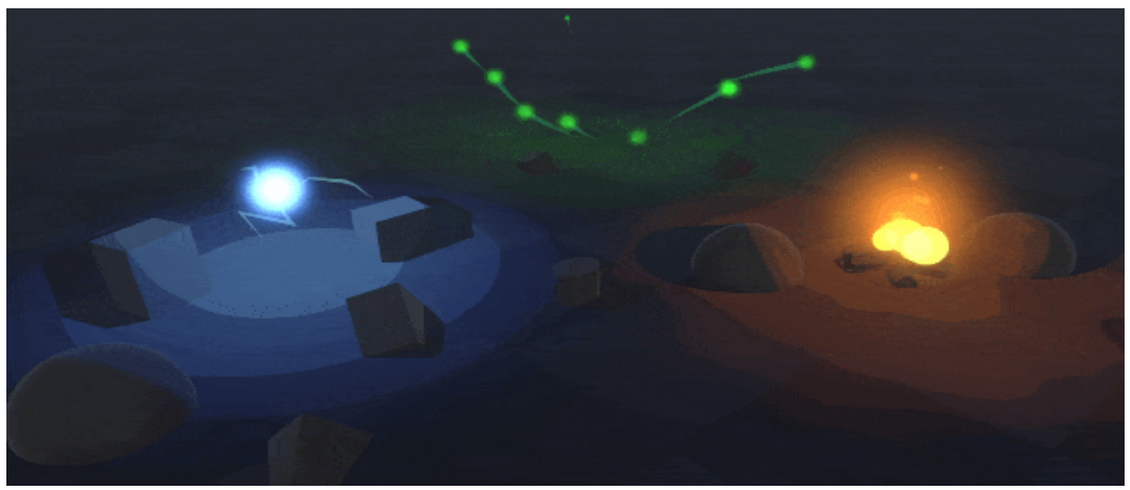
Fake Point Light for Unity

(URP / Built-in)

Welcome and thank you for your interest in this asset!

In this documentation you will find information on how to set up and use the Fake Point Light Shader and avoid possible issues.
Before purchasing, be sure to check the ***Compatibility / Limitations** page to make sure Fake Point Light will work with your project.



❖ If you like the asset, be sure to **Leave me a Quick Review**. It truly helps a lot with visibility!

Quick Start **2**
- Importing The Package 2
- How to use 2
- Activating Depth Normals (Shading) 3

Shader Properties **4**
- Light Settings 4
- Modules 4
- Extra Settings 6
- Particle Mode 6
- URP Accurate Colors: 7
- Day Fading: 7

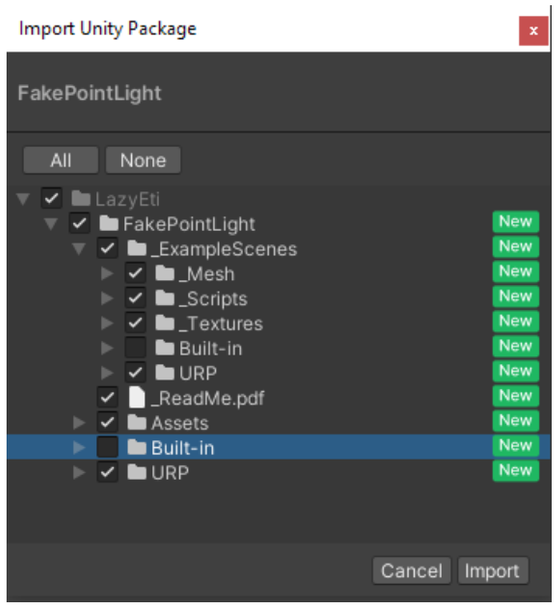
Support **8**
- Compatibility 8
- Known Limitations 8
- Benchmark 9
- Performances Tips 9
- Report an Issue 9

Changelog 10

Quick Start

Importing The Package

When first importing the asset in your project, be sure to select only what is necessary for your Pipeline.
In this example (URP project) the “built-in” folders are deselected to avoid importing incompatible shaders:



The Files that can be discarded are:

- ***Urp*** folders (If using Unity Built-in)
- ***Built-in*** folders (If using Unity URP)

Optional files:

- ***_ExampleScenes***
- ***ReadMe.pdf***

How to use

How does a Fake Point Light work? Well it is called a fake light because it’s actually just a mesh with a special shader applied to it that calculates lighting in screen space.

