Analytical Volumetric Lighting v1.2

Thank you for purchasing Analytical Volumetric Lighting tool! This documentation should help you better understand all the features available, how they can be used in practice, as well as implementation and architecture ideas applied developing AVL 1.0.

- **Unity Asset Store Page**
- **GitHub Bug Tracker & Roadmap & Demo**

👋 Introduction

Analytical Volumetric Lighting is a post-processing render feature intended to replicate the behavior of the volumetric lighting focusing on **high performance** and **high real-time light count**. This is achieved thanks to the analytical approach which makes per-light calculations constant-time. Instead of estimating air light integrals with raymarching, they are calculated precisely.

Compatibility

Due to the specific shader features used by internal AVL shaders, some of the platforms might be incompatible.

✅ Supported Unity Versions

- **2021.3.x**
- **2022**
- **2023**
- **Unity 6**

✅ Supported Rendering Paths

- **Forward**
- **Deferred**
- **Forward+**

✅ Supported Platforms

These are the platforms that have been tested through running all the example scenes:

- **Windows DX11 (Nvidia / AMD / Intel)**
- **Windows Vulkan**
- **Windows OpenGLCore**
- **Windows OpenGLES 3.2***
- **Android Vulkan**
- **Metal (MacOS)**

* - **Depth Culling** is not supported.

❎ Unsupported Platforms

These are the platforms that have **not** been tested:

- **OpenGLES 3.1**
- **Other Mobile Platforms**

These are the platforms that have been tested and are **not** supported:

- **OpenGLES 3.0 or lower**
- **Any VR Platform**

Limitations

When compared to popular raymarching solutions, analytical approach has its own limitations:

⚠ Shadow Mapping

Lights cannot cast any shadows of any sort as this is not possible due to the integration becoming too complex.

⚠ Noise Textures

Lights cannot have noise textures for the same reason - this is not possible due to the integration becoming too complex.

⚠ Light Blending

AVL uses additive light blending. This is not the most accurate way to account for the volumetric lighting but is quite simple and can be handled using the analytical approach.

⚠ Transparency

AVL, as for the moment, does not support transparent objects. They will be rendered on-top.

⚙ First Setup

AVL first setup is quite simple:

Prerequisites

- Check if your graphics API and platform is supported. Refer to **Compatibility** section if needed.
- Check if **Depth Texture** is enabled in the URP asset.
- Check if **Post-processing** is enabled in the URP renderer asset.

Render Feature

- Add **AVL Render Feature** in the URP renderer asset.
- **⚠ Set up Clustering, Rendering, and Debug shaders.** These shaders can be found in “AkiDevCat/AnalyticalVolumetricLighting/Shaders/Passes”.
- Set up as needed. Refer to **AVL Render Feature** section.

Post-Process Volume

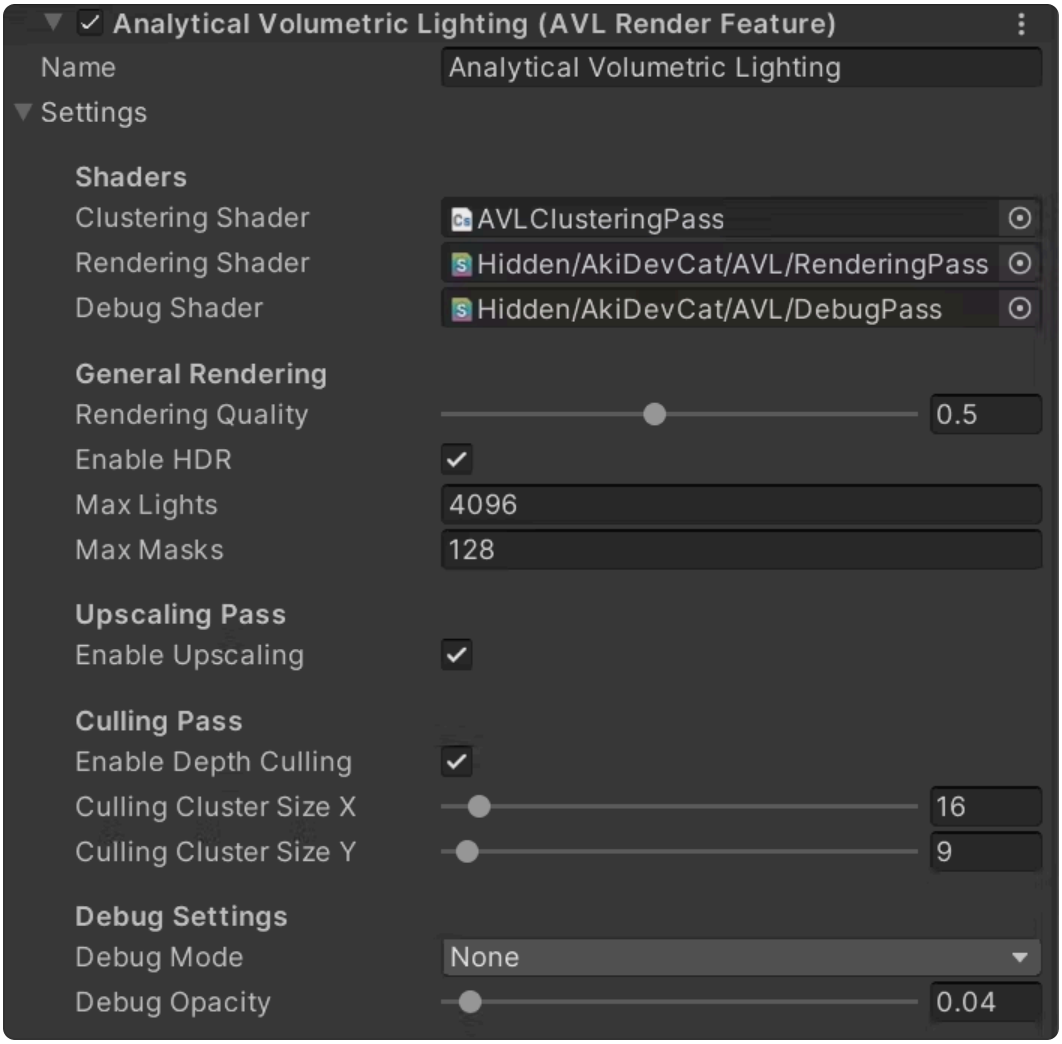
- Add **AkiDevCat/Analytical Volumetric Lighting** override.
- Set up as needed. Refer to **AVL Volume Component** section.
- Make sure your camera has **Post Processing** enabled and has your renderer selected.

