ALL IN 1 SPRITE SHADER

By Seaside Studios
For technical support and requests write to:
seasidegamestudios@gmail.com
Twitter of the creator: https://twitter.com/GerardBelenguer

Index

Overview

First of all thanks for downloading this asset! The intention of this asset is to provide you with an all in one solution to include cool effects to sprites, UI images effects to your project in the easiest and fastest way possible.

What makes this asset unique is that you choose which effects you desire and the material will blend and stack all your desired effects appropriately. So the same material allows you to create a huge variety of effects without altering the setup of your sprites.

On top of that it offers a very flexible, powerful and easy to learn workflow that will allow you to enhance your projects visuals in the easiest way possible.

Link to the youtube playlist that explains how to use this asset:
https://youtube.com/playlist?list=PLKS0HUbkxp-mLfgqHc4Qglb7Maq0ZMGSo

Feel free to contact me over at this email if you have any issue, request or question. I'm always trying to improve the asset, open to suggestions and I'll be more than happy to help you out: seasidegamestudios@gmail.com

Please make sure to drop a review on the Asset Store page if you like the asset. I helps a ton:
https://assetstore.unity.com/packages/vfx/shaders/all-in-1-sprite-shader-156513

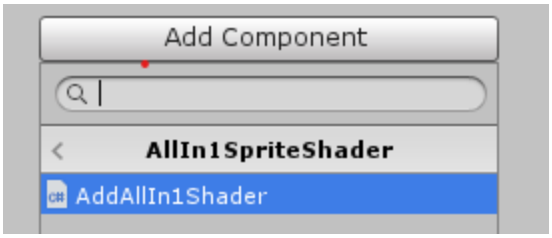
First Steps (Must Read)

In order to get everything looking like in the store images and trailer you'll need to add post processing Bloom and change a couple project settings, here's how:

URP pipeline steps:
1. Import the "URP_Import_All1Sh" Unity Package
2. Set Color Space to Gamma in Player Settings, Other Settings, Color Space
3. In Graphics Settings set the Scriptable Render Pipeline Settings to "AllIn1UrpAsset"
3. In Quality Settings set the Rendering asset to "AllIn1UrpAsset"
You can also watch these 2 short videos here:
https://youtu.be/e7jyq-MXLEo (URP Renderer Setup)
https://youtu.be/ZJBw7sGG63g (URP Post Processing)

Built In pipeline steps:
1. Install Post Processing package from the Package Manager
2. In Graphics Settings enable HDR in all checkboxes
3. Make sure you allow HDR in you camera
4. Add Post Process Layer and Volume
5. Properly configure them make making it Global and setting the Layer to the same Layer of the Camera you are using
You can also watch a video about it here:
https://youtu.be/pq5dTygcFVU

The asset includes a component that will do all the setup for you. The component is called "AllIn1Shader":

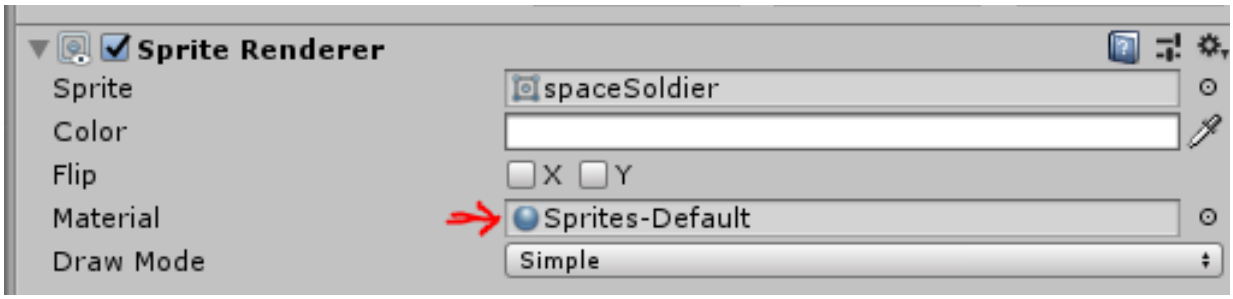


When you add it, the component will swap the current material for a new instance of the AllInOneSpriteShader material. The component also has some features that are overviewed in the next point.

Here you have a link to a video that gives an overview of the asset in case you prefer a visual explanation:

https://youtu.be/ThvqkJ5q-gk

But you can also do it the classic way: right clicking the AllInOneSpriteShader and then going Create->Material. You can then name this material and drag it into the Sprite Renderer Material slot in the Inspector of the desired Sprite:



The same process can be followed for UI Images, Particle Systems, Tilemap Renderers, Sprite Shapes and Mesh Renderers.

Component Features

Once added the component will look like this:

The buttons do the following:

- **Deactivate All Effects**: It will deactivate all effects but won't touch any properties. So if you activate an effect again you will obtain your previous visual results.
- **New Clean Material**: It will create a new instance of the AllInOneSpriteShader material and assign it to the Sprite.
- **Create New Material With Same Properties**: It will create a new instance of the AllInOneSpriteShader material with the same properties of the previous one. This is useful when you want to create a material similar to another one.
- **Save Material to Folder**: Creates a Material asset with the name of the current GameObject and saves it in the following path:"Assets/AllIn1SpriteShader/Materials". This can be used to assign the same Material to many different Sprites.
- **Apply Material To All Children**: Applies the material of the current selected object to all the objects under its hierarchy.
- **Change Shader Variant**: This dropdown allows you to swap the different variants of the shader included in the asset. You can learn more about the Scaled Time, Masked UI and URP 2D Renderer variants in the Scaled Time, UI Masking and URP 2D Lights sections of this Documentation.
- **Sprite Atlas Auto Setup**: Use this in case your sprite is contained inside an atlas. This will add the SetAtlasUvs component for you and will make sure that the effects get properly drawn on your sprite (See Sprite Atlases section for more details).
- **Remove Sprite Atlas Configuration**: Removes SetAtlasUvs component and the rest of the sprite atlas configuration.
- **Remove Component and Material**: Removes the component from the GameObject and sets the Sprite Material back to the Sprite/Default one.

Asset Window

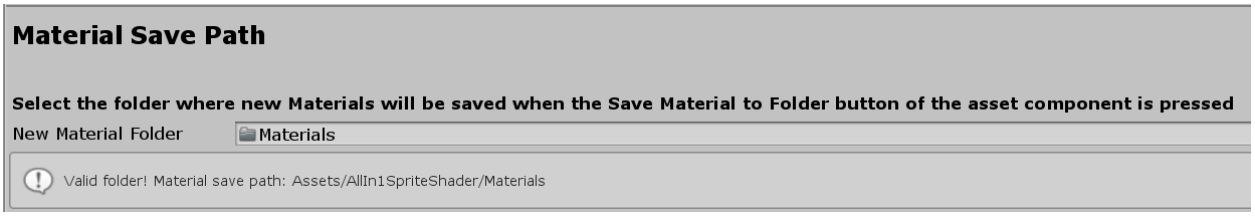
Video tutorial here: https://youtu.be/N0IEFVmuFvc

You can access it by going to Window -> AllIn1ShaderWindow.
The Asset Window offers you a bunch of settings options and utilities:

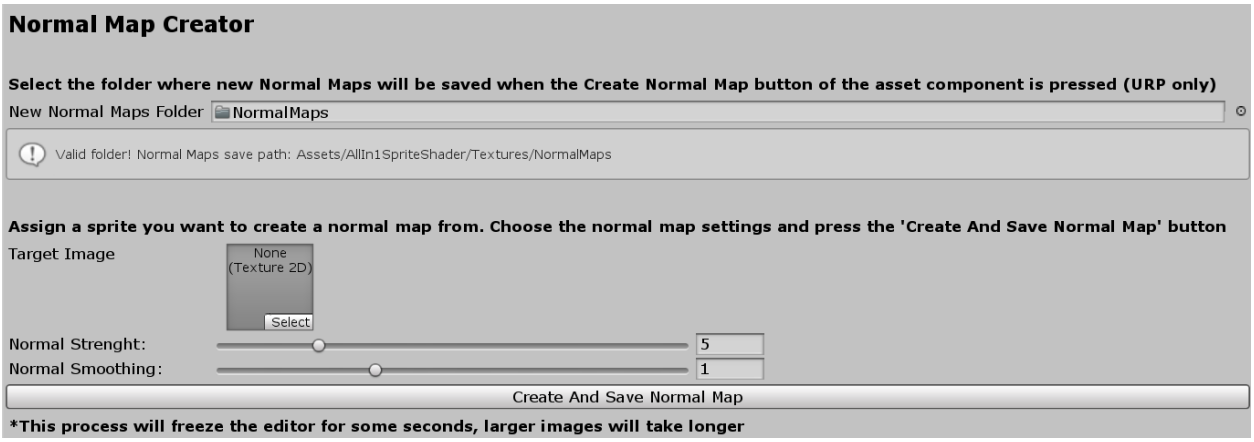
1. Changing the default Asset Shader: this is the shader that gets added when you add the asset component to a valid gameobject (a gameobject with a Renderer component or UI Image). Usually it should be Default or Urp 2D Renderer if you are doing a game with the new URP 2D Renderer Lights (this last option will only work if the URP package

is imported, see URP 2D Lights section for more info).