- Move the GameObject where you need it in your scene.
- Scale it to increase the light radius.
- Modify the material parameters to customize the look.

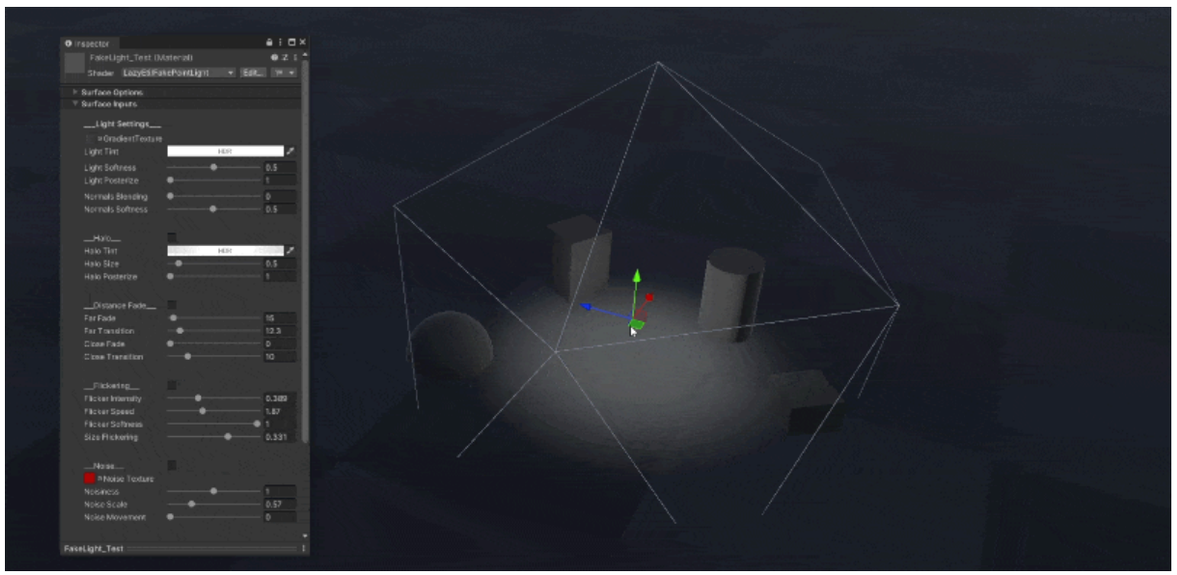
* Activating the **Camera Depth Texture** is Required for the shader to work properly
* A **Directional light** is required in the scene for the shader to work properly

This package contains a basic Prefab ready to be duplicated, instanced or dragged right into your scene. However, here is how you can create a new light from scratch:

1) Create a new Material using “LazyEti/URP/FakePointLight” as its shader.