📃 Documentation

This section describes all of the AVL systems from the user experience standpoint.

AVL Render Feature

AVL Render Feature is required for AVL to take effect in the game and scene view. It should be added as a separate render feature for each of your URP renderers:



Below you can find description for each of the settings parameter:

Rendering Quality

Controls resolution scaling which will be applied during AVL rendering pass. For example, if the screen resolution is **1920x1080 pixels** and the rendering quality is **half**, then the AVL fog texture resolution will be **960x540 pixels**. Recommended to be kept at **half** on high-end / mid-end devices with **quarter** or **eighth** on low-end devices.

Half Rendering Quality **Quarter Rendering Quality**

Enable HDR

Whether or not to use HDR-capable texture format for AVL textures.

HDR Enabled

HDR Disabled

Max Lights

Max light count that can surround camera (not CPU-culled) at the same time. Higher values might increase **VRAM usage** as well as **GPU time**.

⚠ AVL also has `MAX_LIGHT_PER_CLUSTER` constant defined in `Scripts/AVLConstants.cs` and `Resources/Shaders/Includes/AVLStructs.hlsl`. If needed, these constants can be modified. **Be sure to modify the constant in both locations**.

Max Masks

Max mask count that can surround camera (not CPU-culled) at the same time. Higher values might increase **VRAM usage** as well as **GPU time**.

⚠ AVL also has `MAX_MASKS_PER_CLUSTER` constant defined in `Scripts/AVLConstants.cs` and `Resources/Shaders/Includes/AVLStructs.hlsl`. If needed, these constants can be modified. **Be sure to modify the constant in both locations**.

Enable Upscaling

Enables AVL fog texture upscaling from its native resolution to the game rendered screen resolution.

Upscaling Enabled

Upscaling Disabled

Enable Depth Culling

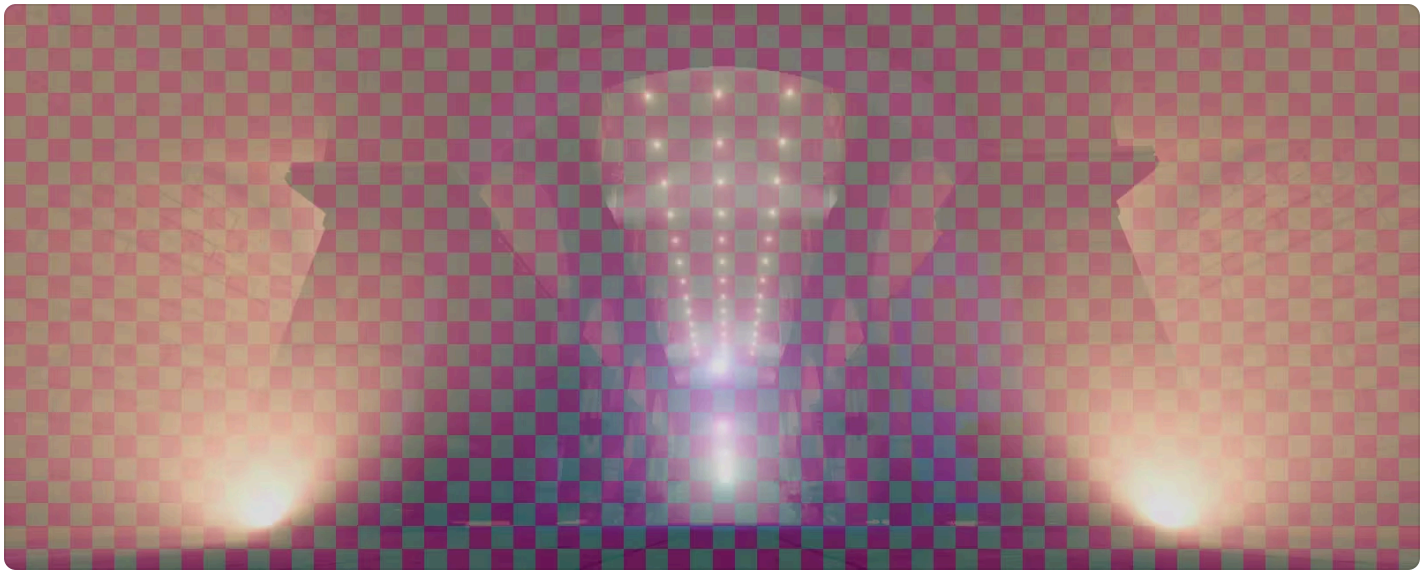
Enables depth culling algorithm for the clustered rendering. Allows lights to be culled by camera depth (behind walls) by trading some constant performance. See **Clustering Pass** for more information.

Depth Culling Enabled (Light Count Overlay)

Depth Culling Disabled (Light Count Overlay)

Culling Cluster Size X / Y

Light cluster count per X / Y screen axis (horizontal / vertical). High values might increase **VRAM usage**. Low values might increase **GPU time**. See **Clustering Pass** for more information.



64x25 Light Clusters

Export Volumetric Light Texture

Exports the volumetric light texture if needed to be used somewhere else. You can get this texture by acquiring **AVLExternalData** in **Unity 6** the following way:

```
public override void RecordRenderGraph(RenderGraph renderGraph, ContextContainer frameData)
{

var externalData = frameData.GetOrCreate<AVLExternalData>();

}
```

Do not forget to mark the texture as read-only:

```
builder.UseTexture(externalData.VolumetricLightTexture, AccessFlags.Read);
```

Light Model

Changes the way volumetric light texture is applied to the screen color texture:

- **Additive** - The volumetric light texture will be applied additively (COLOR + V_LIGHT) - This is the standard option
- **Density Over Luminance** - An additional density value is calculated that is used to apply the volumetric light texture using Beer-Lambert law. The density value is approximated using the luminance function (lerp(COLOR, V_LIGHT, 1.0 - 1.0 / exp(luminance(V_LIGHT.rgb))))
- **Density Over Buffer** - An additional density value is calculated that is used to apply the volumetric light texture using Beer-Lambert law. The density value is taken from the additional V_LIGHT buffer channel. This requires a higher VRAM usage. (lerp(COLOR, V_LIGHT, 1.0 - 1.0 / exp(V_LIGHT.a)))

Debug Mode

Enables debug overlay in scene and game views.