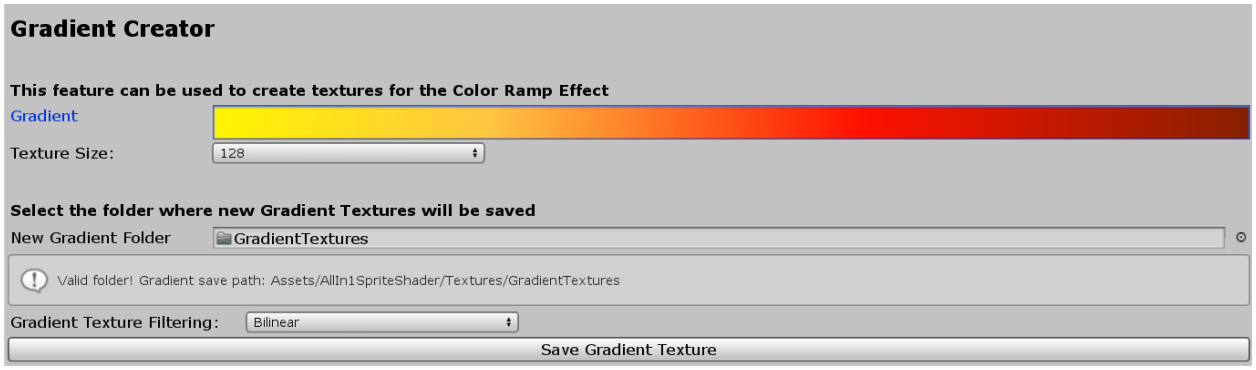
2. Material Save Path: As mentioned in the previous section the Asset can save Materials. By default these materials will be saved into a pre assigned “Materials” folder under the Asset root folder. But you can change the folder here:



3. Normal Map Creator: You can change the save route of the Normal Maps used in the Urp 2D Renderer shader and you can also create Normal Map textures using this window. You can do so by adding a Target Image, choosing a Normal Strength and Smoothing values and pressing the Create And Save Normal Map.



4. Gradient Creator: Allows you to create Gradient textures that can then be used with the asset shader. The most straightforward use for it is the Color Ramp effect. There is an example in the third row of the Demo called “Custom Gradient” that uses a texture created with this tool.



Textures Setup

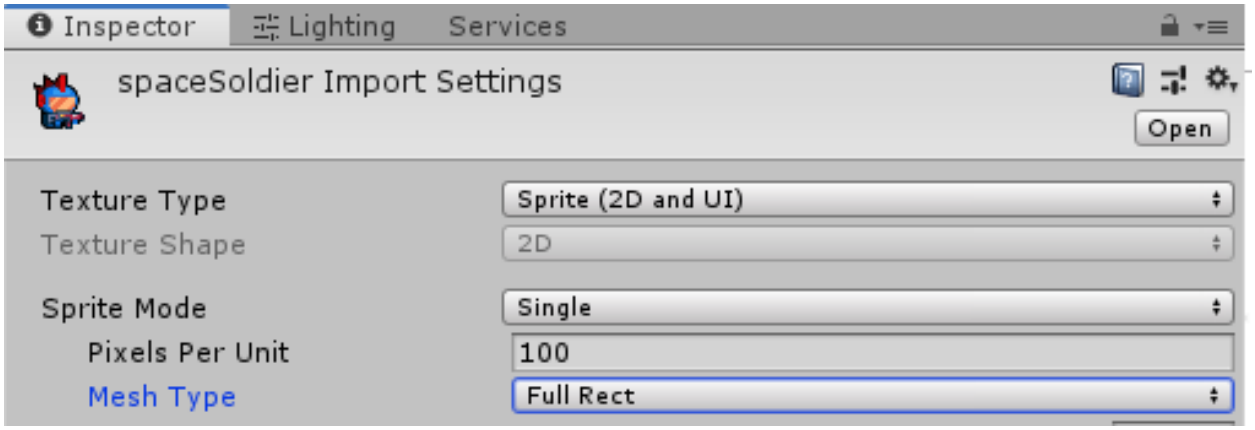
In order to get the effects looking as they should, it's important to know how to import and set up the textures we'll be using for our sprites and UI images.

Here you have a link to a video that explains how to setup your textures in case you prefer a visual explanation:
https://youtu.be/DGzBCGHg8BE

1. The most important part is having enough room within the sprite shape to show the effects. In the top of the Scene window you can choose how the scene view is rendered. If we change from Shaded to ShadedWireframe we'll be able to see the sprite rect size:

In the Shaded Wireframe view we can see black lines, that's the shape of the mesh where your sprite and effects will be rendered. To get all the effects to display properly we'll need more space. Otherwise the effects will get cropped where the black line ends.

The easiest way of doing so is changing the Mesh Type to Full Rect in the import settings of the sprite:

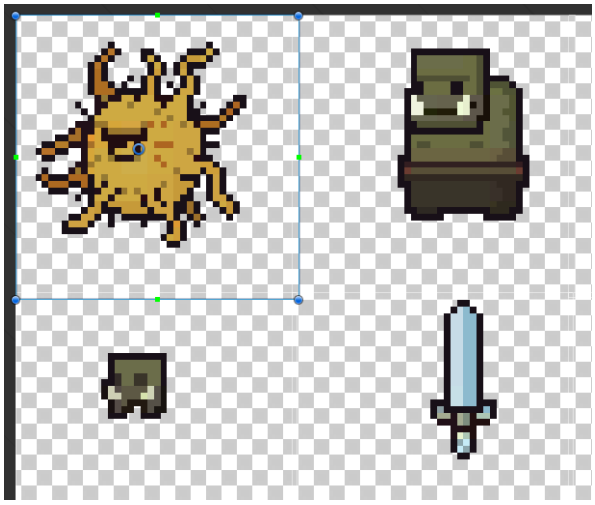


We should always have plenty of space around the sprite so that all the effects can get rendered properly like in this picture (red lines show all the available

space where effects can get rendered):

Moreover, if you need even more space you could use the Rect Size effect. This effect will make the available space for the sprite even bigger. Note that this effect will only properly work on single image sprites, it won't work on spritesheets. And doesn't work well for UI Images either, please use this effect as a last resort.

2. In a similar fashion if we are using a spritesheet you must import the sprites in Multiple Sprite Mode (as you always do) and then make sure that when you Slice your sprites they have some spacing:

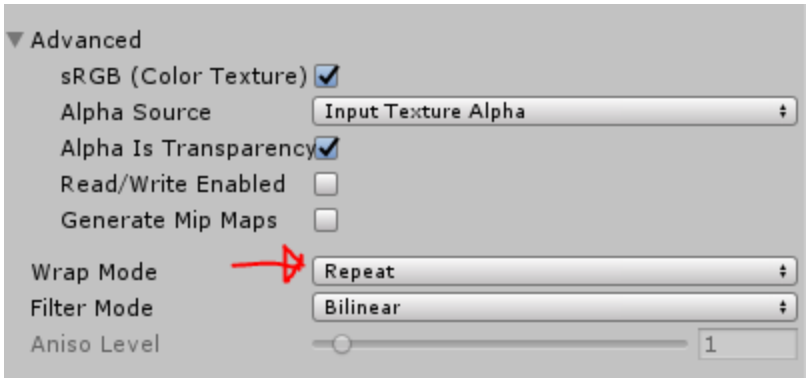


Notice how sprites have a generous spacing between them.
Be aware that some effects won't display correctly unless you set up your atlas sprites properly, see next section to know how (Sprite Atlases).

3. You'll see that some effects have a Glow property. In order for these properties to have the desired effect you need to add Post Processing and Bloom to the Main Camera of your scene.

