Lumi

Documentation (v1.1.2)

thecarow.com

Overview

Lumi is a system for sampling light at a given location.

Lumi primarily consists of 2 components:

- *LightDetector* - performs all visibility calculations.
- *LumiLight* - component that attaches to Unity *Lights*.

Compatibility

Supported render pipelines:

- Built-in Render Pipeline (BIRP)
- Universal Render Pipeline (URP)
- High Definition Render Pipeline (HDRP)

Supported realtime lights:

- Directional
- Point
- Spot

Supported baked lighting sampling:

- Light Probes
- Lightmaps

Getting Started

1. Create a *GameObject* and add a *LightDetector* component to it.
2. Create a light (spot, directional, or point) and add a *LumiLight* component to it.
3. Create a *GameObject* that will act as a light sample point.
4. Assign the *LumiLight* and sample point *Transform* components to the corresponding **Lights** and **Sample Points** fields in the *LightDetector's* inspector.
5. Go into Play Mode.
6. Observe the Output and Output Color shown on the *LightDetector* component when the *SamePoint* is within the lights area of influence.

Sample scene

There are sample packages for the built-in render pipeline (BIRP), Universal Render Pipeline (URP), and High Definition Render Pipeline (HDRP). The packages can be found under **Lumi/Samples**. Simply import the package appropriate for your render pipeline and open the **Playground** scene.

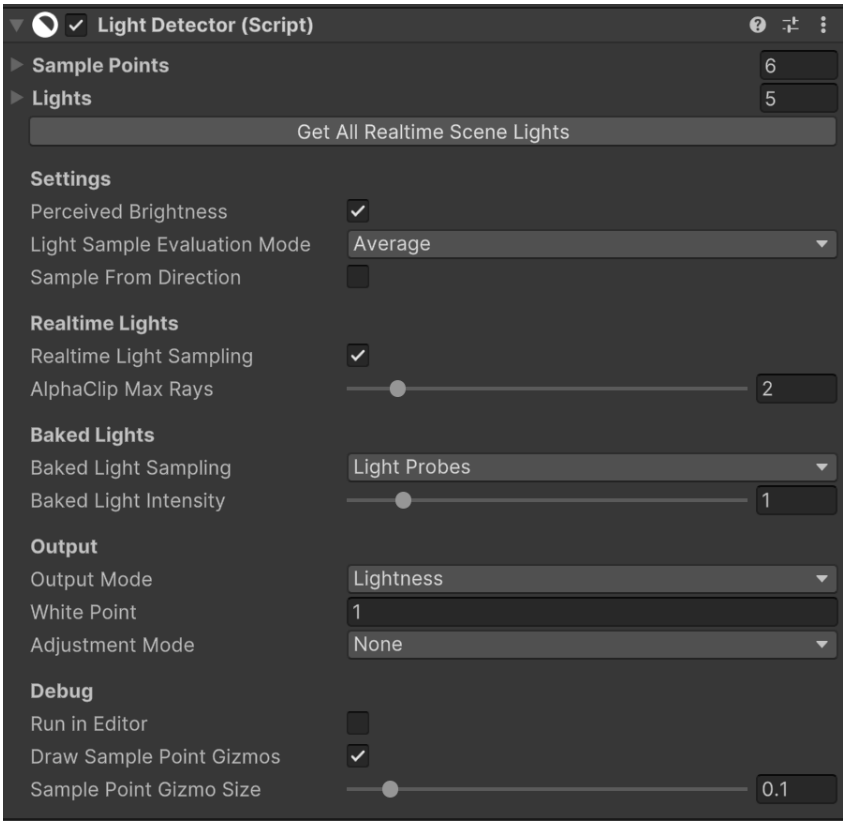
Components

LightDetector

LightDetector is a component that performs all visibility calculations of the specified *LumiLights* and sample points.

The end result of all sampling is accessible as Output and Output Color.

More than one *LightDetector* can run in a scene.



LightDetector inspector.

Sample Points

A list of all the sample point *Transforms* that the *LightDetector* will use as positions to sample.

Using more sample points increases performance cost. Optimize your setup by using only as many as your scene requires.

Light

A list of all the LumiLights that the *LightDetector* will use to sample the sample points.