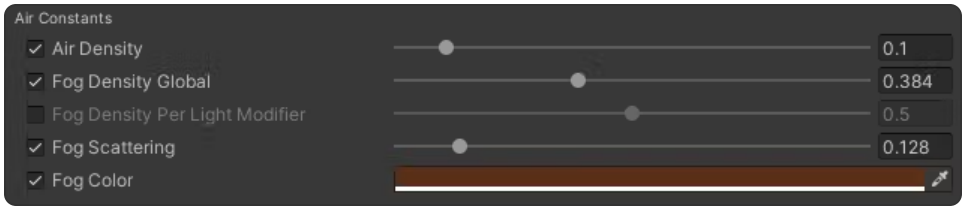
- **None** - Overlay is not rendered.
- **Light Clusters** - Checkerboard pattern visualizing light clusters.
- **Light Overdraw** - Overdraw of lights per light cluster.
- **Light Count** - Light count per light cluster.
- **Volumetric Light** - AVL Fog texture.

Debug Opacity

Debug overlay opacity.

AVL Volume Component

In order for AVL to be visible, it is also required to override the post processing volume settings using **AVL Volume Component**.



Below you can find description for each of the volume settings parameter.

Air Density

Basically, represents a light intensity (brightness) multiplier for each of the light. Low values like **0.02** or **0.03** are usually sufficient.

Air Density 0.01

Air Density 0.03

Fog Density Global

Defines global fog density. AVL global fog is an alternative to the built-in fog feature.

⚠ As for the moment, AVL global fog does not support transparent objects and will be rendered before transparent objects.

Fog Density 0.0

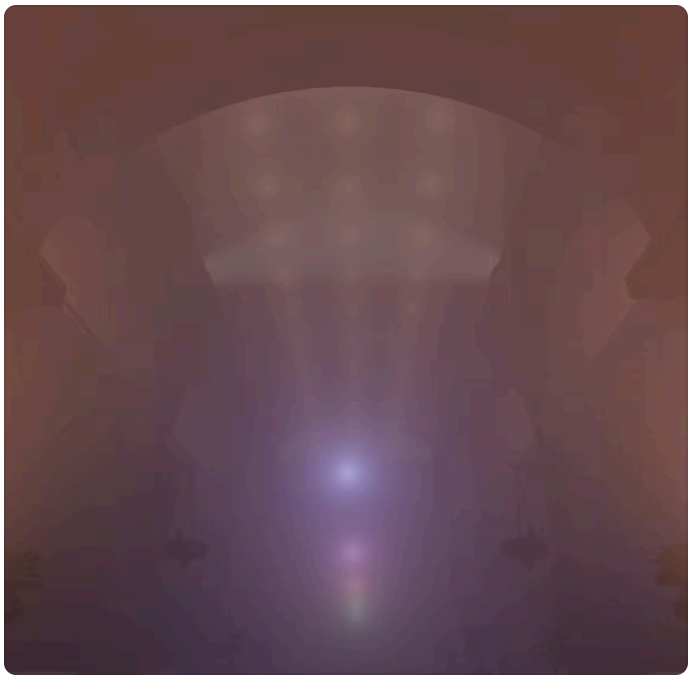
Fog Density 0.3

Fog Density Per Light Modifier

Defines the multiplication factor for the global density value when used evaluating lights fog density. This can be useful to “slow down” light disappearance compared to other geometry.

Fog Scattering

Defines global fog light scattering. Higher values make volumetric lights scatter (”blur”) by distance.



Fog Scattering 0.015

Fog Scattering 0.0

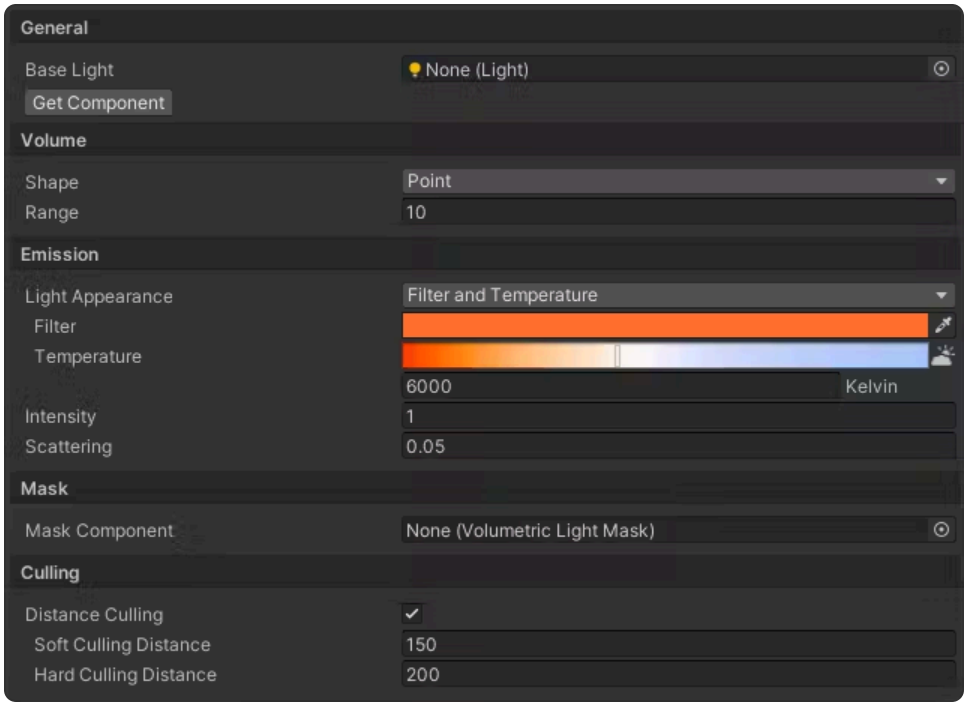
Fog Color

Defines global fog color.

⚠ Fog is blended using traditional alpha blending.

Volumetric Light

Volumetric Light component is the core component used by the AVL systems. It acts similarly to the **Light** component. **Volumetric Light** component can be connected to the **Light** component, but this is not required.



Base Light

Base **Light** component which will be used for settings copy. Might be null.

Copy Settings

Enabled settings sync with the base **Light** component every frame. This might degrade performance.

Copy From / To Light Component

Copies settings from / to the base **Light** component.

Shape

Volumetric shape used when rendering this light. See **Light Shapes** section for more information.

Range

Light range used for culling and rendering. Defines how far the light will be visible from its origin.

⚠ In real world lights do not have max range and contribute to the lighting infinitely according to the inverse square law. Therefore, high range values might appear more realistic but also greatly increase performance impact.

Outer Angle

Spotlight's outer angle.

Inner Angle

Spotlight's inner angle. Should be less than **Outer Angle**.

Rect

Area light's surface rectangle.

Light Appearance

Defines what color mode to use - a color value or a color temperature. This is very similar to the Unity's **Light** component.