   If you don't know how this is done you can follow these videos:
   URP: https://youtu.be/ZJBw7sGG63g
   Built-in: https://youtu.be/pq5dTygcFVU

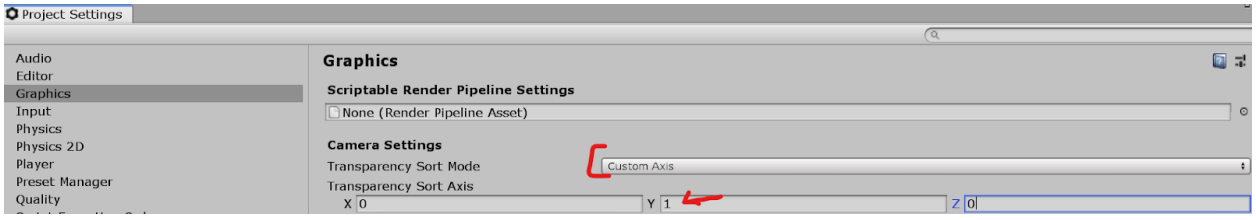
4. Most of the time it will be a good idea to set the Wrap Mode of the Import Settings of the textures you use to Repeat. This will assure a proper result when using effects that use scrolling textures



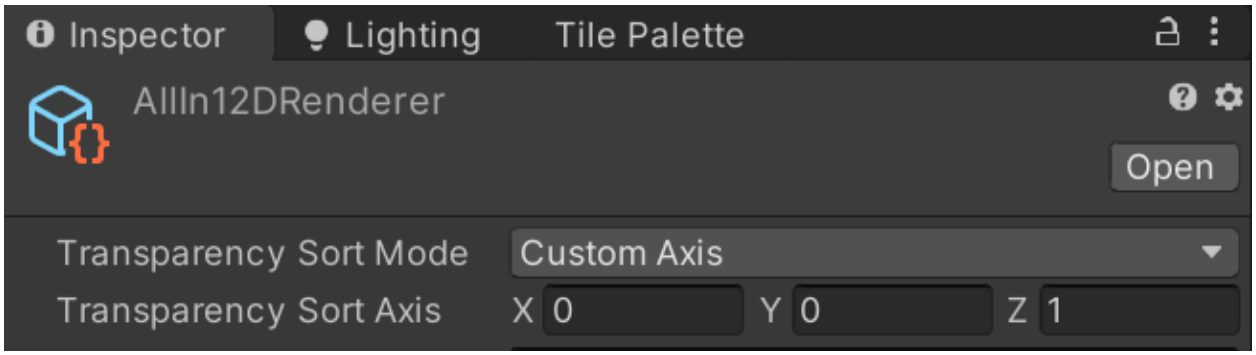
5. The same exact process applies to UI images

Custom Sort Axis

In 2D games we generally want to sort sprites along the Y axis. So we want to have objects that are below to render on top of the ones that are higher up.
This can be achieved by playing around with the Sprite Renderer Order in Layer values, but we can also configure Unity to sort sprites and objects along the Y axis automatically if we want to:



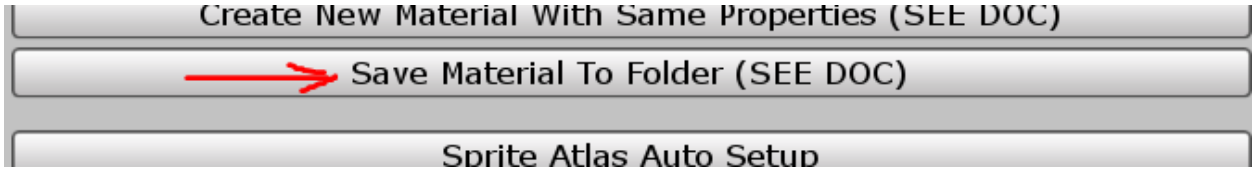
If you are using URP you can change this option in the Pipeline Renderer asset (if you set the sorting axis to the Z axis sprites will get sorted by depth, this is used in 2.5D and 3D games):



Saving Prefabs

By default this asset doesn't save the Material you are using, instead it keeps it as part of the Scene in order to avoid having too many objects cluttering your project. This means that by default, when you turn a GameObject with an AllIn1SpriteShader material into a prefab, the prefab won't render correctly since it doesn't have a reference to the Material inside the Project Asset files.

In order to save a Prefab you first need to save its Material. You can do so with the "Save Material to Folder" button that you'll find on the asset component:

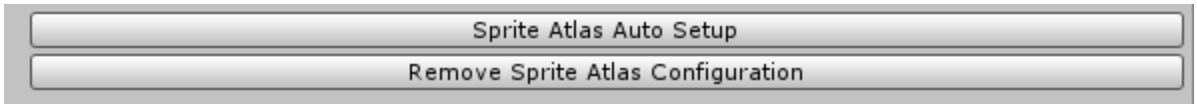


Sprite Atlases

You can also find a video about it here:
https://youtu.be/xzKYMUnmjnQ

By default if you use this tool with a sprite that's inside of a sprite atlas you'll see that the visual results that you obtain are not the ones that you expect. This is because the Sprite Renderer knows that it's a partial image (since it's inside an atlas) but the shader doesn't, which causes unexpected results.

To avoid this there are 2 buttons in the component editor to automatically add a component that will manage all of this for us:



With the Setup button the SetAtlasUvs component will be automatically added:

This will manage everything for us so we don't have to worry about manually doing anything. And in case we want to remove this setup we can press the second button and everything will go back to normal.

If we are using a sprite that won't be changing we can leave everything as is after pressing the Setup button but if we are using an animation that swaps the sprite (a flipbook) we'll need to check the Update Every Frame checkbox in the SetAtlasUvs component:

UI Images work the same but have a slight exception: for this to work every sprite inside the same Atlas must have a different material assigned. This can be achieved in several ways:

1. Adding the AllIn1SpriteShader component (this will create a fresh new material)
2. Duplicating an existing image gameObject and then pressing the New Clean Material or New Material with same properties button
3. Duplicating a saved material (Ctrl+D when selected) and assigning this new copy to the desired Image component

Finally keep in mind that there are 2 effects that don't really work with this feature, these are: Rect Size and Polar Coords.

How to animate effects

The custom material inspector properties can be animated through the Animation window as any other Unity component.

If you don't know how this is done you can follow this video:
https://youtu.be/aNastuqGBik

Please keep in mind that UI material properties can't be animated using the Animator, the reason being that Unity won't allow you to animate shared material properties. Unity UI Images materials are always shared, which means that all Images use the exact same material instance of a particular Image and therefore if a property is changed for one Image material all the other Images that share material will change too. Since Unity won't allow this behaviour it doesn't support using the Animator in UI Material properties. And unfortunately I can't do anything about it.

I recommend using an amazing free asset in the store called DoTween to animate the UI material properties through code or if you prefer you can use the function calls described in the following section.

Also consider that since material instances are shared you may want to create a copy of each material through script on an Awake method:

```
void Awake()
{
    Image uiImage = GetComponent<Image>();
    uiImage.material = new Material(uiImage.material);
}
```

Scripting

If you prefer avoiding animations or want to change properties through code you also have the possibility.

To do so you'll need to use the following Unity functions:

- Material.SetFloat: https://docs.unity3d.com/ScriptReference/Material.SetFloat.html
- Material.SetColor: https://docs.unity3d.com/ScriptReference/Material.SetColor.html
- Material.SetTexture: https://docs.unity3d.com/ScriptReference/Material.SetTexture.html

You can find all property names on AllIn1SpriteShader/Resources/AllIn1SpriteShader.shader. All properties are located from line 5 to 225 and can also be found at the Effects and Properties Breakdown section.

Here an example code snippet:

```
Material mat = GetComponent<Renderer>().material;
mat.SetFloat("_Alpha", 1f);
mat.SetColor("_Color", new Color(0.5f, 1f, 0f, 1f));
mat.SetTexture("_MainTex", texture);
```

*Note that there is an important distinction to be made between a “material” and a “sharedMaterial” of a Renderer. You shall use “material” if you only want to change a property of that instance of the material. And “sharedMaterial” if you want to change the property of all the instances of that material

Finally, there is an exception with materials used by a Masked UI Image where you’ll need to use “materialForRendering” instead. An example would be:

```
image.materialForRendering.SetFloat("_FadeAmount", t);
```

How to Enable/Disable Effects at Runtime

There are 2 ways of achieving this:

1. All effects have a property value combination that makes them look deactivated (usually by reducing the amount to 0, but it may vary depending on the effect). So the most clean way of deactivating and activating effects is by enabling all the effects you’ll use and then dynamically changing the property values either by animating the properties or by modifying the values by script as seen in the previous section.
2. This other way is less efficient, messier and will cause sprites to become invisible in the final build if you set a combination of effects that isn’t included in some other material in your project. So be warned, use this with caution and test it on the target platform. If sprites disappear at some point make sure to have some material in your project that includes the same set of effects than the sprite that isn’t showing.

   If you must use this feature I recommend having all the effects you’ll need enabled when you are in the editor and to disable them in the Start method. This will prevent any error in the final build.

