ALL IN 1 SPRITE LIGHTING

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 add good looking lighting plus a couple cool popular sprite 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 and looks without altering the setup of your sprites.

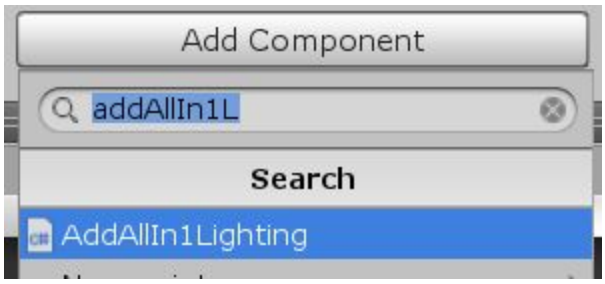
Link to the youtube playlist that explains how to use this asset:
https://www.youtube.com/playlist?list=PLKS0HUbkxp-lyRT1E2GzzEV0AX3CSTbFZ

Feel free to contact me over at this email if you have any issue, request, question or want to show off your work: seasidegamestudios@gmail.com

How to use

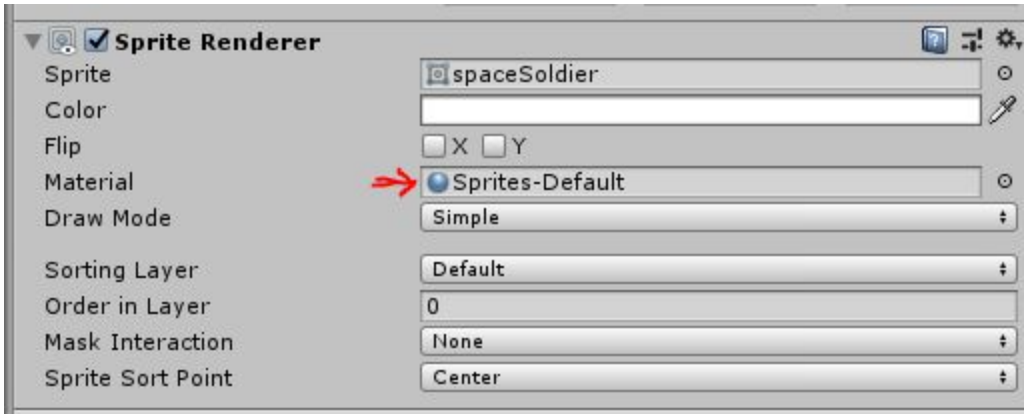
Here you have a link to a video that explains how to add the tool to your sprites in case you prefer a visual explanation:
https://www.youtube.com/watch?v=nUQdKZwedkw&list=PLKS0HUbkxp-lyRT1E2GzzEV0AX3CSTbFZ

The asset includes a component that will do all the setup for you. The component is called “AllIn1Lighting”:



When you add it, the component will swap the current material for a new instance of the AllIn1SpriteLighting material. The component also has some features that are overviewed in the next point.

But you can also do it the classic way: right clicking the AllIn1LightShader 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:



Component Features

This component will work with SpriteRenderers, MeshRenderers and UI Images.

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 AllInOneSpriteLighting material and assign it to the Sprite.
- **Create New Material With Same Properties**: It will create a new instance of the AllInOneSpriteLighting 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/AllIn1SpriteLighting/Materials”. This can be used to assign the same Material to many different Sprites.
- **Create and Add Normal Map**: A normal map for the current sprite will be computed and created. This new normal map will then be added to the AllIn1SpriteLight and the Normal Mapping effect will be enabled. If you want to know more details about this feature and how it works please go to the Normal Creator section.
- **Remove Component and Material**: Removes the component from the GameObject and sets the Sprite Material back to the Sprite/Default one.

Change Saving Paths

As mentioned in the previous section the Asset can save Materials and Normal Maps. By default these files will be saved into a pre assigned folder under the Asset root folder.

These paths can be changed through the AllIn1LightWindow that you can access by going to Window -> AllIn1LightWindow:

Deferred vs Forward Rendering

Since this is a Lighting oriented asset I'll quickly go over the 2 main Rendering Paths so you can better choose which one suits your project best:

1. Deferred: The final frame is rasterized into different image buffers without lighting. Then all pixels of these buffers are processed at the same time in order to calculate the lighting.
   The clear advantage of this method is that adding more light sources has no extra cost. But the disadvantage is that old hardware may not support it and that it requires a higher GPU bandwidth.
2. Forward: All the objects of the scene need to be checked against every light on the scene in order to calculate the lighting. Once all the lighting has been calculated, the scene is rasterized.
   This method is ideal for low end devices/platforms or when using a single light source (or very very few of them). The clear disadvantage is performance cost (it increases exponentially with the number of lights).