The more *LumiLights* are used, the larger the performance impact. As such, use as few *LumiLights* as necessary.

It is possible to add/remove lights at runtime. This can be used to minimize performance impact, as well as support continuously loaded content. As an example, in a stealth game scene with many lights, if only a few lights are expected to affect the player character at a given time, *LumiLights* can be added/removed as the user moves through the scene.

Perceived Brightness

If enabled, adjusts the Output in accordance with human perception.

When a light source is sampled, the luminance is calculated using the following formula:

lum = 0.2126 * r + 0.7152f * g + 0.0722 * b

If disabled, the luminance is calculated as

lum = (r + g + b) / 3

Example: with perceived brightness enabled, a green light (RGB 0,1,0) will return a higher luminance value than a blue light (RGB 0,0,1) under the same conditions (intensity, position, etc.)

Light Sample Evaluation Mode

Specifies how the user-specified sample point results are aggregated together:
Average - The luminance/lightness of each sample point is added together and divided by the number of sample points, providing an averaged Output.

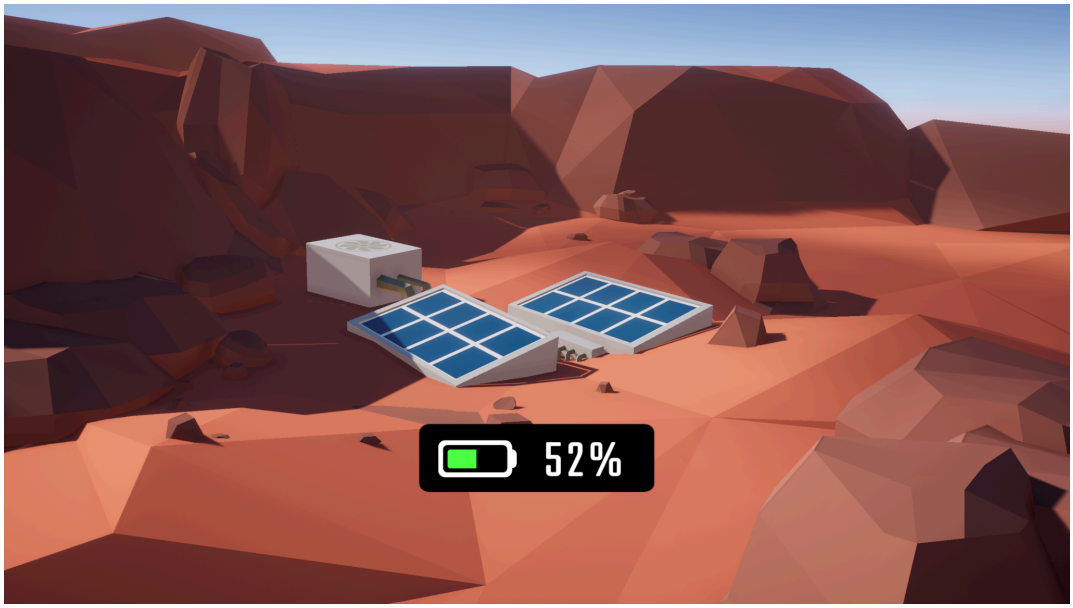
Max - Returns the highest luminance/lightness out of all the sample points.

Sample From Direction

Specifies if incoming light should be sampled from the forward direction of the sample point.

As an example, this can be useful when simulating the light coming from a *Directional Light* into solar panels. Alternatively, it can be useful when the visibility of a character is important from a specific angle, such as the viewing angle of an NPC.

Note: Light Probe sampling when **Sample From Direction** is enabled can only return a maximum light intensity of 1.



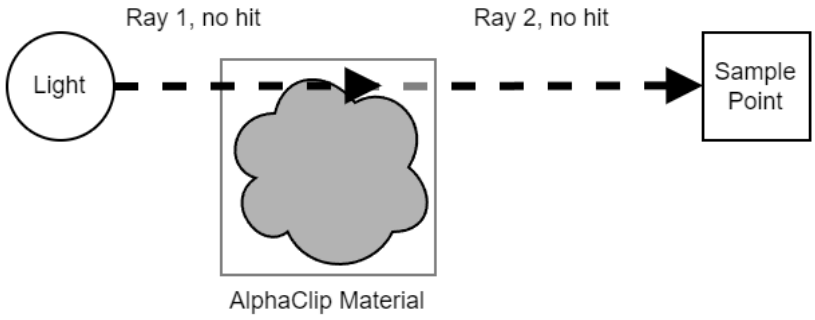
Example use of **Sample From Direction** - solar panels report less incoming energy as the angle between the sun's *DirectionalLight* and the solar panel increases.