   This method consists on enabling and disabling the shader compilation flags at runtime, so Unity will compile and replace the shader at runtime. To do so you first need to have a reference to the material and then use the Enable/Disable Keyword method like so:

```
Material mat = GetComponent<Renderer>().material;
…
mat.EnableKeyword("GRADIENT_ON");
mat.DisableKeyword("GRADIENT_ON");
```
(Keyword names of every effect can be found at the Effects and Properties Breakdown section)

Random Seed

Video about it: https://youtu.be/VIXgACEVQDo

There are some effects with a random component (such as flicker, hologram, glitch...). By default if you have 2 instances of a material that uses these effects they will look the same and it may look bad in your game. To avoid repetition you can add the RandomSeed component to the Object that contains the Renderer (Sprite, Tilemap, Sprite Shape, Particle System, Mesh) or UI Image you want to affect.

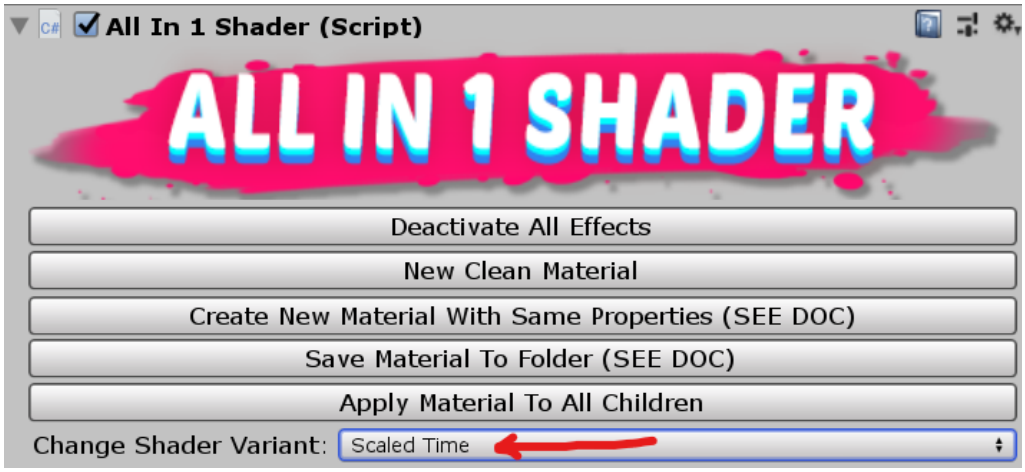
Scaled Time

Video about it: https://youtu.be/7_BgglufV-w

The default shader of the asset uses the built in Unity "_Time" shader property. This property doesn't support regular Scaled Time, which means that when the game gets paused the animated effects will keep moving (shaders use Unscaled Time by default).

If you need the effects to stop moving when the game is paused the asset contains an Scaled Time shader that has a global time property that can be set from a script. There's a SetGlobalTime.cs script included that will send the time every frame, but feel free to modify it to fit your needs.

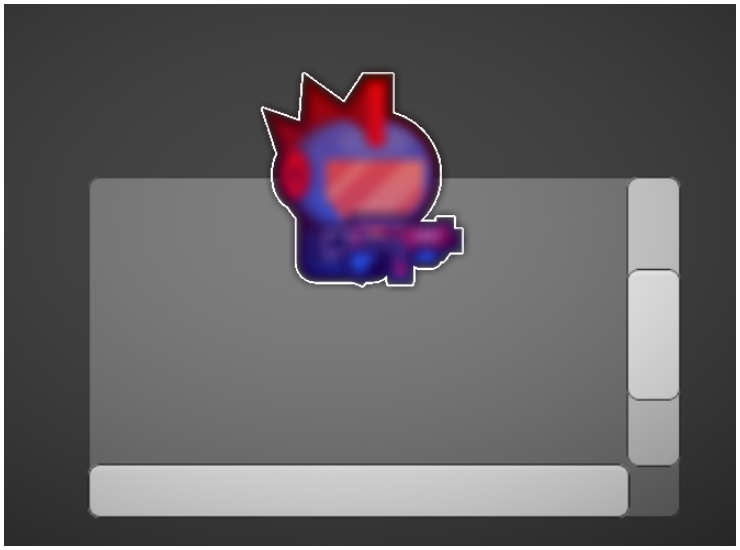
To use this feature you'll need to create a Material that uses this shader and assign it to the sprites you want to pause or use the dropdown in the asset component:



***From Unity version 2019.4 onwards shader "_Time" functions in the literal opposite way. The time will get scaled and in case you need an unscaled time you'll need to modify SetGlobalTime.cs and change Time.time in line 19 to Time.unscaled**

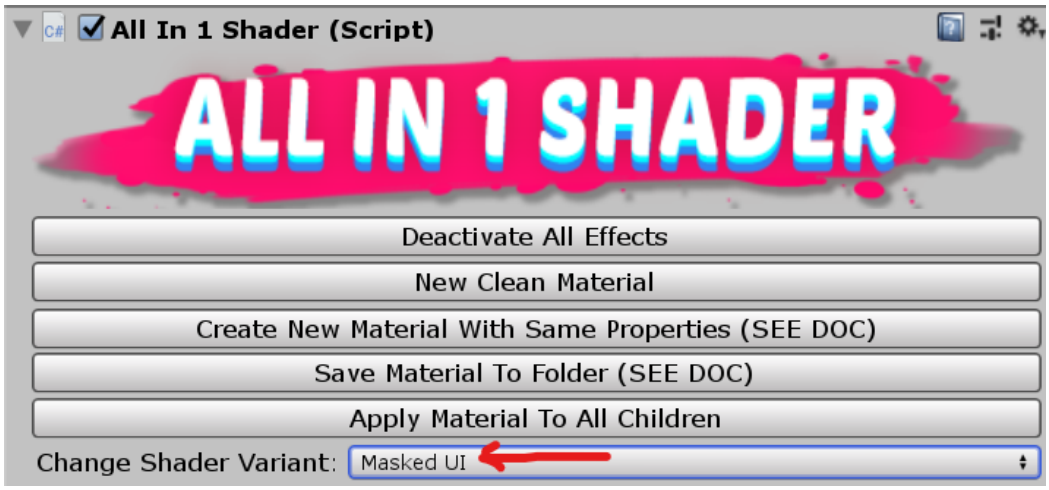
UI Masking

Video about it: https://youtu.be/X8mb93B6Xq4
If you try to use the asset material on a masked UI element such as the ones contained on a Scroll view you'll see that the object won't get masked.

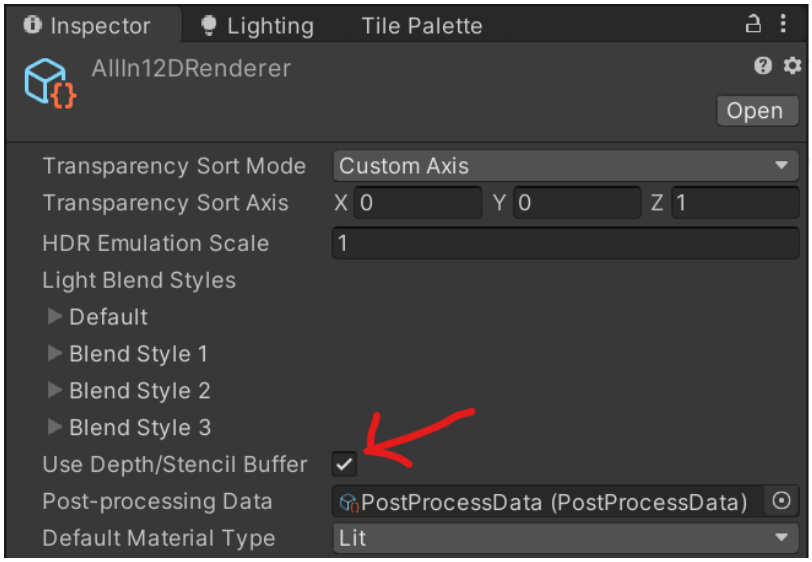


In this image we can see how we have a UI image inside of a Scroll View and it's not getting masked (it should only be visible inside the grey area). This happens because elements that need to be masked have a unique Stencil configuration, this configuration is not included on the default material since it would bring problems to the regular use cases.

But you can change the shader variant with the shader component and the Image will get masked:



***For this to work the Stencil buffer must be enabled. In the Built-In pipeline is enabled by default in most platforms. In case you are building for Android you can go to Player Settings, Resolution and Presentation and make sure that the Stencil isn't disabled.**
In URP you need to enable the stencil in the Pipeline Settings asset:



URP 2D Lights

In order to get access to these features you'll first need to import the "URP_Import_All1Sh" Unity Package that you'll find in the root of the asset. To do so just double click it and Import the files.