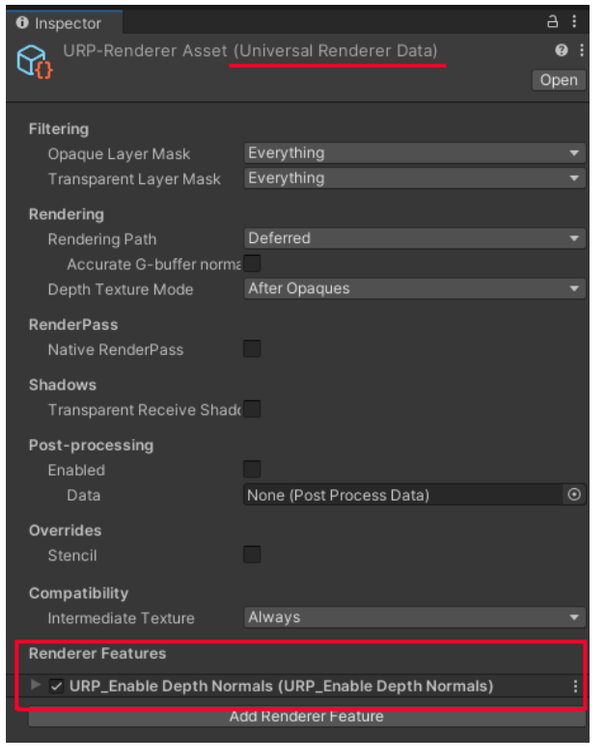
2) In your scene, create a 3D mesh and apply the material to it. (it is recommended to use the low poly icoSphere mesh included in the project files for best performances)

3) Tweak the material parameters and toggle effect modules to customize the look of your Light.



Activating Depth Normals (Shading)

In order for shading to display properly Fake Light requires Unity's "**DepthNormalTexture**".
However, this texture is not rendered by default so here's how you can enable it in your project.

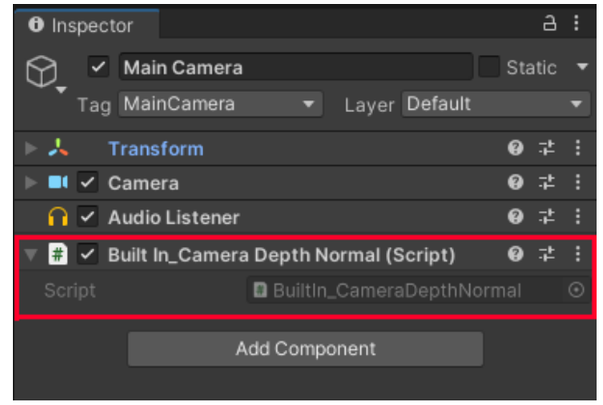


Unity URP:

- Navigate in your project files and select the active **UniversalRendererData** ProjectSettings > Quality > Render Pipeline Asset
  (by default it should be called "URP-HighFidelity-Renderer")

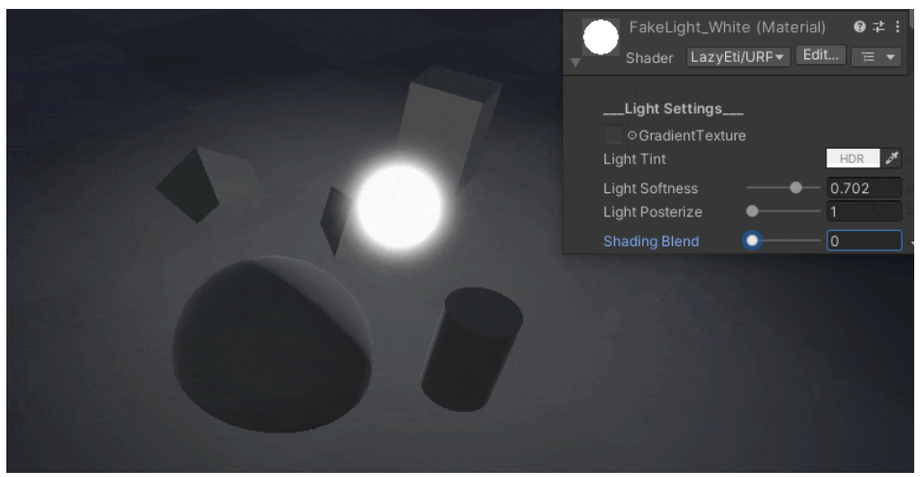
- Press "Add Renderer Feature" and
  select the "**URP_EnableDepthNormals**" feature

* Note that it might not be necessary to do this if you are already using Unity's SSAO Render Feature or another render pass that already renders the DepthNormalTexture.