Color / Filter

Light color / filter used for rendering. In filter mode, temperature color is multiplied by the **Filter** parameter resulting in visible color.

Temperature

Light color temperature used for rendering.

Intensity

Light intensity (brightness) used for rendering. Internally, this value is multiplied by the **Air Light** parameter.

Scattering

Light scattering used for rendering. This parameter scatters ("blurs") lights lowering their visible brightness which can be useful for "fog" areas.

Mask Component

Volumetric Light Mask component which contains this volumetric light. Each light may have only one mask, but each mask may have many lights. See **Volumetric Light Mask** section for more information.

Distance Culling

Enables light culling by camera distance. Culled lights will not be processed by the GPU.

Soft Culling Distance

Used for distance culling. Defines camera distance at which the light will start slowly disappearing.

Hard Culling Distance

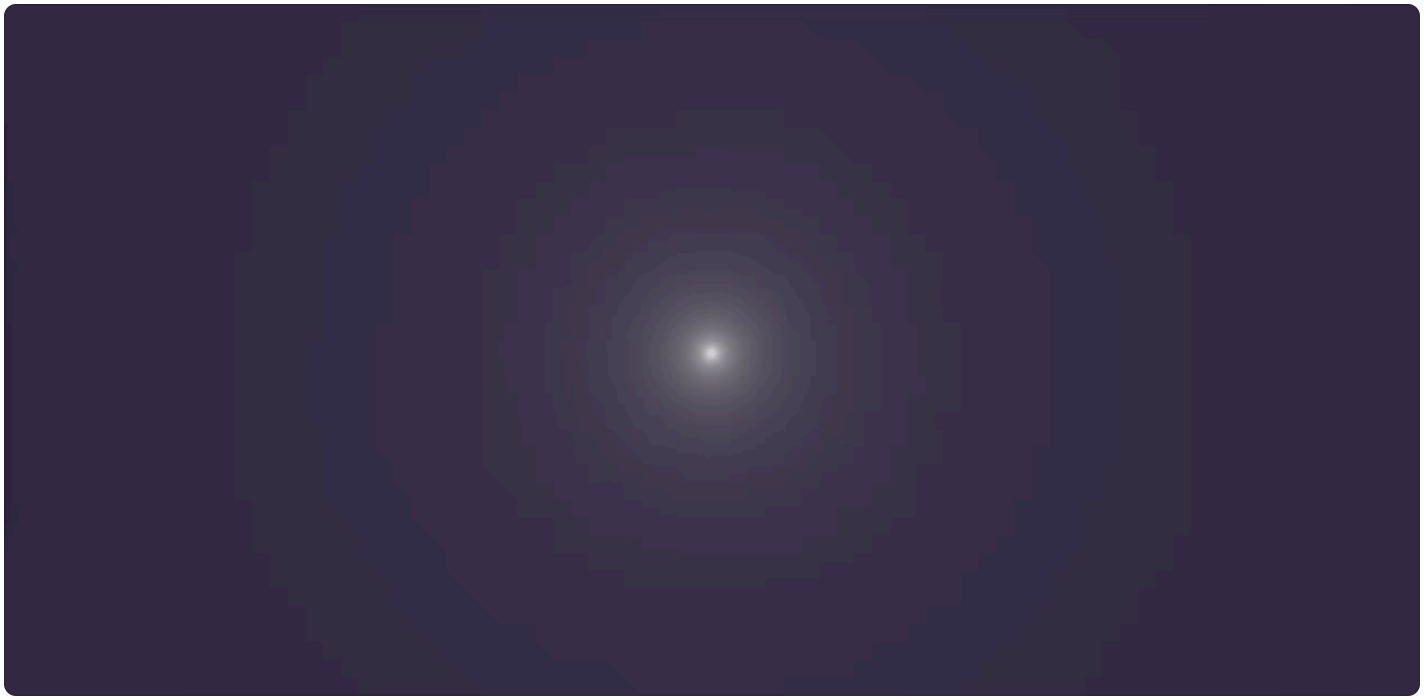
Used for distance culling. Defines camera distance at which the light will be completely culled.

Light Shapes

Point

Represents a punctual light evenly distributed in all directions. The simplest form of a volumetric light (excluding uniform volumes) requiring few calculations.

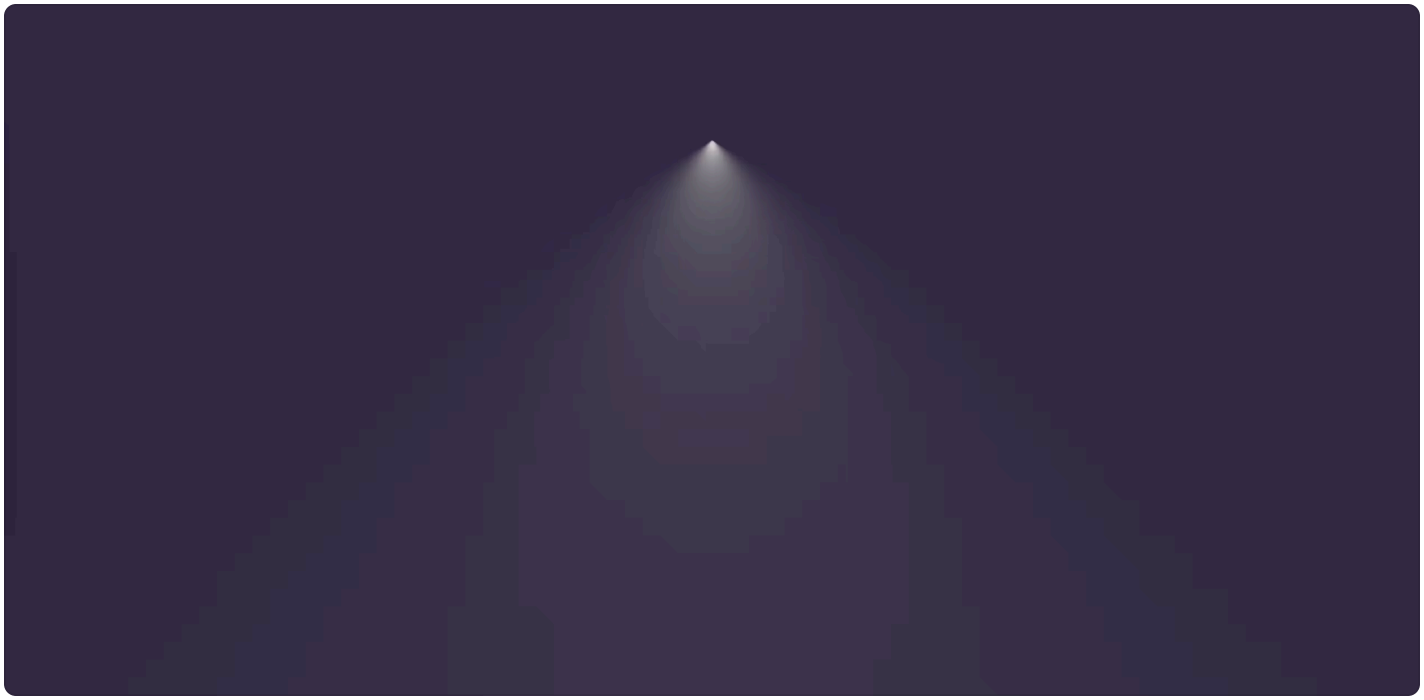
🟢 **Low** performance impact



Spot Soft Edge

Represents a spotlight with a soft-edge falloff. Requires the greatest number of calculations to be made per-pixel.