The takeaway is to use Deferred when using more than 1 light and to use Forward when only using 1 light.
Read more about the topic here:
https://docs.unity3d.com/Manual/RenderingPaths.html

Normal Creator

Here you have a link to a video that explains how to use the Normal Creator in case you prefer a visual explanation:
https://www.youtube.com/watch?v=v4JbkvDkP50&list=PLKS0HUbkxp-lyRT1E2GzzEV0AX3CSTbFZ

In order to use the Normal Creator you'll first need to add the AllIn1Lighting component (see How to Use section if you don't know how) to the game

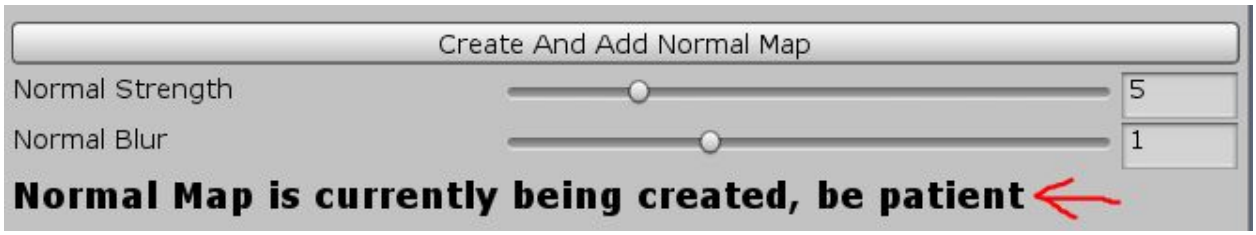
object that contains the Sprite Renderer that your targeted sprite is assigned to.

To create the normal map for the sprite press the “Create And Add Normal Map” button:

Pressing this button will create a normal map named like the current game object and save it on the following path:
AllIn1SpriteLighting/NormalMaps
The Normal Strength property dictates how pronounced the resulting normal map will be (0 is completely flat and 20 is maximum volume and detail). Keep in mind that Normal Strength can also be tweaked on the Material Inspector.

The Normal Blur property dictates how blurry the resulting normal map is. 0 means completely sharp and 5 means maximum blurriness.

Computing this Normal Map may take a few seconds. And the bigger the image, the longer it will take. Generally this won't be a problem except you use massive textures. If you go over 4k resolution things could get real slow. While the Normal Map is being created the Unity Editor will be blocked and this message will be displayed on the Inspector:



* The result of this automatic blurring method will never be as good as more "handcrafted" methods such as hand drawing the normal map or extracting it from specialised normal creating software

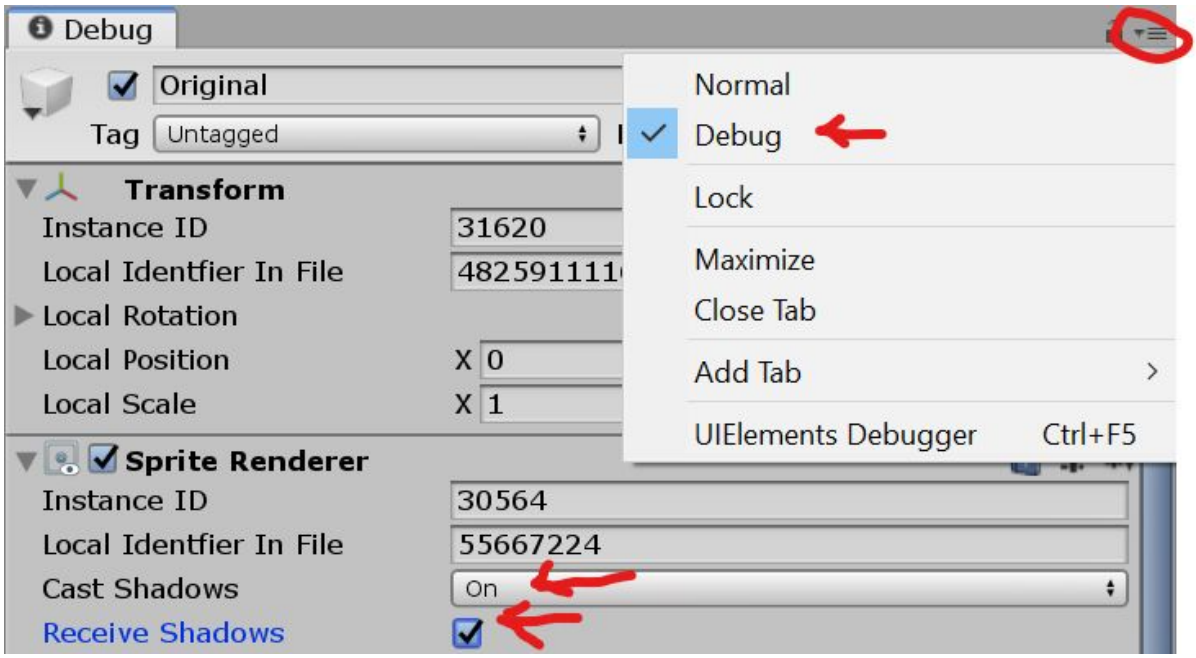
* Also if you prefer, and don't mind spending more time, there is a website that allows you to easily create a normal map and allows a more control than the asset automatic method:
https://cpetry.github.io/NormalMap-Online/