Unity Built-In:

- Open your scene and select the main camera.
- Add the included "**BuiltIn_CameraDepthNormal**" script to your camera.



Once the "**DepthNormalTexture**" **is enabled**, Fake Point Light shader **"Shading Blend"** parameter will control how much light can bleed through shaded surfaces.

Shader Properties

Each section of the Fake point Light material is a module that can be turned on to add more visual flair to your effect.
caution: each module activated creates more GPU overhead

Light Settings

“**Gradient Texture**”	Specify a color gradient texture to customize the color attenuation of your Light. (This is applied to both the Light and Halo)
“**Light Tint**”	Controls the color, intensity and opacity of the Light.
“**Light Softness**”	Controls the attenuation of the Light.
“**Light Posterize**”	Posterizes the Light for a more stylized look.
“**Shading Blend**”	Controls how much light bleeds through surfaces. 0 = fully shaded, 1= fully lit. (Requires Unity’s “DepthNormalTexture” for shading)
“**Shading Softness**”:	Controls how harsh or soft the shading is.(Requires Unity’s “DepthNormalTexture” for shading)

Additional Modules

Halo:

“**Halo Tint**”	Controls the color, intensity and opacity of the Halo.
“**Halo Size**”	Controls the size of the Halo.
“**Halo Posterize**”	Posterizes the Halo for a more stylized look.
“**Halo Depth Fade**”	The distance at which the Halo opacity blends when intersecting with other objects.

Distance Fade:

”**Far Fade**”	Distance where the light starts fading out.
”**Far Transition**”	Distance over which the Light will fade out.
“**Close Fade**”	Distance where the light starts fading in.
“**Close Transition**”	Distance over which the Light will fade in.

Flickering:

“**Flicker Intensity**”	Controls the intensity of the flickering effect.
“**Flicker Hue**”	The tint toward which the light will shift while flickering.
“**Flicker Speed**”	The speed of the flickering loop.
“**Flicker Softness**”	The abruptness of the flickering transition.
“**Size Flickering**”	The influence of flickering on the size of the light.

Noise:

“**Noise Texture**”	Specify a noise texture to customize the shape of the light.
“**Texture Packing**”	Select which texture channel to sample. (Red, Red*Green, Alpha)
“**Noisiness**”	Controls the noise intensity.
“**Noise Scale**”:	Controls the noise scaling (tilling).
“**Noise Movement**”	Controls the noise movement speed.

Dithering Pattern:

A screen space dithering pattern applied over the light.

“**Dither Intensity**”	Controls dithering intensity.

Specular Highlight:

Renders a specular highlight over lit surfaces.

“**Spec Intensity**”	Controls specular intensity.

Screen Shadow:

Heavy experimental screen space effect. Use at your own risks, Not recommended for low end devices*

“Screen Shadow”	Low = 16 samples, Medium = 32 Samples, High = 64 samples, Insane = 128 Samples.
“Shadow Threshold”	Controls the shadow collision threshold (small values might cause artifacts).