🟡 **Medium** performance impact



Spot Hard Edge

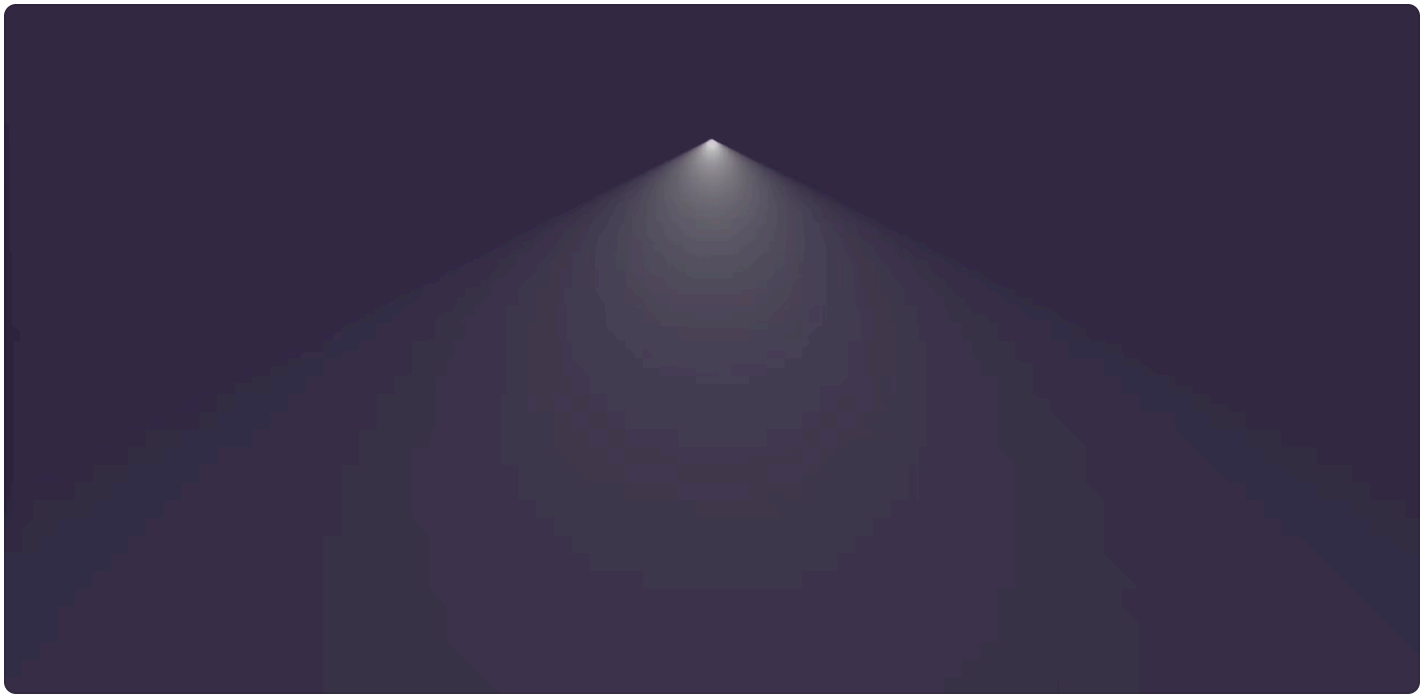
Represents a spotlight with a hard-edge falloff consisting of two cone lights.

🟡 **Medium** performance impact

Spot Hard Edge Single

Represents a spotlight with a hard-edge falloff consisting of one cone light. Similar to **Point** shape but requires extra cone intersection calculations.

🟢 **Low** performance impact



Area Hard Edge

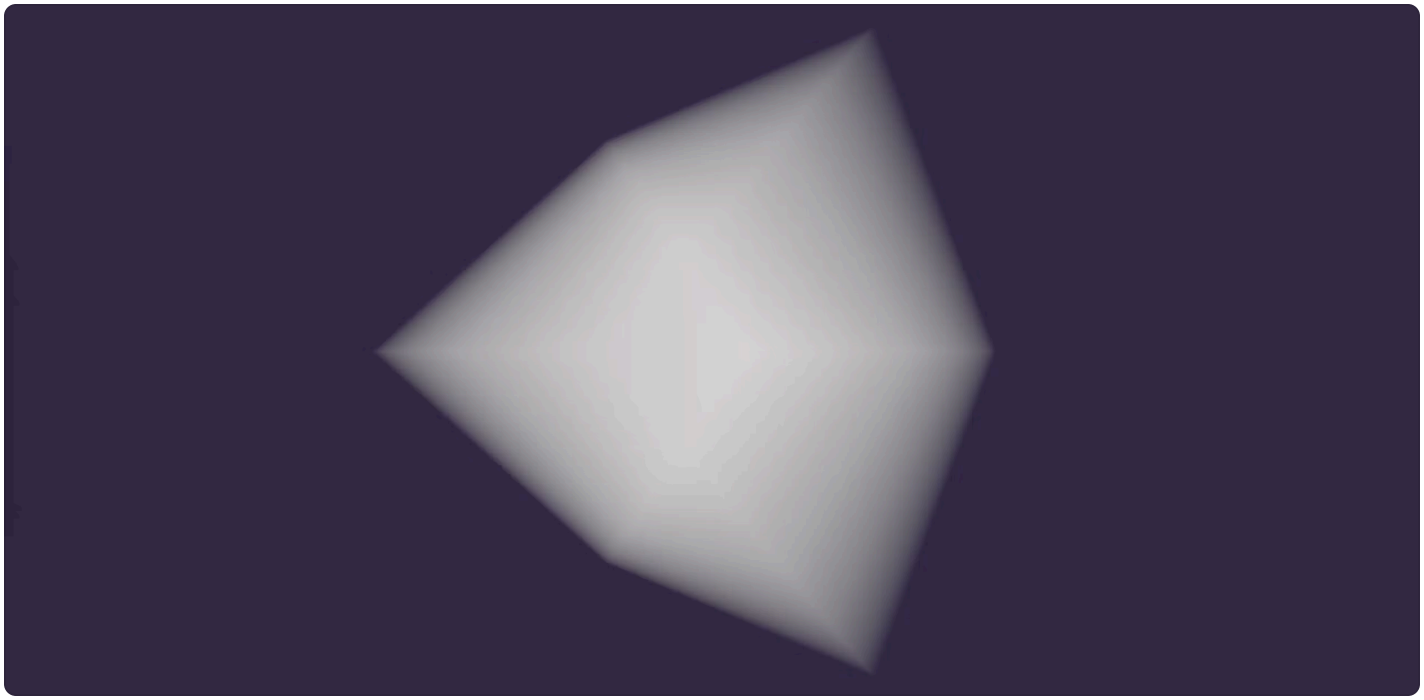
Represents an area of light defined by the **Rect** parameter. Very useful for portals, windows, displays.