Realtime Light Sampling

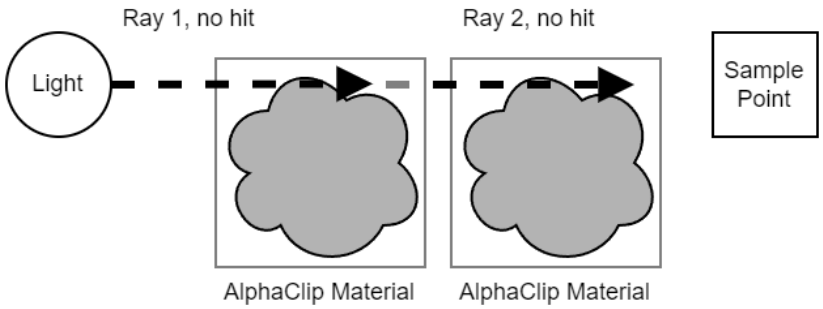
Specifies if the specified realtime lights will be sampled.

Transparency Max Rays

When a *LumiLight* has its setting **Transparency Support** enabled, this setting specifies the maximum number of rays that will be used to check for occlusion.



Scenario: **Transparency Max Rays** set to 2 and neither ray hits an opaque texel - sample point is illuminated.



Scenario: **Transparency Max Rays** set to 2, but 3 are required to reach the sample point - sample point will erroneously be reported as occluded.

Note: the higher number of rays used, the higher impact on performance.

Baked Light Sampling

Specifies how baked light will be sampled:

None - baked light will not be sampled.

Light Probes - Will sample nearby light probes for baked light information. Ensure that light probes are present and baked. **NOTE**: Adaptive Prove Volumes (APV) are **NOT** supported.

Lightmap - Sample point will cast a ray in its forward direction and sample the hit lightmap. If no lightmap is hit, Lightmap Fallback Color will be returned. The following requirements and limitations apply:

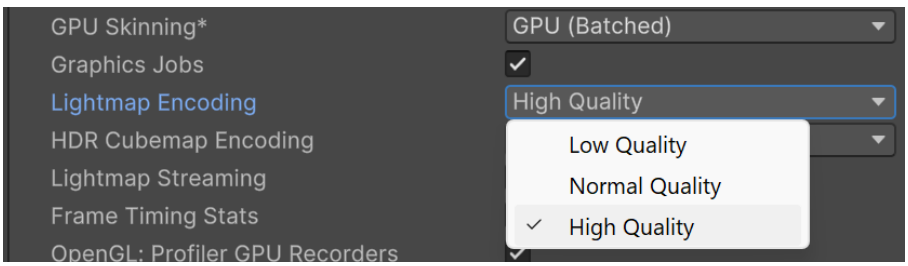
- The lightmap texture must have the **Read/Write** option enabled on the textures import settings.
- Lightmap Encoding Quality should be configured.
- Double-check the Lightmap Ray Distance.

Baked Light Intensity

Multiplier applied to sampled baked light. Useful for fine tuning Output results.

Lightmap Encoding Quality

Specifies what calculation is used to sample lightmaps. This MUST match **Project Settings > Player > Other Settings > Lightmap Encoding** if lightmap sampling is used in order to ensure accurate lightmap sampling results.



Lightmap Encoding setting.

See Lightmap data format Unity documentation for more information on lightmap encoding.

Lightmap Ray Mask

Determines which layers lightmap sampling rays interact with.

Lightmap Ray Distance

Specifies the length of the lightmap sampling ray.

Lightmap Fallback Color

When Baked Light Sampling is set to **Lightmap**, this setting specifies what color is used if a lightmap raycast fails to hit and sample a target.

Output Mode

Specifies what calculations are applied to Output.