Shadows

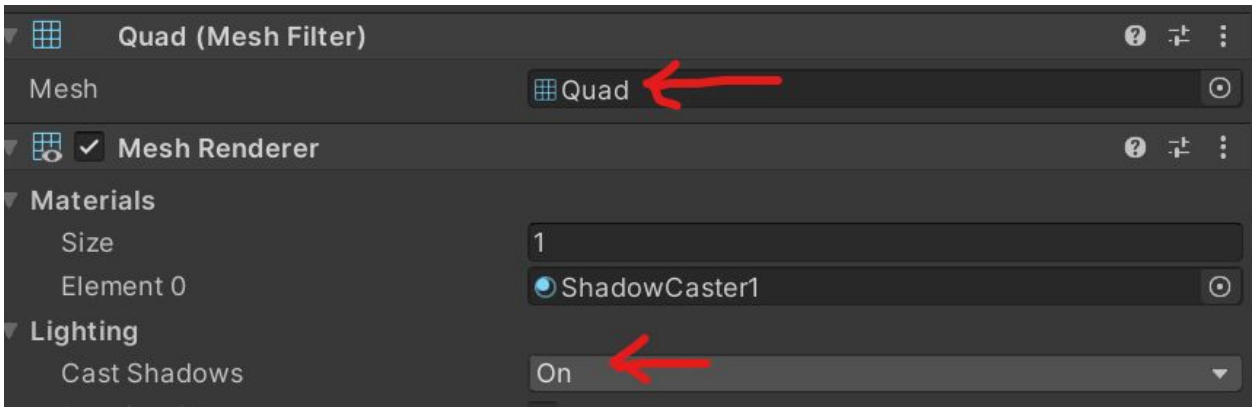
If you want to receive or cast shadows we can use a Sprite Renderer or a Mesh Renderer.

Sprite Renderer: Just add the asset component and everything will just work automatically. Otherwise you can enable the debug view of the

component and toggle the shadows:



Mesh Renderer: You'll need to set it up yourself, the component won't do it for you.
A typical Mesh Renderer setup will look like this:

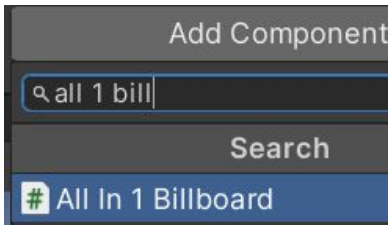


Usually we use a Quad (a flat square) to render our sprite on but any mesh would work too.