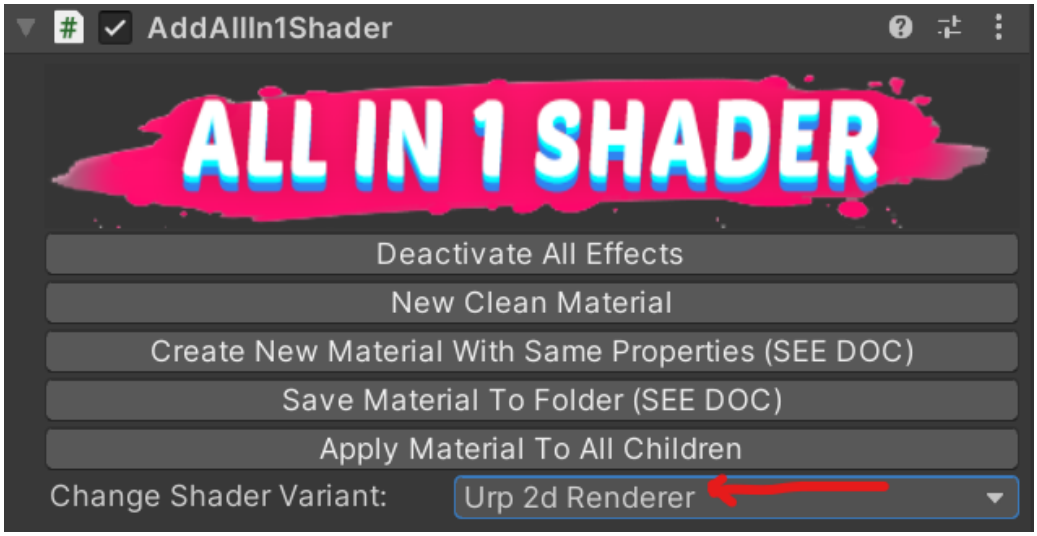
Once you have the files imported please make sure to follow the steps previously described in the First Steps section:
1. Import the "URP_Import_All1Sh" Unity Package
2. Set Color Space to Gamma in Player Settings, Other Settings, Color Space
3. In Graphics Settings set the Scriptable Render Pipeline Settings to "AllIn1UrpAsset"
3. In Quality Settings set the Rendering asset to "AllIn1UrpAsset"

You can also watch these 2 short videos here:
https://youtu.be/e7jyq-MXLEo (URP Renderer Setup)
https://youtu.be/ZJBw7sGG63g (URP Post Processing)

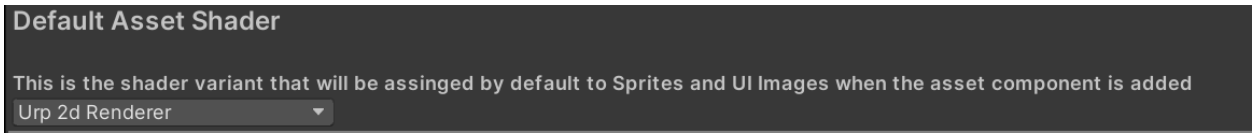
And here you can find a video showcasing this new feature:
https://youtu.be/uo-oD4NIVO8

The best way of seeing what you can do with these features is to take a look at the URP demo scenes inside the URP folder. In particular the DemoLighting one.

If you want to assign a Urp 2D Renderer shader to any particular sprite the easiest way to do so is by adding the asset component asset and changing the shader variant:



If most of the assets you use will use this variant and you don't want to manually change the shader variant every time you can change the default shader variant in the asset window (Window -> AllIn1ShaderWindow):

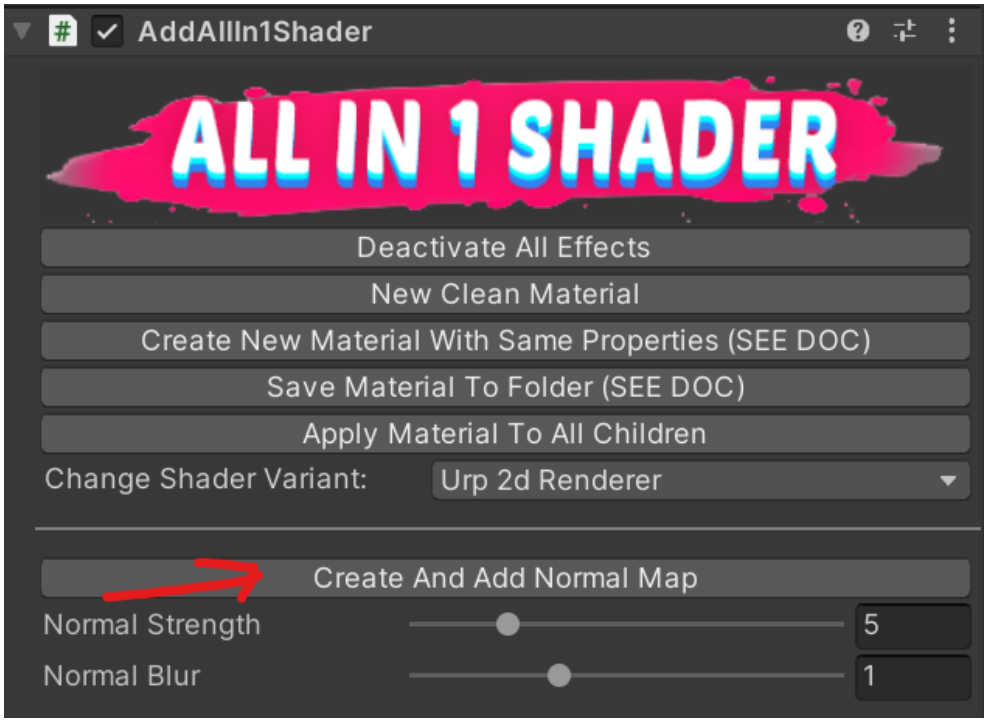


You can add a normal map that affects how the sprite is lit in the Material Inspector:

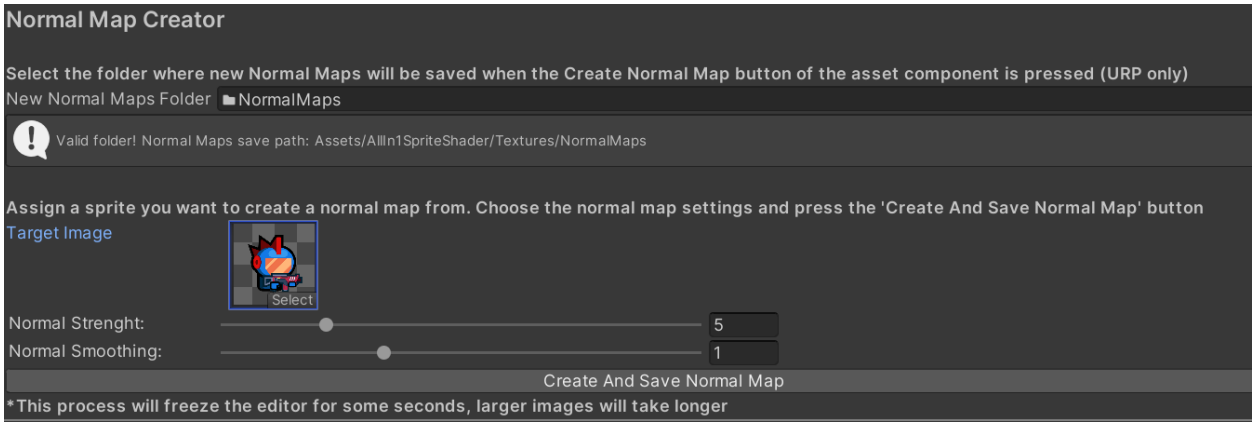
You can create the Normal Map externally or you can use a normal map creator included in the tool. You can use these 2 methods:

1. When the Urp 2D Renderer shader variant is being used you can automatically

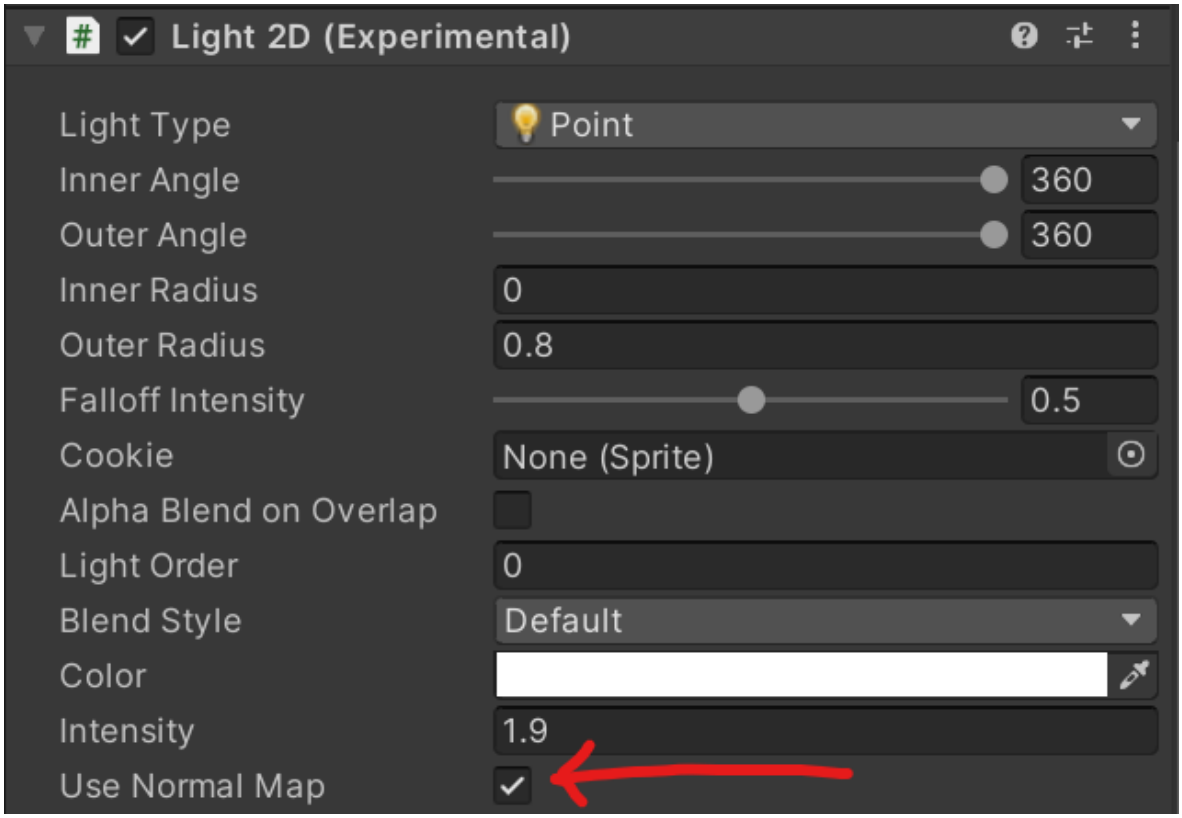
create and add the normal map to the sprite but pressing the Create And Add Normal Map button:



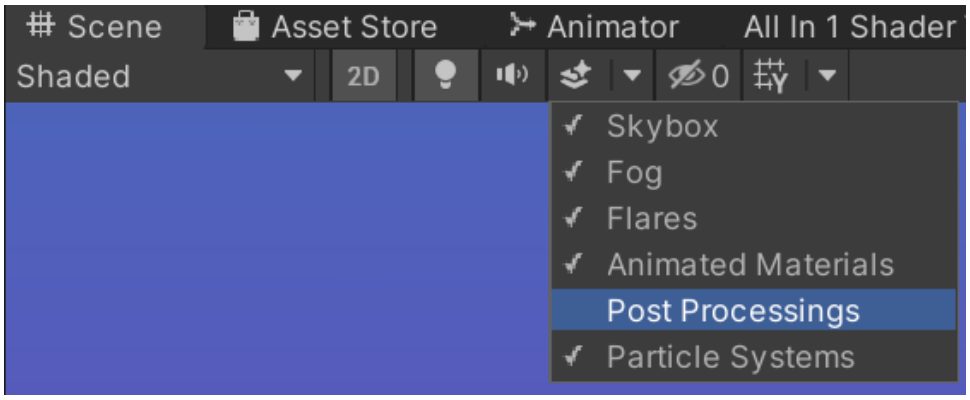
2. You can also use the asset Window to use this feature without needing a sprite+material+asset component combo (more info about this feature in the Asset Window section):



Keep in mind that the normal maps will only be taken into account if the Use Normal Map checkbox of the 2D Light is set to true:



To end this section I'd like to mention a very annoying problem/bug that exists with the URP Volumes post processing Bloom and the 2D Renderer in the Scene view (everything works perfectly fine in the Game view and in the final build). If you have Bloom active you may get very annoying crazy flickering colors when you move the Scene camera. I haven't found any solution and I hope that Unity fixes it soon. For now the best solution is to disable Post Processing in the scene tab:



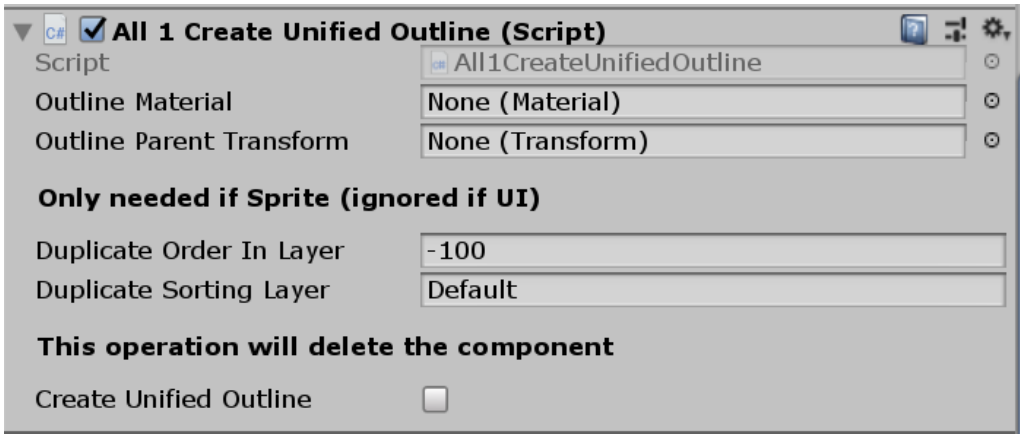
This feature is very new, if you encounter any problem, question or have any suggestion please reach out and I will quickly look into it: seasidegamestudios@gmail.com

Unified Outline

A component was created (All1CreateUnifiedOutline) to automate the sprite duplication task that is usually used to create a Unified Outline. This effect is achieved by adding an additional sprite behind each original sprite we want to outline:

(Example available in the Demo scene)

With the All1CreateUnifiedOutline component we can do that automatically by adding it to the parent gameobject of the hierarchy we want to outline:



To use it we need to add an Outline Material to the first slot. We can do so by saving a material with the desired outline (see Component Features if you don't know how). The Outline Parent Transform is optional. If added, all the new outline gameobjects will be

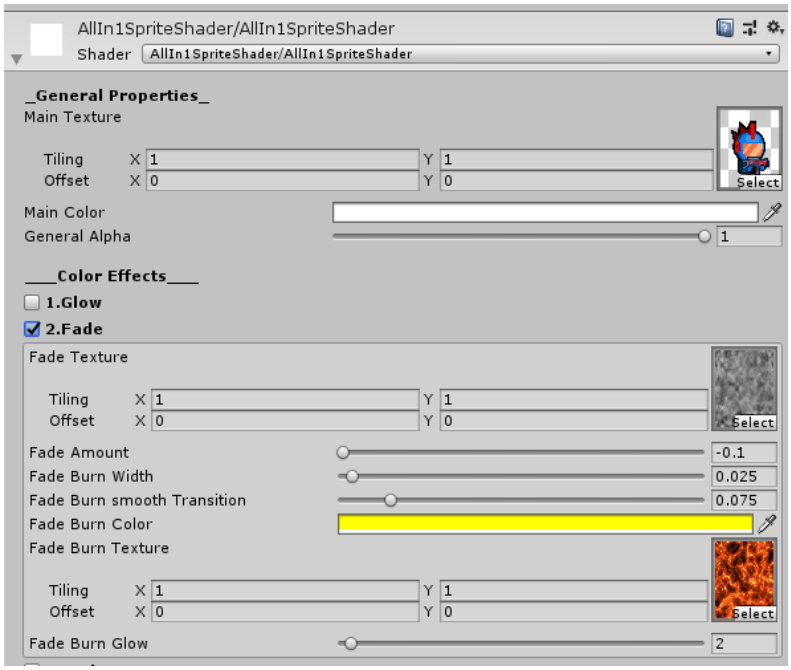
placed under this transform, otherwise the duplicates will be placed under the gameobject they've been duplicated from.

The next 2 properties are used to set the order in layer and sorting layer of the outline sprites. We'll want to place them behind the original sprite. But it's up to you what order in layer and sorting layer they use.

To create the Unified Outline press the Create Unified Outline checkbox. This will set up everything for you and will then delete the component.

Effects and Properties Breakdown

The AllIn1SpriteShader has a custom Material Inspector that allows you to activate and deactivate effects. When an effect is activated it displays it's properties so that they can be modified:



This is the custom Material Inspector. In this image we can see that the Fade effect is activated and all its properties.

Keep in mind that there's an example of every effect on the Demo scene.