🟢 **Low** performance impact

Uniform Box

Represents a uniform box of light. Width (X scale) and height (Y scale) are defined by the **Rect** parameter. Depth (Z scale) is defined by the **Range** parameter. Note that they are “radius” values and not “diameter” values.

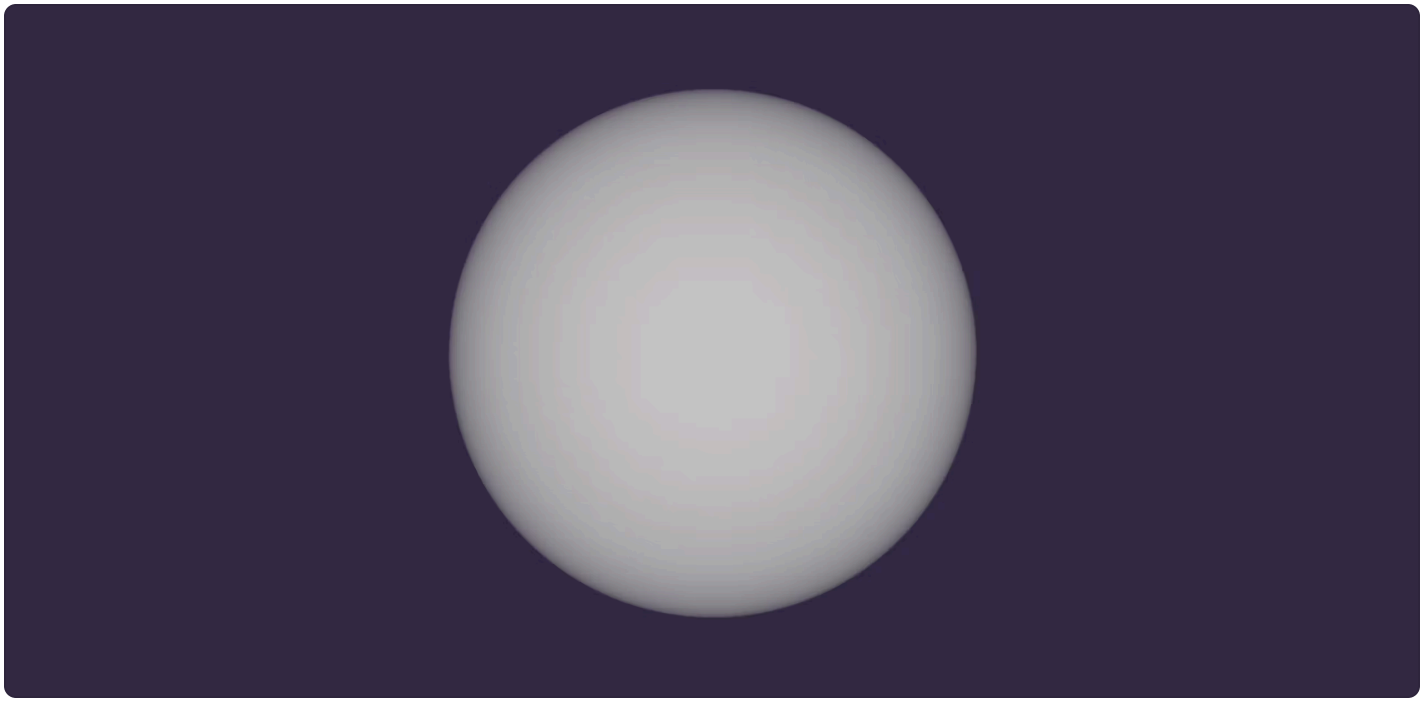
🟢 **Low** performance impact



Uniform Sphere

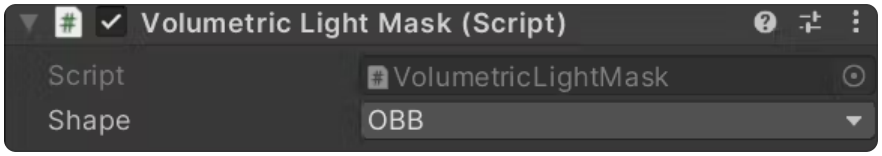
Represents a uniform sphere of light.