Luminance - linear measurement of light. Use this for advanced visibility calculations. Values can exceed 1, unless Normalize Output is used.

Lightness - linear prediction of the human perception of luminance. Normalized 0-1. Recommended starting point.

White Point

Used when Output Mode is set to **Lightness**. Reference point for maximum luminance in a given viewing environment or color space. Configuring the white point is necessary for accurately calculating perceived lightness because it establishes the upper limit of luminance in a scene or display.

Example setup to establish a white point:

1. Set up a light in the scene that will represent the highest light intensity that is expected to illuminate a character.
2. Position a *SamplePoint* right next to the light source and measure its luminance level.
3. Set White Point to the sampled luminance level.

As mentioned in the Limitations section, Lumi does not account for post processing. As such, effects such as tonemapping and exposure can influence a user's perception while not affecting lightness readings. Further fine tuning of the white point and Output might be necessary. Adjustment Mode is one tool available.

Normalize Output

Used when Output Mode is set to **Luminance**. If enabled, each sample point's luminance/lightness is remapped using Remap From Range to a normalized 0-1 range. For advanced uses.

Remap From Range

Used when Output Mode is set to **Luminance**. Specifies the range of luminance values to remap to a normalized 0-1 range when Normalize Output is enabled. For advanced uses.

Adjustment Mode

This setting is used to apply additional fine tuning adjustments to sampled light.

None - Sampled light is not adjusted.

Power - Sampled light is raised to the user specified power.

Curve - Enables the user to specify a remap curve.

Adjustment Mode is useful when a sample point moving from various shades of light is not perceived as correct by the user.

Output

The luminance or lightness calculated by a *LightDetector*. Can be accessed via code using *LightDetector.Output*. Visible in editor if *LightDetector* is set to run in Editor or if Editor is in Play Mode. This is the value that can be used to determine how visible sample points are.

Output Color

The color calculated by a *LightDetector*. Can be accessed via code using *LightDetector.OutputColor*. Visible in editor if *LightDetector* is set to run in Editor or if Editor is in Play Mode.

Run in Editor

Specifies if the *LightDetector* should calculate and output Output and Output Color while in Editor mode. Useful for quickly prototyping and iterating.

Draw Sample Point Gizmos

Specifies if green gizmo spheres should be drawn at the locations of sample points.

Sample Point Gizmo Size

Specifies the size of sample point gizmos.

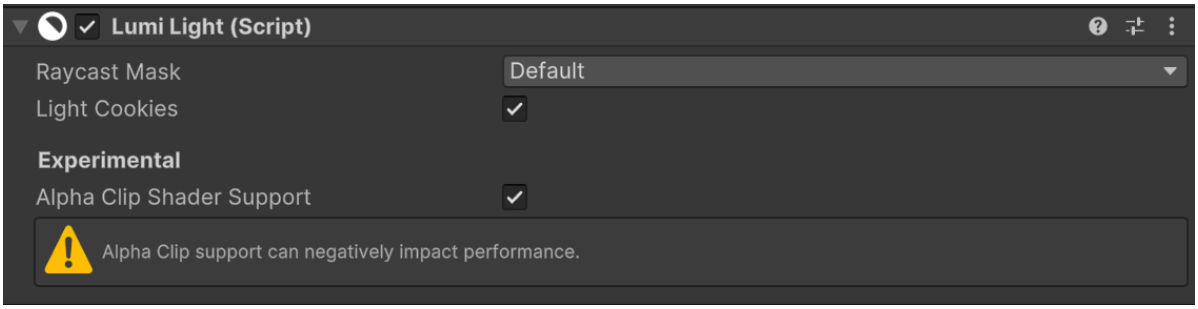
LumiLight

LumiLight is a component that attaches to Unity *Lights.*

Physics raycasts are used to evaluate light at each sample point. Ensure shadow-casting objects have accurate colliders for reliable results.

A disabled *LumiLight* component is ignored by the *LightDetector*.

A *LumiLight* component will continue to be active even if its associated *Light* is disabled. This is useful if an artificial luminance increase is needed without an accompanying visual element.



LumiLight inspector.

Active If Light Is Disabled

If enabled, the *LumiLight* will be used in light calculations even if the associated *Light* component is disabled.

Useful to artificially boost *SamplePoint* light values, or when used with packages such as FakeLight.