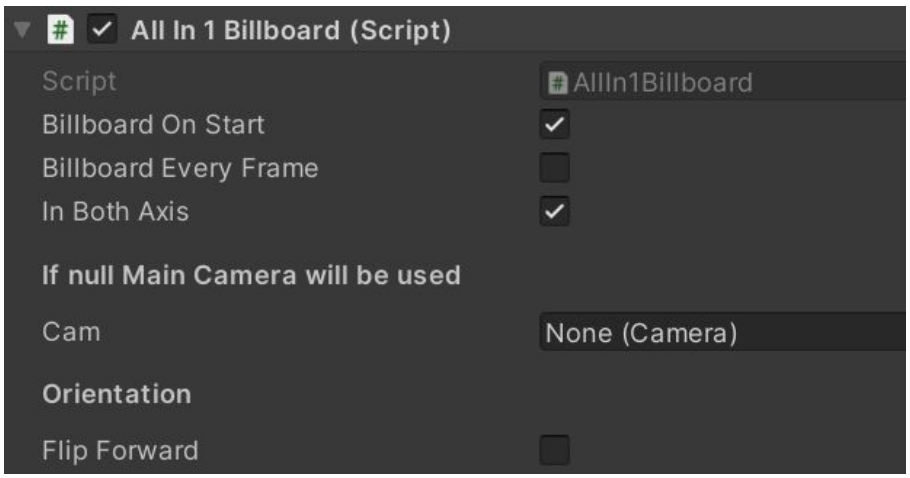
We can then add the asset component and everything will get setup for us and we can use the material inspector as if it was a regular sprite.

Sprite Billboarding

The asset includes a helper script to allow for all sorts of sprite billboarding:



And this is how the component looks once added:

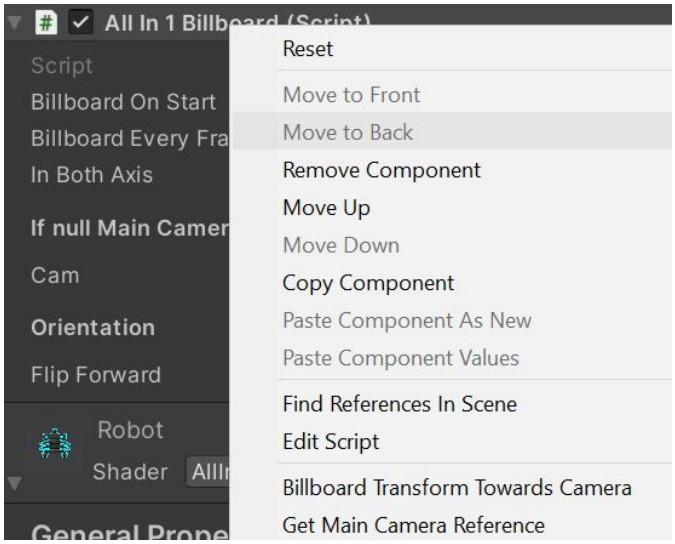


Let's go over its properties:

- Billboard On Start: If checked the object will get billboarded towards the camera on its Start method, so right when the scene is loaded or the object instanced
- Billboard Every Frame: If checked the object will get billboarded towards the camera on its Update method. It will always be facing the camera
- In both axis: If checked the object will totally face the camera. Otherwise, it will rotate around its Y axis
- Cam: A reference to the scene camera, there's no need to set this parameter unless you have more than 1 camera in the scene. If the reference is not set the script will look for the Main Camera (the result is cached, so we only search for the camera once)

- Flip Forward: When checked the billboard operation is flipped. This is useful when our sprite is black since it's facing the opposite direction than we expect to

Finally in the editor you can right click the component and this options will appear:



If you click one of the 2 options you can Billboard the object towards the camera in the editor or cache the Main Camera so we don't have to do it while running the game.

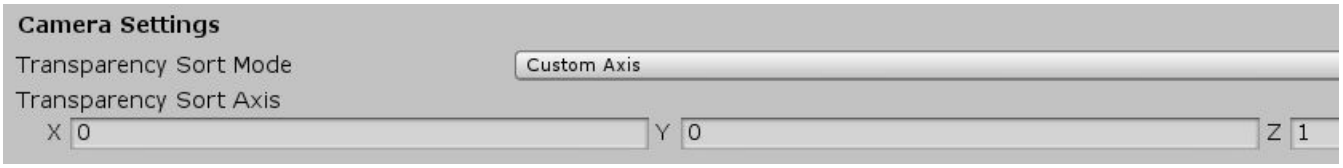
Sprite Sorting

When using this asset you may encounter with sprite sorting situations similar to this one:

The reason being that both sprites are on the same position on the Z axis and have the same Sorting Order. When this happens the light that affects both sprites gets added. To solve this you can either change the Sorting Order of one of them, change their position on the Z axis or avoid the overlap.

Also, when using this asset you may want to use a Perspective camera in order to get a nice depth effect by having sprites across different depth values on the Z axis (like on the Demo2 scene included on the asset).