🟢 **Very Low** performance impact

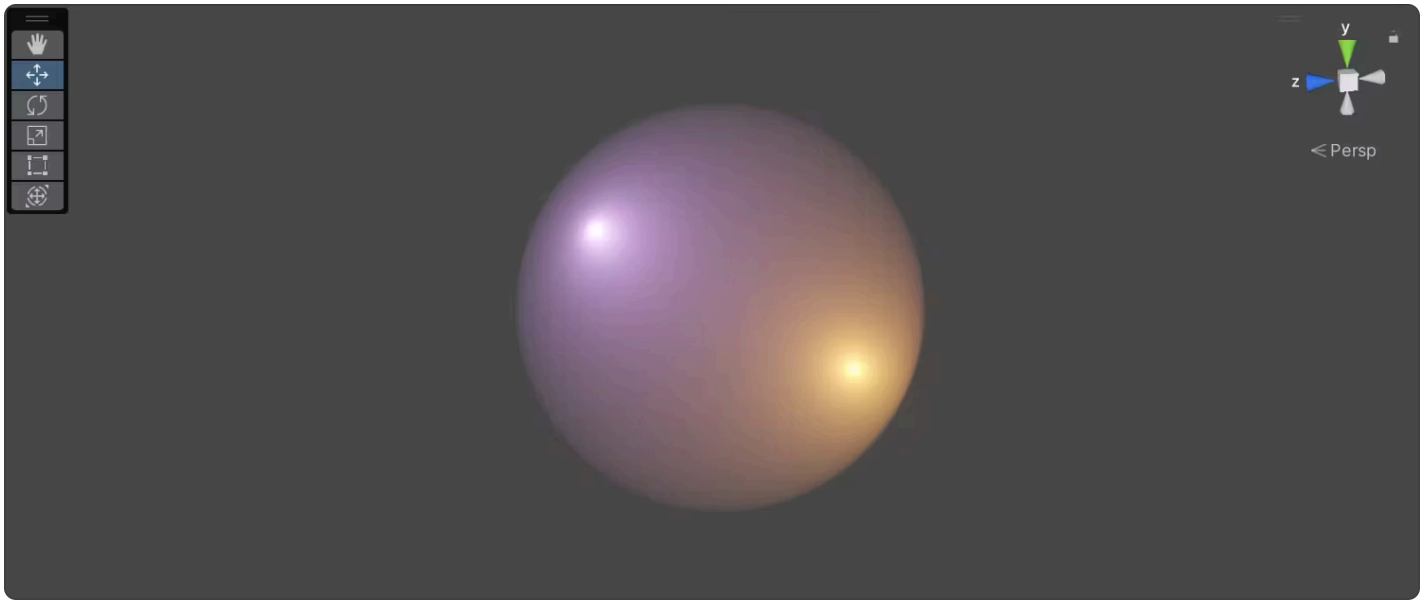


Volumetric Light Mask

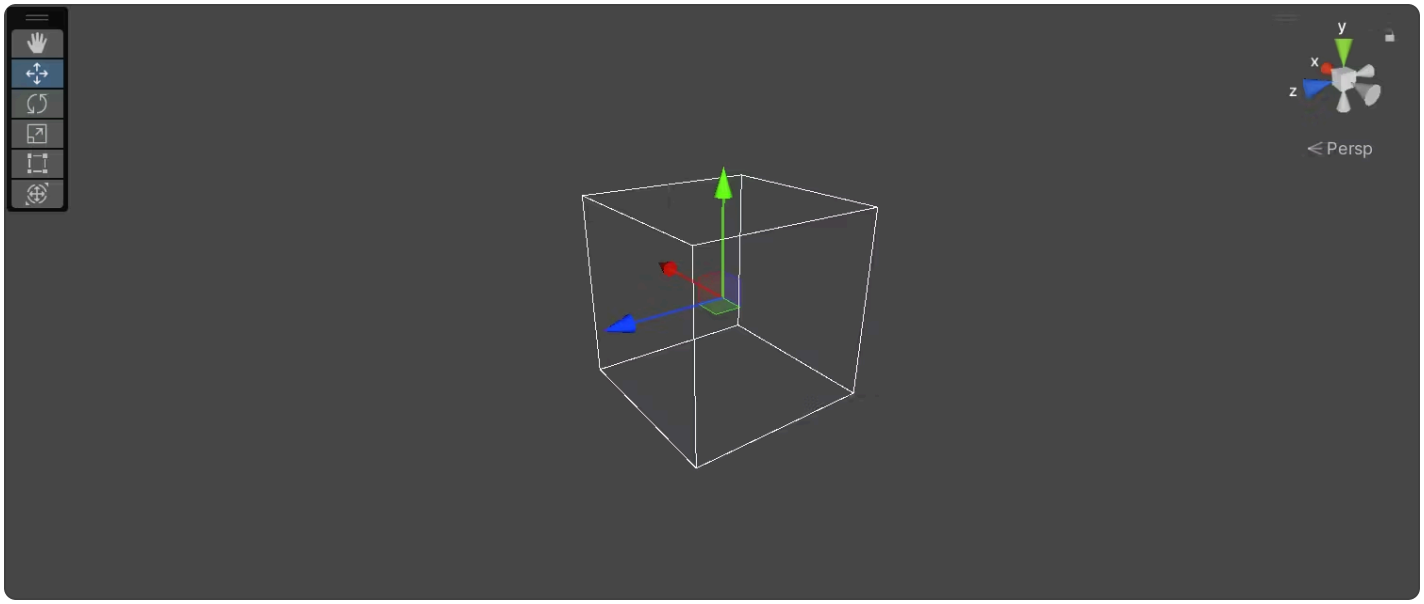
Volumetric Light Mask component is an additional AVL component which enclosures volumetric lights inside simple shapes such as **OBB** (Oriented Bounding Box) or **Ellipsoid**.



Only one **Volumetric Light** component can reference only one **Volumetric Light Mask** component, but many **Volumetric Light** components can be referenced to the same **Volumetric Light Mask** component.