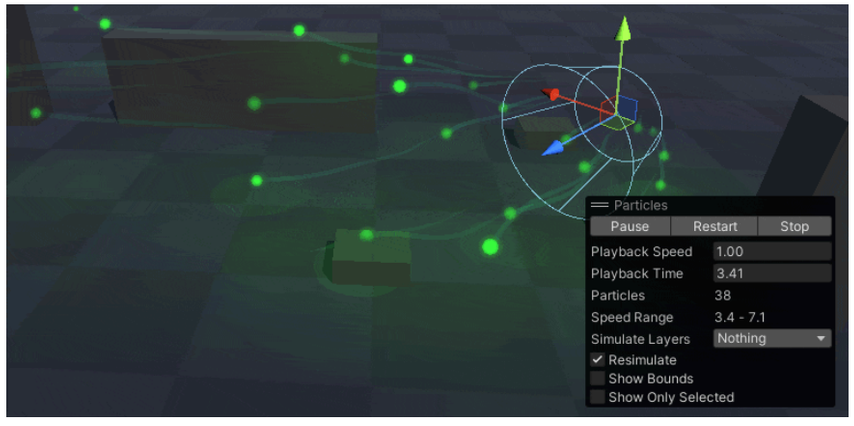
Extra Settings

”**Particle Mode**”	Makes the light compatible with Particle Systems. *Read “**Particle Mode**” for more information.
“**Accurate Colors**”	Blends the light with the opaque texture for a more accurate lighting style instead of just superimposing the light.
“**Day Fading**”	Automatically fades out the material opacity depending on the sun's (Directional Light) direction. Fades out when pointing down (Day time)

Particle Mode

The Fake Point Light Shader is totally compatible with particle systems!

If you are creating a new effect from scratch, follow carefully these steps.
They are all required for the fake light particles to work properly:



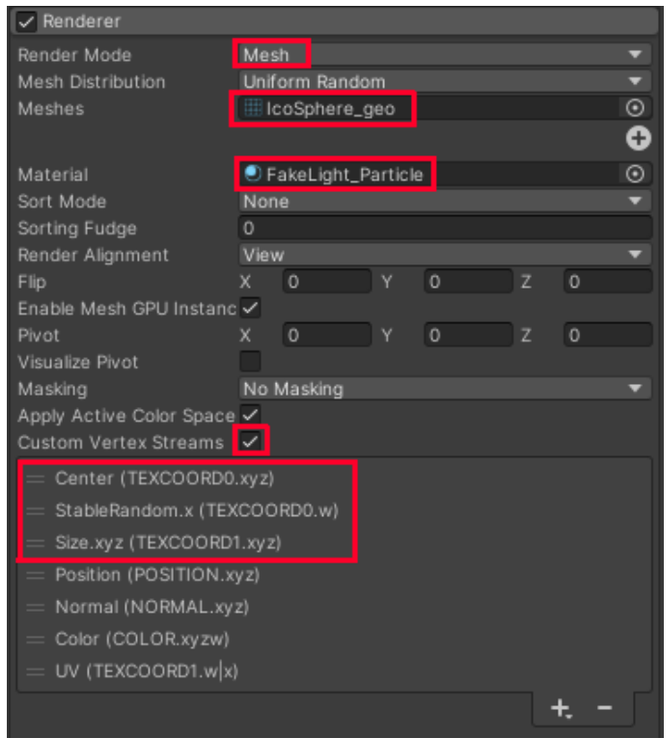
1) Create a new FPL material with the “**Particle Mode**” toggle on.

2) Select your Particle System “Renderer” Module and set the **Render Mode** to **Mesh**.

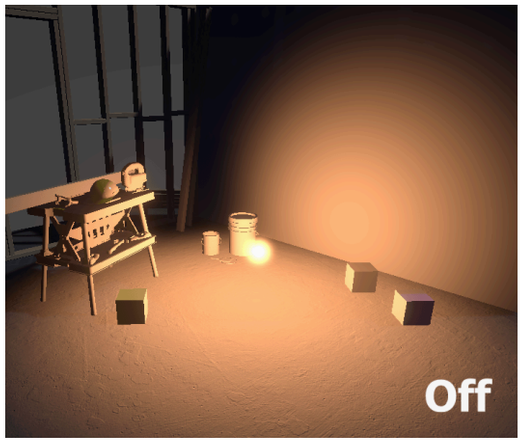
3) Use “**IcoSphere_geo**” as its Mesh and apply the newly created Material

4) Toggle on the “**Custom Vertex Streams**” check box.