When using this technique you'll probably want to change how the sprites are sorted. To do that you can go to Edit -> Project Settings -> Graphics and change the sorting axis like so:



GPU Instancing

In order to use GPU instancing and save some performance you shall use the same Material + Sprite combo. This will ensure that Unity batches all the instances of this Material + Sprite and render them at the same time.

Light Pixel Count

In some cases you'll see some weird light artifacts such as appearing/disappearing light effects. This would be probably due to the default Light Count.

Unity, in order to save some performance (on the default Forward rendering path) sets a limit of 4 active lights at the same time. So if you place many lights yourself and don't change this Light Count you may get some weird artifacts.

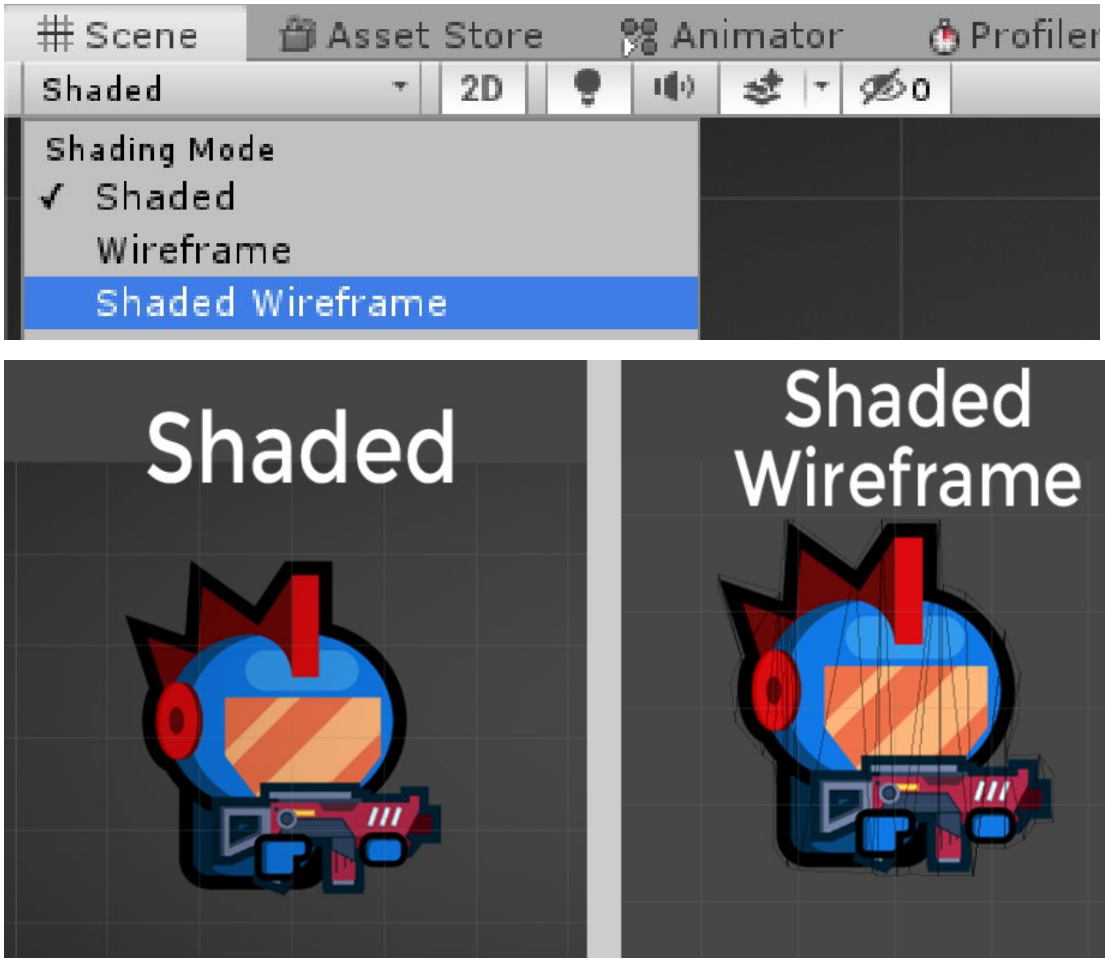
In order to change it go to Edit -> Project Settings -> Quality and change the Pixel Light Count to a higher value such as 20:

Textures Setup

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

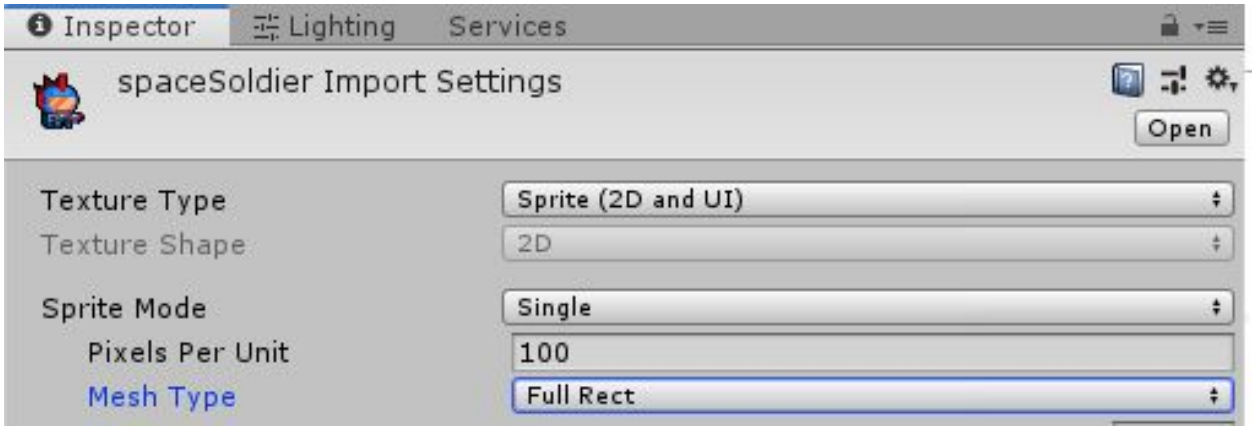
Here you have a link to a video that explains how to setup your textures in case you prefer a visual explanation:
https://www.youtube.com/watch?v=tS1Dt4JIpg8&list=PLKS0HUbkxp-lyRT1E2GzzEV0AX3CSTbFZ

1. The most important part is having 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:

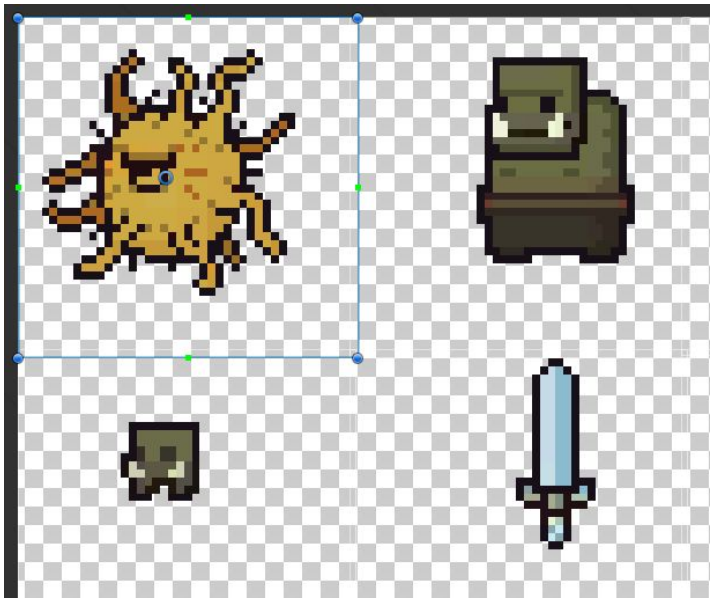


In the Shaded Wireframe view we can see black lines, that's the shape of the mesh where your sprite is being rendered. To get all effects to display properly we'll need more space.

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



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.

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 this video:

https://www.youtube.com/watch?v=2CICmkNQJ_4&list=PLKS0HUbkxp-lyRT1E2GzzEV0AX3CSTbFZ

Saving Prefabs

By default this asset doesn't save the Material you are using, instead it keeps it as part of the Scene. 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:

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://www.youtube.com/watch?v=9KqVkVpnQJk&list=PLKS0HUbkxp-lyRT1E2GzzEV0AX3CSTbFZ

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 AllIn1SpriteLighting/Resources/AllIn1SpriteLighting.shader. All properties are located from line 6 to 193 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

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.