Shadow Occlusion

Checks if a sample point is occluded (in shadow) using raycasts. Default: **On If Light Renders Shadows**.

On If Light Renders Shadows - Only checks for occlusion if Unity *Light* setting **Shadow Type** is set to **Hard Shadows** or **Soft Shadows**.

Forced On - Always performs occlusion raycasts.

Forced Off - Never performs occlusion raycasts.

Raycast Mask

The Raycast Mask is used to determine which layers the raycast can interact with for a given light.

This is useful if the scene has objects with colliders that should not obstruct the light sampling. As an example, if a sample point is inside a player character, the character's colliders should not be in the **Raycast Mask**. This can lead to the character mesh occluding its own sample point.

As raycasting can negatively impact performance, ensure that only necessary layers are specified.

Light Cookies

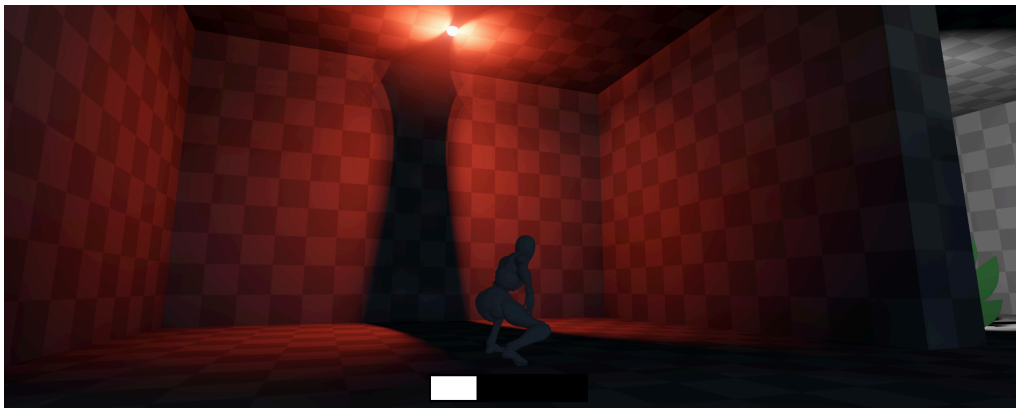
Specifies if light cookie support is enabled for a given realtime light.

When the feature is enabled and a light cookie is provided to a *Light*, the light cookie is used when calculating Output.

The following requirements apply to light cookie textures:

- Texture must have the **Read/Write** option enabled on the textures import settings.
- Crunch compression must not be used on the texture.

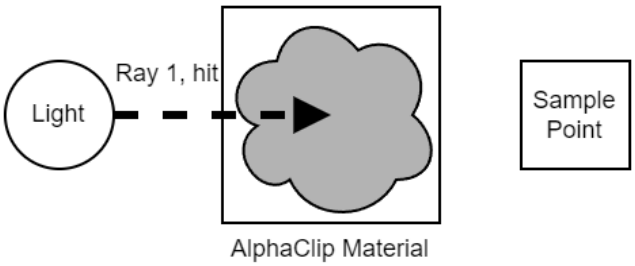
As light cookie sampling can impact performance, ensure only lights that must have this feature have it enabled.



Example of **Light Cookies** - the character is in a low visibility area as the alarm light's cookie is sampled.

Transparency Support

Specifies if light sample rays coming from a given light check intersecting geometry for AlphaClip materials.