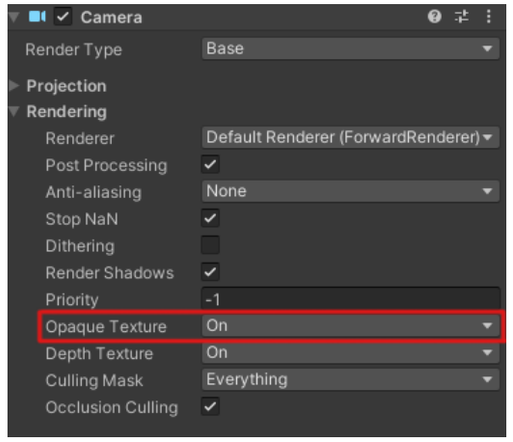
5) Add the following settings at the top of the **Custom Vertex Streams** list in this exact order :
   - **Center** (TexCoord0.xyz)
   - **StableRandom.x** (TexCoord0.w)
   - **Size.xyz** = (TexCoord1.xyz)



URP Accurate Colors:

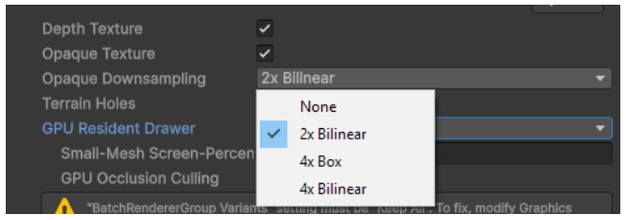


This option can be toggled to achieve a more faithful lighting style.
It blends colors from the background instead of just superimposing the light value on top of surfaces.



In order for “Accurate Colors” to work, Fake Light requires Unity’s “**OpaqueTexture**” to be activated.

The option is found in the settings of your camera
OR
Adjusted in the Universal Render Pipeline Asset.

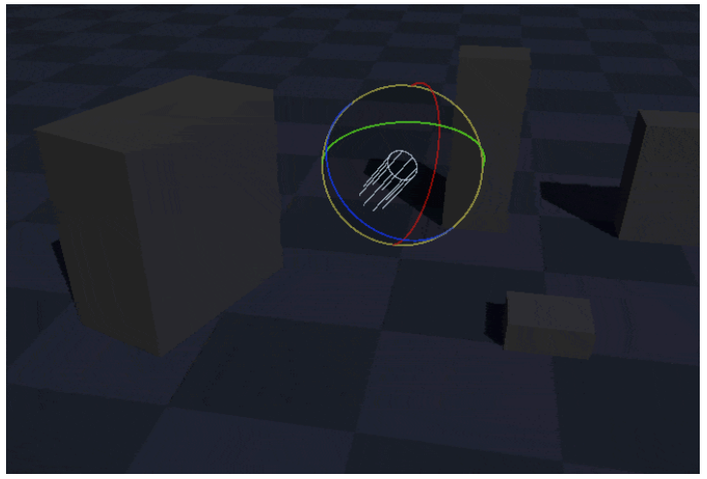


Day Fading:

Day Fading is automatically adjusting the opacity of the light depending on the sun's (Directional Light) direction.

This is useful if your game has a day/night cycle and you want lamp posts to automatically turn on at night and turn off during day time.

The FPL material fades out when the directional light is pointing down (Day time) without having to manually fade it out:



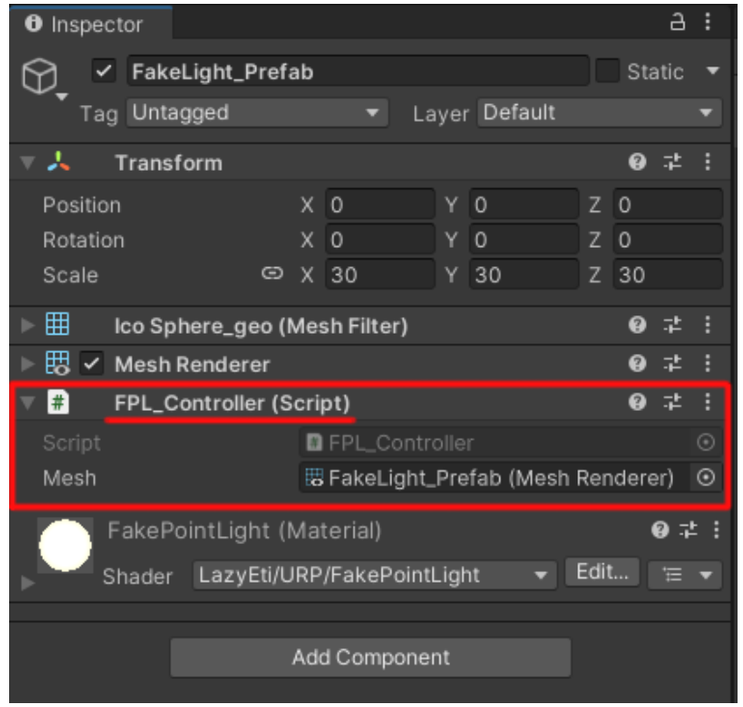
Controlling Lights at Runtime

It is recommended to use the “FPL_Controller” script to easily modify a parameter of the light shader at runtime, Add the script to your light object and call the following methods from any other script to set a parameter:

```
Set a float parameter:
FPL_Controller.SetProperty (FPL_Properties lightProperty, float value)

Set a color parameter:
FPL_Controller.SetProperty (FPL_Properties lightProperty, Color value)
```

In the following example, pressing spacebar will apply a random color to the serialized FPL_Controller:

```
[SerializeField] FPL_Controller fplController;

void Update()
  {
      if(Input.GetKeyDown(KeyCode.Space))
      {
          Color randomColor = Random.ColorHSV () * Random.Range(1,5);

          fplController.SetProperty (FPL_Properties._LightTint, randomColor);
      }
  }
```



FPL_Controller works by overriding the Fake Point Light material parameters through MaterialPropertyBlocks.

This method allows setting unique values per light object without breaking dynamic batching.

For best performance, it’s recommended to avoid setting properties every frame.

Support

Known Limitations

- **HDRP Pipeline is NOT supported.**

- **2D projects are NOT supported:**
  - (Elements such as "Sprite Renderers" and "Renderer 2D Data" are not compatible since they don't write to the depth buffer.)

- **Semi-transparent materials must have a render Queue of <2500 to be lit by the fake light shader:**
  - (This can cause some rendering issues for example if you don’t want your transparent material to produce ssao)

- **Lights CANNOT be set to “Static”**
  - Using the “Static” toggle on a light gameobject will break the effect. If you need to batch lights, turn on the “GPU Instancing” toggle on your light material instead.

- **Shadows are not accurate**
  - Fake point light is a cheap and stylized technique, not a perfect lighting solution. In some cases, the limited screen space information can cause light to bleed through walls. If you need very accurate and realistic shadows, it's better to go with a traditional point light.

Compatibility

Should be compatible with Unity 2019 - 2023 but hasn’t been tested on every version.

Version #	URP	Built-In	HDRP		VR	WebGL
Unity 6	✅ Supported	✅ Supported	🚫 Not supported		❓ Not tested	❓ Not tested
Unity 2023.2	✅ Supported	✅ Supported	🚫 Not supported		✅ Supported	❓ Not tested
Unity 2022.3	✅ Supported	✅ Supported	🚫 Not supported		✅ Supported	⚠️ Supported*
Unity 2021.3	✅ Supported	✅ Supported	🚫 Not supported		✅ Supported	⚠️ Supported*
Unity 2020.3	⚠️Supported*	✅ Supported	🚫 Not supported		⚠️Supported*	⚠️ Supported*
Unity 2019.3	❓ Not tested	❓ Not tested	🚫 Not supported		❓ Not tested	❓ Not tested

* **2020.3 - URP**: shading is not supported

* **2020.3 - VR**: Multi-pass rendering not supported in URP

* **WebGL:** is compatible as long as you manually open the .shader file and change: ***#pragma target 4.5*** to ***#pragma target 3.5***

Benchmark

(A detailed benchmark is coming soon)

Here are my informal thoughts about it until I finish creating a real benchmark:

All calculations are running on the GPU so it's not super recommended for mobile devices but it's fine if using only a few lights / not using shading.