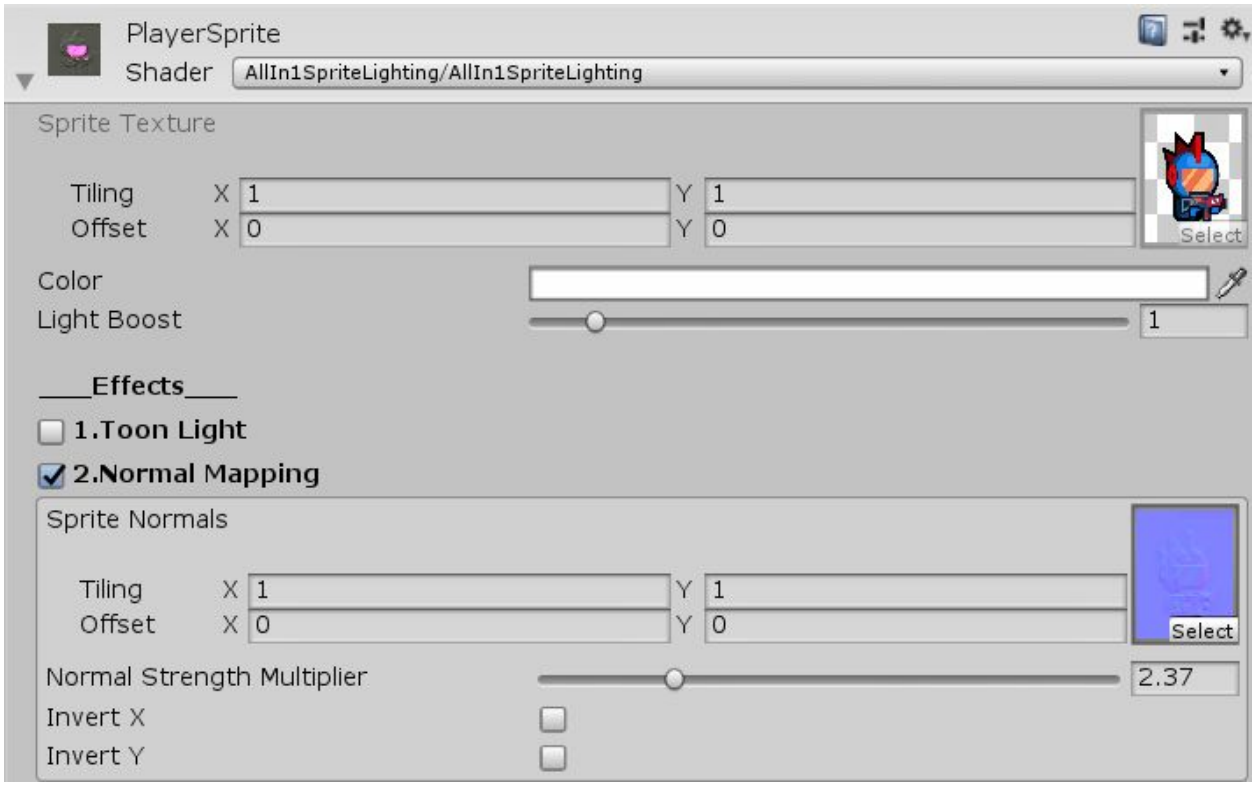
2. This other way is less efficient, messier and may not even work on some platforms. So be warned, use this with caution and test it on the target platform before committing to this solution.
   It consists on enabling and disabling the shader compilation flags at runtime, so Unity will compile and 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("OUTLINE_ON");
mat.DisableKeyword("OUTLINE_ON");
```

(Keyword names of every effect can be found at the Effects and Properties Breakdown section)

Effects and Properties Breakdown

The AllInOneSpriteLighting 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 Normal Mapping 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
  - Light Boost (_LightBoost): How much the light affects this sprite

- Alpha cutoff (_Cutoff): The shader doesn't support regular transparency, instead it discards pixels under a certain transparency threshold. This is this threshold. Any pixel more transparent than this value won't be rendered

- Effects

1. Toon Light [TOON_ON]
    a. Shadow Threshold (_ToonShadowThresh): The higher it is, the more light we'll need to light the sprite
    b. Shadow Smoothing (_ToonShadowSmooth): The higher it is, the less pronounced the shadow edge will be
    c. Specular Threshold (_ToonSpecThresh): The higher it is, the more light we'll for the specular spots to show (needs the Specular effect active in order to work)
2. Normal Mapping [NORMALMAP_ON]
    a. Normals Map (_NormalsTex): The texture of the normal map. It will be used to dictate how light affects the sprite
    b. Strength Multiplier (_NormalStrenght): The higher it is the more the bumpiness of the normal map texture will show
    c. Invert X / Invert Y (_NormalFlipX / _NormalFlipY): Inverts the direction of the normal map. This property is used to fix normal maps imported from other sources. Since there is no universal way of encoding a normal map
3. Specular
    a. Specular Boost (_SpecBoost): Intensity of the specular spots
    b. Specular Map (_SpecularTex): Dictates where the sprite is specular and where it's not. Whhite means fully specular, black means not specular.
4. Glow (Needs Post Processing Bloom to work as intended) [GLOW_ON]
    a. Glow Color (_GlowColor): Color of the Glow
    b. Glow Amount(_GlowAmount): Indicates how much the sprite will glow

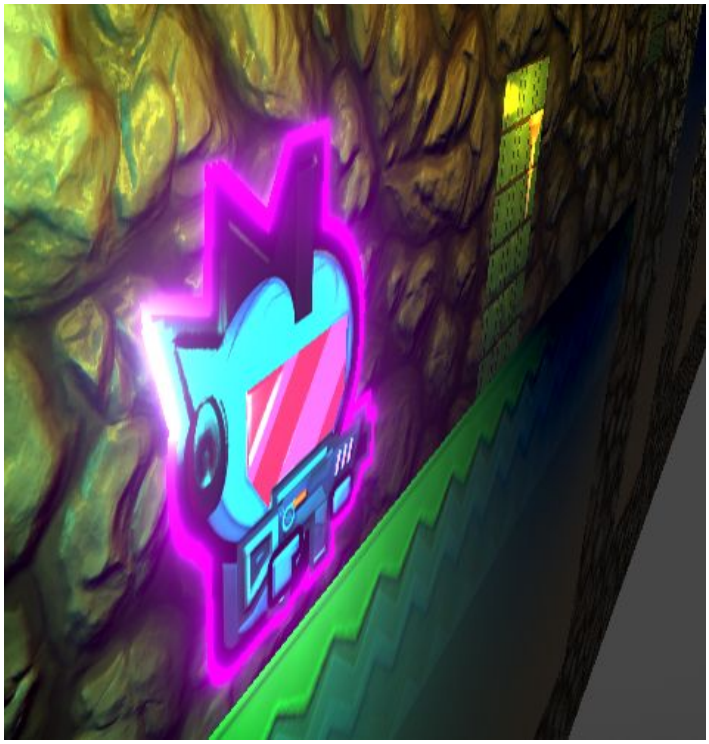