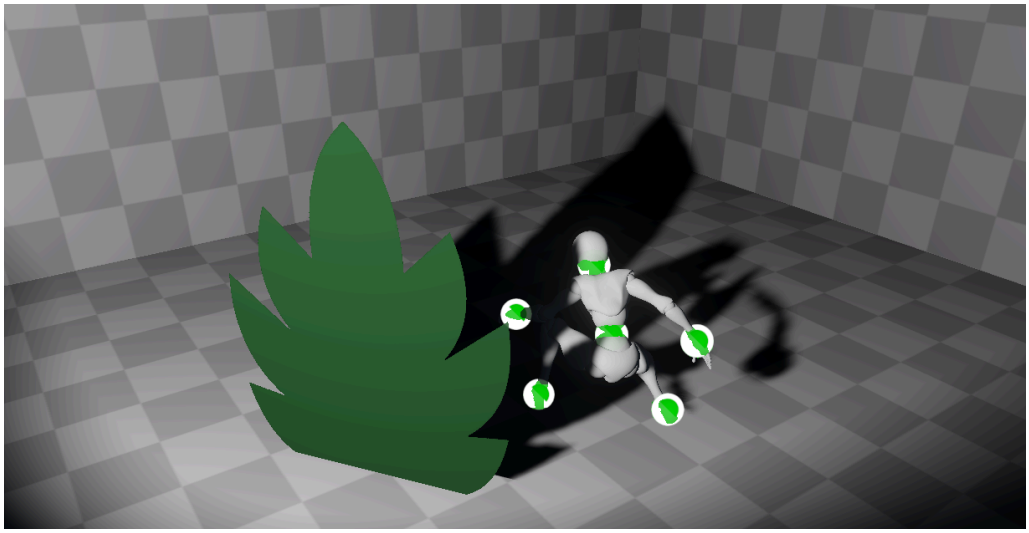
Scenario: A light ray hits an object with an AlphaClip material. The hit texel of the texture is opaque. Sample point is occluded.

The following requirements and limitations apply:

- The intersected material shader must have the following configuration (*Standard*, *Universal Render Pipeline/Lit*, and *HDRP/Lit shaders* are supported):
    - **_MainTex** property must be present specifying the AlphaClip texture.
    - For Built-in, **_Mode** set to 1 and **_Cutoff** optionally specifying AlphaClip threshold.
    - For URP, **_AlphaClip** set to 1 and **_Cutoff** optionally specifying AlphaClip threshold.
    - For HDRP, **_AlphaCutoffEnable** set to 1 and **_AlphaCutoff** optionally specifying AlphaClip threshold.

- Only 1 material per mesh.
- The texture must have the **Read/Write** option enabled on the textures import settings.
- Back faces rendered by the shader will not be sampled correctly. Back faces should be authored as dedicated geometry.
- Collider must be a convex mesh collider.

As AlphaClip texture sampling can negatively impact performance, ensure this feature is enabled only when necessary. Additionally, ensure that Transparency Max Rays property is set to the lowest acceptable number.



Example use of **Transparency Support** - character's sample points are not occluded by transparent texels.

Transparent Shadows (BIRP only)

Transparent shadows are supported in BIRP. The Standard shader must have **Rendering Mode** set to **Transparent**.

For custom shaders, the following requirements have to be met:

- **_Mode** float must be present and set to **3**.
- **_MainTex** should specify the transparent texture.

Light Sampling via API

Once a *LightDetector* is set up in a scene, the following public methods can be used to do manual luminance sampling:

- *SampleDirectionalLight*
- *SamplePointLight*
- *SampleSpotLight*
- *SampleLightProbes*
- *SampleLightmap*

- *SampleLightmapDirect*

Note that using the previously mentioned methods directly will not subject the output to Normalize Output and associated settings.

Performance

Lumi performs most of its work on the CPU. Profiling tools, such as the Unity Profiler can be used on development builds to assess performance.

As a general rule of thumb, use as few *LumiLights* and sample points as necessary. Ensure only necessary features are enabled.

If more control over the *LightDetector* update frequency is needed, it is possible to disable the *LightDetector* component and to invoke *PerformSampling* method as necessary.

Features such as lightmap sampling, AlphaClip support, and light cookie support require Read/Write enabled textures. Note that enabling the **Read/Write** flag on a texture effectively doubles its memory footprint as an additional copy is loaded into CPU-addressable memory.

Limitations

Lumi samples luminance/lightness at given sample point locations, using compatible realtime and baked light sources. It does not take into consideration screen space effects (post processing, SSAO, GI, etc.), materials, VFX, shaders, and so on. As such, fine tuning of Output (within or outside of *LightDetector*) might be needed to match developer/user expectations.

Example: Objects with specular reflections can appear clearly visible to the user, but the *LightDetector* will have no way of detecting this.

Example of specular reflections making a character more visible in the dark, creating a mismatch between user perception and reported visibility.

Lumi was designed with linear color space in mind. For gamma space, using luminance instead of lightness can in some cases be more accurate.