On a Nintendo Switch, a couple lights on screen have no noticeable performance impact.
However performances will start tanking only when a large amount of lights is displayed on screen.

For example on my mid-high end GPU (GTX 1080) there is zero impact on performance until reaching around 600 lights on screen at which point the GPU starts to bottleneck.
After which the profiler starts to show increasingly bigger lag spikes as you add more lights (~8+ms when there are over 1000 lights on screen).

Performances Tips

- Turn off modules you don’t need. Each module activated on the light material will increase its GPU Overhead.

- Experimental Screen Shadow setting is by far the heaviest feature. For best performance, it is recommended to keep it off.

- A large quantity of lights can cause lag spikes (Semaphore.WaitForSignal()) depending on the target GPU. Try keeping the number of lights visible on screen under 400.

- On low end devices (such as phones), it is usually not recommended to use Unity’s Depth Normals. This means that for best performances, shading should be turned off all together (“**Shading Blend**” = 1).

Report an Issue

If you have any issues or additional questions, you can contact me at: **support@lazyeti(dot)xyz**

Please make sure to include in your message:

1. **Unity version & Pipeline**
2. **Fake Point Light version (can be seen in package manager)**
3. **Description of the issue (ideally with images if applicable)**
4. **Editor or Build?**

Changelog

Version	Bug Fixes	New features
1.0.1	❖ Fixed warning when Amplify is not installed ❖ Fixed BIRP not displaying lights if shadow rendering is disabled in settings ❖ Fixed URP <2021: Error: Couldn't include DOTS.HLSL ❖ Fixed URP <2021: URP_EnableDepthNormals ERRORS ❖ Fixed Build Error: Light instancer.cs -> Clear Grids() ❖ Fixed Particles Materials Unity <2021	★ Added "SmoothEdge" Parameter to spotlight shader
1.0.3	❖ Fixed VR compatibility shader Error ❖ Fixed WebGL compatibility shader Error	★ Added more spotlight 3D meshes
1.0.4	❖ Fixed Orthographic camera issues	
1.0.5	❖ Fixed BIRP VR Single pass SAMPLE_TEXTURE2D_ARRAY error ❖ Fixed AR Passthrough rendering compatibility on Meta Quest ❖ Changing between isometric and perspective should preserve halo size	★ Urp: "Accurate Colors" Setting ★ "Halo Depth Fade" control
1.0.6	❖ Fixed "EnableDepthNormal" Pass warning message on Unity 6 ❖ Fixed Dots Instancing Error messages on Unity 6	
1.1.0	❖ Improved noise projection on rounded surfaces ❖ Fixed soft shading issues ❖ Optimized shader and removed unnecessary instructions	★ New custom material editor ★ Added Blend modes ★ Added Extra setting: Specular Highlight ★ Added Extra setting: Dithering Pattern ★ Added Flicker Tint parameter ★ Improved screen space shadows look and reliability
1.1.1	❖ Fixed "CustomMaterialEditor.cs" errors when building	
1.1.2	❖ Fixed warning "_SrcBlend already has a property drawer" ❖ Fixed Error "Material Header Scope List" is not found (BRP 2022.3.6f1)	
1.1.3	❖ Fixed namespace "RenderGraphModule" (Unity 6 URP)	
1.1.4	❖ Fixed Shader Error "'Inverse4x4': no matching 1 parameter function" caused when BatchRendererGroup Variants set to Keep All (Unity 6 URP) ❖ Fixed Isometric perspective not working when recompiling shader with latest Amplify Editor	
1.1.5	❖ Cleaned up example scripts	★ Added FPL_Controller.cs to easily access and change material parameters through code.