on the root transform.

Aligning Pixel Grids

The `ObjectRenderSnapable` MonoBehaviour also provides a way to align the pixel grids of child objects to their root transform. This is useful for things like equipment - you have different meshes but want the complete object to look like one sprite. Set the property `Use Root Pixel Grid` to true to cause the object's pixel grid to be aligned with the root transform of the hierarchy.

An example is given in the picture on the left, showing how the blue/green shield looks when it is and is not aligned to the pixels of the root transform. Look closely within the circles - on the unaligned you should be able to see the slightly 1 pixel misalignment of the two objects with 3x3 pixelsizes.

More control over object outlines

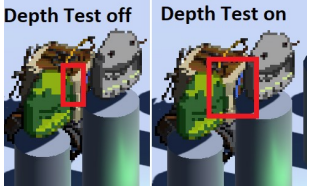
Outlines are drawn whenever two adjacent pixels have a different `ID`, as determined from the `Object Outline` material property.

The `OutlineControl` MonoBehaviour provides some extra ways to control the outlines of your objects:

- You can specify an `ID` to use, or set `Use Random ID` to true to generate a random ID at runtime.
- Set `UseRootID` to true if the outline material should use the ID of the root transform. This also requires an `OutlineControl` MonoBehaviour to be present on the root transform. Like with aligning the pixel grids, this is useful for child objects such as equipment attached to a player model - it ensures the outline is drawn around the entire player, and not around each individual piece of equipment. In the image on the left, I have set `UseRootID` to true on all attached equipment (eg the shield, crossbow, window, etc) but not around the wheels.
- The same is also true of color. You can specify the color to use for the outline with `Color`, and also whether to `UseRootColor`.

Note that many of these changes will only take place once you hit play mode (they require instancing the material).

Technical note for advanced users: The `ID` is rendered into the outline buffer with 8-bit precision, so only 256 different values of ID can be specified.



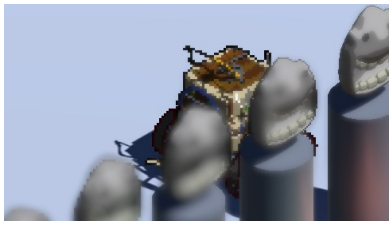
Depth Test outlines

This option was added in **v1.3** and can be found on the `PixelisationFeature` that you added to your `ForwardRenderer` Asset in the *Getting Started* section. When enabled, outlines will only be drawn for edge pixels that are in front of their neighbors. This helps achieve the feel of a hand-drawn sprite, for which the outline would not change depending on the geometry in front of it. A comparison of the two settings is shown in the image here.

Other tips for achieving a good pixel-art feel

Below are a collection of tips to bear in mind when using ProPixelizer:

- Try to avoid small features in geometry that are less than a pixel in size - otherwise they can flicker in and out of visibility.
- Old school sprite art typically only has a few different viewing angles - snap the angles of your objects to achieve the same feel.
- Reduce the number of keyframes in animation and use 'constant' interpolation to give it a stepped feel, as if flicking through pages of a sprite sheet.
- When using Color Grading, it helps to create your assets while targeting a particular color palette. There are significant differences between the various old-school color palettes. GameBoy is monochrome, for instance, while PAL is dark and grungy.



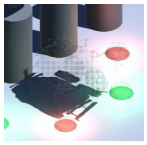
Note that you can also do neat things not normally seen in pixel art games, such as depth-of-field post processing.

I hope you enjoy using ProPixelizer. Please do message me if you have any questions or problems. I'd also be very pleased to hear what you make with it.

Cheers!

Elliot

FAQ



My object looks like a bunch of dots?

The problem: Pixelated objects appear as a bunch of dots when rendered, as if dithered.

The solution: The Pixelisation Feature needs to be added to the Render Features of the Scriptable Render Pipeline - see 'Setting Up The Render Pipeline' under 'Getting Started', above.

If you are interested in *why* it looks like this, you can find the answer in this Medium article which describes how ProPixelizer works. The method used for ProPixelizer is described under Attempt #3; objects are first drawn as a dithered matrix of dots, then the post process fills the surrounding screen pixels to produce the final pixelated image.

Help, I see a black screen in builds/nothing shows in my build!

Make sure that Unity is correctly **adding the ProPixelizer post-process shaders to your build** - in some configurations, Unity mistakenly assumes these shaders are not required and strips them. See 'Setting up the Project' in this guide for the configuration of 'Always Included Shaders' to use.

If you have multiple quality configurations (eg, different assets for low and high quality targets), make sure each Universal Render Pipeline Asset is also set up correctly as in the first section of this guide.

I have a mesh with multiple materials - how should I apply the appearance+outline materials?

- From **ProPixelizer v1.4+**, you can now apply a combined Pixelized+Outline material using the *ProPixelizer/SRP/PixelizedWithOutline* shader.

Update History

Version 1.4

- **Added** dither pattern support.
- **Added** single material to produce both outlines and appearance.
- **Added** example scene (*Floating*) to show new shader and a no-creep setup.
- **Fixed** occasional 'tearing' due to numerical precision error.
- **Fixed** creep in rare cases.
- **Improved** Palette Builder tool.
- Made shader keywords consistent - run *Window/ProPixelizer/Verify Materials* to fix broken materials.

Version 1.3

- **Added** option to not draw outlines when objects overlap (see option in render feature).
- **Added** alpha cutout support.
- **Added** custom shader GUI.
- **Fixed** Object flashing bug for Orthographic projections when near plane was negative.
- **Fixed** gaps at screen edge.
- **Fixed** warnings.
- **Performance improvement:** reduced number of PixelizationPass used by creating re-usable screen space pixelization map.

Version 1.2

- **Fixed** creep/malformation at some resolutions.
- **Fixed** tearing in perspective projection.
- **Fixed** 3x3 pixelation on AMD+openGL targets.
- **Quality of life:** the '_ID' property in the ObjectOutline shader is now specified as integer in the range (0,255). No change is required if using the OutlineControl MonoBehaviour.

Feel free to email with feature requests and suggestions!