    c. Glow Texture (_GlowTex): Acts as a mask. The glow will only be applied where the alpha of this texture is greater than 0
    d. Glow Illumination (_GlowLit): Dictates if the glowing parts receive illumination or not
5. Outline (when the width is large it doesn't look good) [OUTBASE_ON]
    a. Outline Color (_OutlineColor): Tint of outline color
    b. Outline Width (_OutlineWidth): How thick the outline is
    c. Outline Base Alpha (_OutlineAlpha): Transparency of the outline
    d. Outline Base Glow (_OutlineGlow): How much the outline glows (needs Bloom in the scene)
    e. Outline Illumination (_OutlineLit): Dictates if the outline receives illumination or not
6. 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)
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
8. 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

Demo Scenes

The asset includes 2 Demo scenes (Press the change scene button to go back and forth):

1. Demo: A display of many examples of what this asset can achieve (use Arrow keys or A and D to move left and right).
2. A playable demo with a very simple and quickly put together cave where you control a player that can move around and jump. It has some useful features such as jump buffering, coyote time and multiple jumps (feel free to use the character or code on your projects, it may be useful for fast prototyping). It uses a perspective cam and sprites distributed along the *Z* axis to give some sense of depth:



3. Shadows and billboards display: shows some sprite casting shadows and being billboarded towards the camera.

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, keep in mind that computing light can be taxing for the hardware and that depending on the platform you can run into performance problems. Take a look at the Deferred vs Forward Rendering section in order to choose the rendering path that will give you the better performance results.

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 Scroll View version and the Lighting 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)

Art Credits

Some of the images in the demo scenes weren't created by the creator of this asset.

The hand painted tiling textures were downloaded from the following link and used without any modification (license CC BY 4.0):
https://beefpuppy.itch.io/hptt (username: beefpuppy)

And some characters, the gems and decoration elements were obtained from Kenney Assets (license CC0):
https://www.kenney.nl/assets