Down below all the shader properties are explained. In between [] (ex:[GLOW_ON]) you can find the shader keyword name of each effect (see How to Enable/Disable Effects at Runtime section to see how to use it). In between () (ex: _MainTex) you can find the shader property names in case you want to modify then in a script (see Scripting section).

- General Properties
    - Main Texture (_MainTex): Main Texture, supports Tiling and Offset
    - Main Color (_Color): The Tint of the the Main Texture
    - General Alpha (_Alpha): Transparency of the end result

- Color Effects

1. Glow (Needs Post Processing Bloom to work as intended) [GLOW_ON]
    a. Glow Color (_GlowColor): Color of the Glow
    b. Glow Intensity (_Glow): Indicates how much the sprite will glow
    c. Glow Texture used? (_GlowTexUsed): When checked shows the following property
    d. Glow Texture (_GlowTex): Acts as a mask. The glow will only be applied where the alpha of this texture is greater than 0
2. Fade [FADE_ON]
    a. Fade Texture (_FadeTex): Maps how the fade will be made. The fade will be made from black to white
    b. Fade Amount (_FadeAmount): How much fade to apply. 0 is no fading and 1 is completely faded
    c. Fade Burn Width (_FadeBurnWidth): Size of the burned edge. Can be set to 0 to have no burned edge
    d. Fade Burn Smooth Transition (_FadeBurnTransition): How sharp the burned edge is
    e. Fade Burn Color (_FadeBurnColor): Tint of the burned edge
    f. Fade Burn Texture (_FadeBurnTex): Texture of the burned edge
    g. Fade Burn Glow (_FadeBurnGlow): How much the burned edge glows (needs Bloom in the scene)
3. Outline (when the width is large it doesn't look good) [OUTBASE_ON]
    a. Outline Base Color (_OutlineColor): Tint of outline color
    b. Outline Base Alpha (_OutlineAlpha): Transparency of the outline
    c. Outline Base Glow (_OutlineGlow): How much the outline glows (needs Bloom in the scene)

d. Outline Base High Resolution (_Outline8Directions): When toggled the outline has double the resolution and looks smoother on corners. It's more expensive computation wise
e. Outline Base is Pixel Perfect (_OutlineIsPixel): When toggled the outline width increases pixel by pixel (ideal for pixel art games)
f. Outline Width (_OutlineWidth/_OutlinePixelWidth): How thick the outline is
g. Outline uses texture (_OutlineTex): Toggles if the outline has a texture overlay or not (is affected by Outline Base Color)
h. Outline Texture Scroll Speed (_OutlineTexXSpeed / _OutlineTexYSpeed): Scroll speed of the outline texture in the X axis and Y axis
i. Outline Texture is Greyscale (_OutlineTexGrey): When toggled the outline texture will be greyscaled
j. Outline uses distortion (_OutlineDistortToggle): When toggled distortion will be applied to the outline and the outline distortion properties will be shown
k. Outline Distortion Texture (_OutlineDistortTex): Noise texture that determines how the distortion is done
l. Outline Distortion Amount (_OutlineDistortAmount): How much is the outline distorted following the distortion texture
m. Outline Distortion Scroll Speed (_OutlineDistortTexXSpeed / _OutlineDistortTexYSpeed): Scroll speed of the distortion texture in the X axis and Y axis

4. Gradient [GRADIENT_ON]
    a. Radial gradient?: When it's not checked it will use this regular linear gradient
    b. Gradient Blend (_GradBlend): How much of the gradient we show. 0 means the gradient will be fully transparent and 1 means that it will be fully visible
    c. Gradient Colors (_GradTopLeftCol /_GradTopRightCol / _GradBotLeftCol / _GradBotRightCol in order of appearance): The color of each corner of the gradient. Colors will be automatically blended together
    d. Boost parameters (_GradBoostX and _GradBoostY): Biases the gradient in the X and Y axis respectively
5. Radial Gradient [RADIALGRADIENT_ON]:
    a. Radial gradient?: When checked it will use this radial gradient
    b. Top Color (_GradTopLeftCol): The color in the outer part of the radius
    c. Bot Color (_GradBotLeftCol): The color in the inner part of the radius
    d. Boost X (_GradBoostX): Biases the gradient towards the Top or Bot color depending on its value

6. Color Swap (needs a color swap texture to work) [COLORSWAP_ON]
    a. Color Swap Texture (_ColorSwapTex): This texture must contain pure red, blue and green sections. This sections will then be recolored with whatever color we choose in the following properties
    b. Color Swap Red Channel (_ColorSwapRed): New color of the red parts of the Color Swap texture
    c. Color Swap Red Luminosity (_ColorSwapRedLuminosity): How bright the Red Channel Color will be
    d. The green and blue properties work like the red channel properties
7. Hue Shift [HSV_ON]
    a. Hue Shift (_HsvShift): How much the colors will be shifted
    b. Hue Shift Saturation (_HsvSaturation): Saturation of the hue shift result
    c. Hue Shift Bright (_HsvBright): Brightness of the hue shift result
8. Change 1 Color [CHANGECOLOR_ON]
    a. Tolerance (_ColorChangeTolerance): How similar to the Color to Change we need to be in order to change to the new color
    b. Color to Change (_ColorChangeTarget): This is the color that the shader will look for. The shader will change the pixels that are similar to this color by the New Color
    c. New Color (_ColorChangeNewCol): The result color of the pixels that are similar to Color to change
9. Color Ramp [COLORRAMP_ON]
    a. Color Ramp Texture (_ColorRampTex): This texture will be the new color palette of the sprite
    b. Color Ramp Luminosity (_ColorRampLuminosity): Extra of luminosity used to shift the color palette to where you want
    c. Color Ramp Affects Outline (_ColorRampOutline): When checked the color ramp will also affect the outline
10. Hit Effect [HITEFFECT_ON]
    a. Hit Effect Color (_HitEffectColor): The tint of the effect
    b. Hit Effect Glow (_HitEffectGlow): Glow of the effect. Needs Bloom in the scene
    c. Hit Effect Blend (_HitEffectBlend): How much of the effect is shown. When set to 0 it’s not shown, when set to 1 it shows fully. This is meant to be animated to get cool looking results
11. Negative [NEGATIVE_ON]
    a. Negative Amount (_NegativeAmount): How much of the negative effect we want to show
12. Pixelate [PIXELATE_ON]

    a. Pixelate size (_PixelateSize): The lower the number the more pixelated the sprite gets. This effect looks bad with combined with distortions
13. Greyscale [GREYSCALE_ON]
    a. Greyscale Luminosity (_GreyscaleLuminosity): Make sprite whiter
    b. Greyscale Affects Outline (_GreyscaleOutline): When checked the greyscale will also affect the outline
    c. Greyscale Tint Color (_GreyscaleTintColor): Tint of the greyscale
14. Posterize [POSTERIZE_ON]
    a. Posterize Number of Colors (_PosterizeNumColors): The higher the number the more different colors the sprite will display
    b. Posterize Amount (_PosterizeGamma): The higher the number the more different the posterize colors will be
    c. Posterize Affects Outline (_PosterizeOutline): When checked the posterize will also affect the outline
15. Blur (won't affect the outline) [BLUR_ON]
    a. Blur Intensity (_BlurIntensity): How much the sprite is blurred
    b. Blur is low res (_BlurHD): When active an alternative and less expensive (computation wise) blur version is used
16. Motion Blur [MOTIONBLUR_ON]
    a. Motion Blur Angle (_MotionBlurAngle): The direction of the motion blur
    b. Motion Blur Distance (_MotionBlurDist): The amount of motion blur
17. Ghost [GHOST_ON]
    a. Ghost Color Boost (_GhostColorBoost): How white the ghost effect gets
    b. Ghost Transparency (_GhostTransparency): How transparent the effect gets
18. Inner Outline (places outlines over the Main Texture) [INNEROUTLINE_ON]
    a. Inner Outline Color (_InnerOutlineColor): Color of the Inner Outline
    b. Inner Outline Thickness (_InnerOutlineThickness): How thick the Inner Outline is
    c. Inner Outline Alpha (_InnerOutlineAlpha): How transparent the Inner Outline is
    d. Inner Outline Glow (_InnerOutlineGlow): How much the Inner Outline glows (needs Bloom in the scene)
19. Hologram [HOLOGRAM_ON]
    a. Hologram Stripes Amount (_HologramStripesAmount): How much uv space does the hologram stripes take
    b. Hologram Unchanged Amount (_HologramUnmodAmount): How much uv space does the unchanged parts take. These parts will be placed in between the stripes

c. Hologram Stripes Speed (_HologramStripesSpeed): How fast the stripes scroll
d. Hologram Min Alpha (_HologramMinAlpha): The minimum alpha of the stripes parts. The stripe parts will fade the alpha from this number to max alpha. This can be used to set any alpha gradient
e. Hologram Max Alpha (_HologramMaxAlpha): The maximum alpha of the stripes parts. The end value of the alpha gradient. If it's value it's bigger than 1 it will make the stripe part glow
f. Hologram Stripe Color (_HologramStripeColor): Tint of the hologram stripe

20. Chromatic Aberration [CHROMABERR_ON]
    a. ChromaticAberr Amount (_ChromAberrAmount): How visible the effect is
    b. ChromaticAberr Alpha (_ChromAberrAlpha): How transparent the effect is
21. Glitch [GLITCH_ON]
    a. Glitch Amount (_GlitchAmount): The higher the number, the more intense the effect gets
    b. Glitch Size (_GlitchSize): The higher the number, the smaller the glitch squares get
22. Flicker [FLICKER_ON]
    a. Flicker Percent (_FlickerPercent): The percentage of time the sprite is invisible (from 0 to 1)
    b. Flicker Frequency (_FlickerFreq): How often does the flicker happen
    c. Flicker Alpha (_FlickerAlpha): How transparent the sprite is when it flickers
23. Shadow [SHADOW_ON]
    a. Shadow X Axis (_ShadowX): Shadow position offset on the X axis
    b. Shadow Y Axis (_ShadowY): Shadow position offset on the Y axis
    c. Shadow Alpha (_ShadowAlpha): Transparency of the shadow
    d. Shadow Color (_ShadowColor): Tint of the shadow
24. Shine [SHINE_ON]
    a. Shine Mask (_ShineMask): None transparent parts of this mask texture will allow the shine to show
    b. Shine Color (_ShineColor): Color tint of the shine effect
    c. Shine Location (_ShineLocation): Decides the position of the shine line. 0.5 is the center. 0 is one end and 1 is the other end. Animate this value to get a nice scrolling shine effect
    d. Shine Rotate: Rotation of the shine line in radians
    e. Shine Width (_ShineWidth): How wide the shine line is
    f. Shine Glow (_ShineGlow): How much does the shine line glow (benefits from post processing Bloom)
25. Alpha Cutoff [ALPHACUTOFF_ON]

a. Alpha cutoff value (_AlphaCutoffValue): Pixels that are more transparent than this value are not drawn. This is useful to make more cartoon looking effects and to discard unwanted transparencies from certain effects

26. Alpha Round [ALPHAROUND_ON]
    a. Round Threshold (_AlphaRoundThreshold): Rounds the alpha value depending on this threshold. The values above this value turn into 1 and the ones below into 0

- UV Effects

27. Hand Drawn [DOODLE_ON]
    a. Hand Drawn Amount (_HandDrawnAmount): How much of a distortion we apply to make it look hand drawn frame a frame
    b. Hand Drawn Speed (_HandDrawnSpeed): How often we distort the sprite
28. Grass Movement / Wind [WIND_ON]
    a. Grass Speed (_GrassSpeed): How fast it moves from side to side
    b. Bend Amount (_GrassWind): How much the sprite bends
    c. Radial Bend (_GrassRadialBend): How much the sprite twists. The top part of the sprite will twist with the wind the higher this number is
    d. Grass is manually animated (_GrassManualToggle): When checked the sprite won't move on its own (useful if you want a sprite to interact with the player)
    e. Grass manual anim (_GrassManualAnim): If the previous property is checked this property dictates how the sprite bends. -1 means fully bent to the left and 1 fully bent to the right
29. Wave [WAVEUV_ON]
    a. Wave Amount (_WaveAmount): How many waves we make
    b. Wave speed (_WaveSpeed): How fast the wave scrolls across the sprite
    c. Wave Strength (_WaveStrength): How much the wave affects the sprite
    d. Wave X Axis (_WaveX): Position of the wave origin on the X axis (0 is left 1 is right)
    e. Wave Y Axis (_WaveY): Position of the wave origin on the Y axis (0 is bottom 1 is top)
30. Round Wave [ROUNDWAVEUV_ON]
    a. Round Wave Strength (_RoundWaveStrength): How much the wave affects the sprite
    b. Round Wave Speed (_RoundWaveSpeed): How fast the wave scrolls across the sprite
31. Rect Size [RECTSIZE_ON]

    a. Rect Size (_RectSize): Size of the mesh where the sprite is drawn. Set Scene to “Shaded Wireframe” shading mode to see the mesh shape
32. Offset [OFFSETUV_ON]
    a. Offset X axis (_OffsetUvX): Offset of the sprite on the X axis
    b. Offset Y axis (_OffsetUvY): Offset of the sprite on the Y axis
33. Clipping (useful for sliders) [CLIPPING_ON]
    a. Clipping Left (_ClipUvLeft): How much of the image we clip from left to right
    b. Clipping Right (_ClipUvRight): How much of the image we clip from right to left
    c. Clipping Up (_ClipUvUp): How much of the image we clip from up to down
    d. Clipping Down (_ClipUvDown): How much of the image we clip from down to up
34. Texture Scroll [TEXTURESCROLL_ON]
    a. Texture Scroll Speed X (_TextureScrollXSpeed): Scrolling speed on the X axis
    b. Texture Scroll Speed Y (_TextureScrollYSpeed): Scrolling speed on the Y axis
35. Zoom [ZOOMUV_ON]
    a. Zoom Amount (_ZoomUvAmount): How much the sprite is zoomed
36. Distortion [DISTORT_ON]
    a. Distortion Texture (_DistortTex): Noise texture that determines how the distortion is done
    b. Distortion Amount (_DistortAmount): How much the image is distorted following the texture pattern
    c. Distortion scroll speed (_DistortTexXSpeed and _DistortTexYSpeed): Scroll speed of the distortion texture in the X axis and Y axis
37. Twist [TWISTUV_ON]
    a. Twist Amount (_TwistUvAmount): How much is the sprite twisted
    b. Twist Pos X Axis (_TwistUvPosX): Position of the center of the twist on the X axis (0 is left and 1 is right)
    c. Twist Pos Y Axis (_TwistUvPosY): Position of the center of the twist on the Y axis (0 is bottom and 1 is top)
    d. Twist Radius (_TwistUvRadius): The radius of the twist effect
38. Rotate [ROTATEUV_ON]
    a. Rotate Angle (_RotateUvAmount): Indicates in radians the angle of rotation of the sprite texture
39. Polar Coordinates [POLARUV_ON]

    a. Transforms the uv coordinates into polar coordinates (this effect looks goods with tiling on the main texture + texture scrolling)
40. Fish Eye [FISHEYE_ON]
    a. Fish Eye Amount (_FishEyeUvAmount): How much fish eye distortion we want to apply
41. Pinch [PINCH_ON]
    a. Pinch Amount (_PinchUvAmount): How much pinch effect we want to apply
42. Shake [SHAKEUV_ON]
    a. Shake Speed (_ShakeUvSpeed): How fast it shakes
    b. Shake X Multiplier (_ShakeUvX): The higher the value the more it will move on the X axis while shaking
    c. Shake Y Multiplier (_ShakeUvY): The higher the value the more it will move on the Y axis while shaking

- Lighting Properties (Urp 2D Renderer shader)
    a. Lighting Mask: A greyscale texture that indicates where the sprite should be affected by light. White means affected by light and black means unlit
    b. Lit Amount: How lit the sprite is. 1 is fully lit and 0 is not affected by light
    c. Normal Map: Normal map texture, affects how the sprite is lit
    d. Normal Strength: The higher this value gets the more pronounced the normal map effect gets (very high values tend to look bad)

Considerations

The shader that the material uses compilations flags to enable and disable the code of the different effects. So only the effects that you enable will be taken into account by the GPU and therefore only those parts will be computed.
The shader code is also fast, uses as less memory as possible and has no conditionals. That being said, having all these properties in the shader has a slight GPU overhead. But there's nothing to worry about unless your game displays a huge amount of different materials at the same time since Unity will batch instances of the same material and sprite into a single draw call.

Running out of Shader Keywords

If you are using other assets or if you've written some complex shaders yourself you may run out of shader Keywords. Unity hass 256 possible global Keywords for shaders, Unity itself takes around 60 of them, so the user has around 190 available Keywords. This asset uses many Keywords, so running out of them may be a possibility if you are using other assets.

So what's the solution? Since Unity 2019.1 Unity has included local Keywords. This asset is prepared to work with any Unity version and that's why these local Keywords aren't used. But if you are on Unity 2019.1 onward this is what you can do:

1. Go to: AllIn1SpriteShader/Resources
2. There you'll see the Default version, the UI Mask version and the Unscaled Time version
3. Open all of them or just the Default one if you don't use the others
4. Change all shader_feature for shader_feature_local (in visual studio ctrl+f will open the search and replace bar)