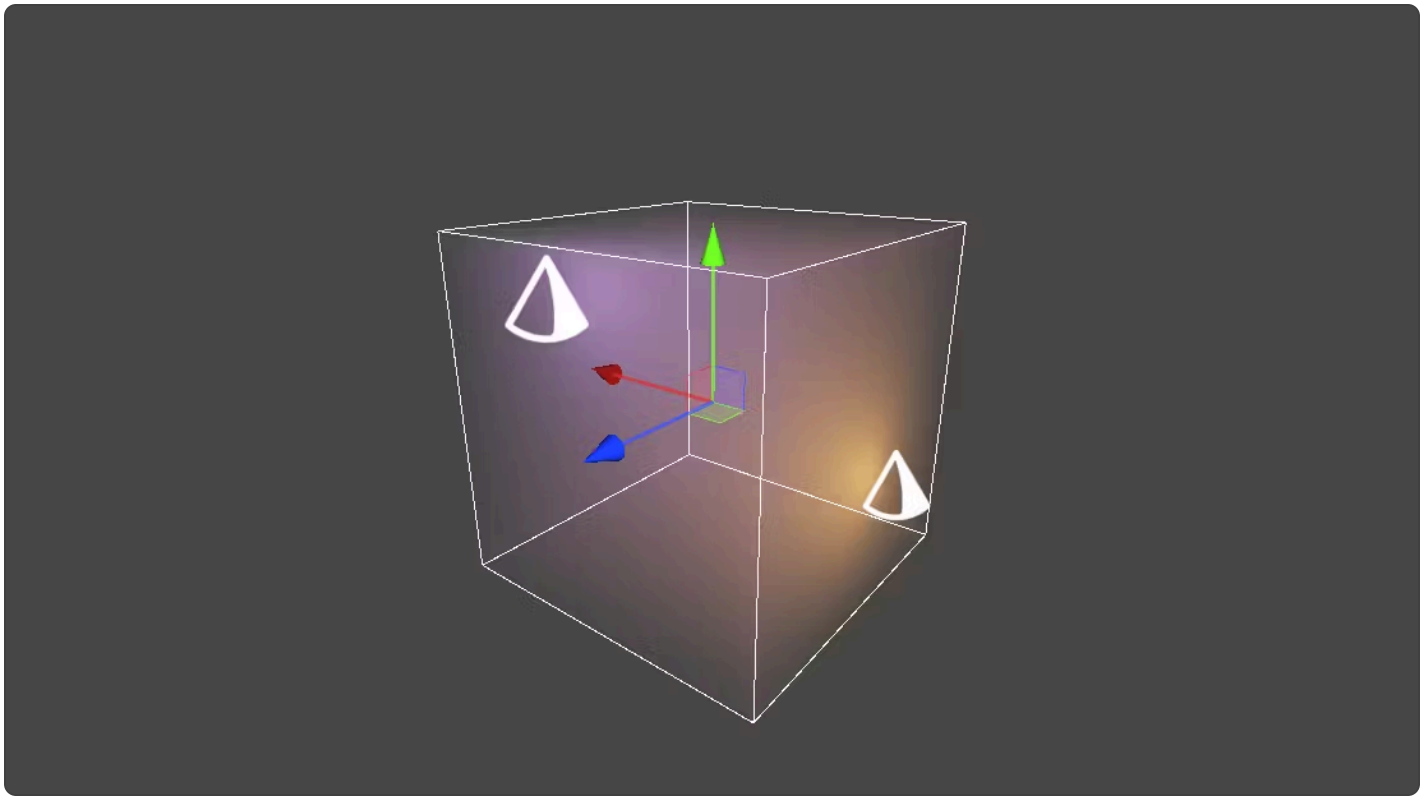
Volumetric Light Mask components are visible as scene gizmos when selected.



Mask Shapes

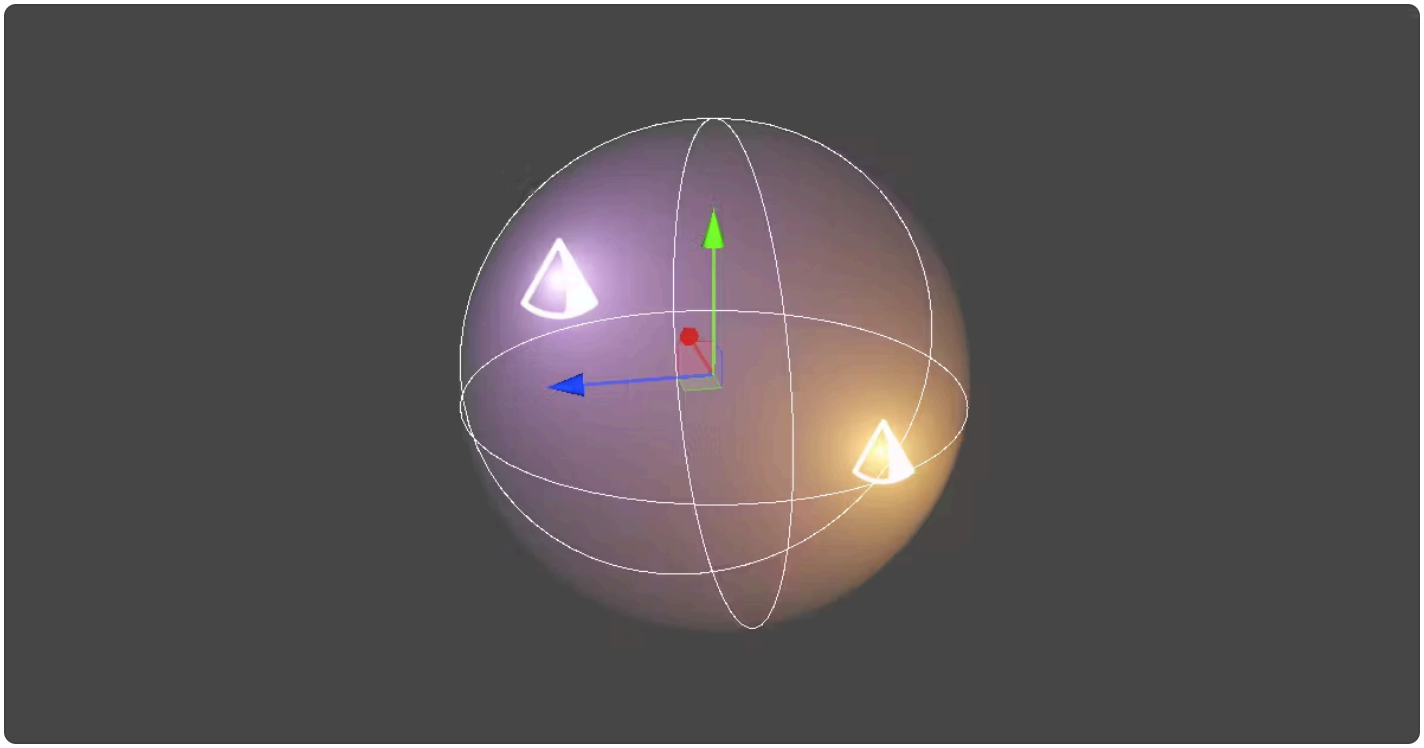
OBB

Represents an oriented bounding box (scalable in any direction).



Ellipsoid

Represents an ellipsoid (scalable in any direction).



🔧 Troubleshooting

Not Working

❔ **Volumetric Lighting is not visible.**
❕ Check if your first-time setup is done correctly. Refer to the **First Setup** section.

Performance

❔ **How can I see the performance impact of AVL?**
❕ Use Unity's Profiler. AVL is made to be shown in the profiler correctly.

❔ **I think I have too many lights in the scene. How can I debug what areas are the densest?**
❕ Use **Light Count** debug mode. This will show the light count per cluster in the game & scene view.

Artifacts

❔ **I can see noise when using soft edge spotlights.**
❕ This is a known issue. Refer to the **Spot Soft Edge** section.

❔ **I can see some blinking happening when I use a lot of lights at the same place.**
❕ You might be exceeding the max light count per cluster. Use Light Count debug mode if that is the case. Areas having the exceeding light count will be shown strongly pink-colored. The limit can be raised if needed, refer to the **Max Lights** section.

❔ **When I use feature [X], it looks off.**
❕ Please use GitHub issues repository to describe the problem.

Other

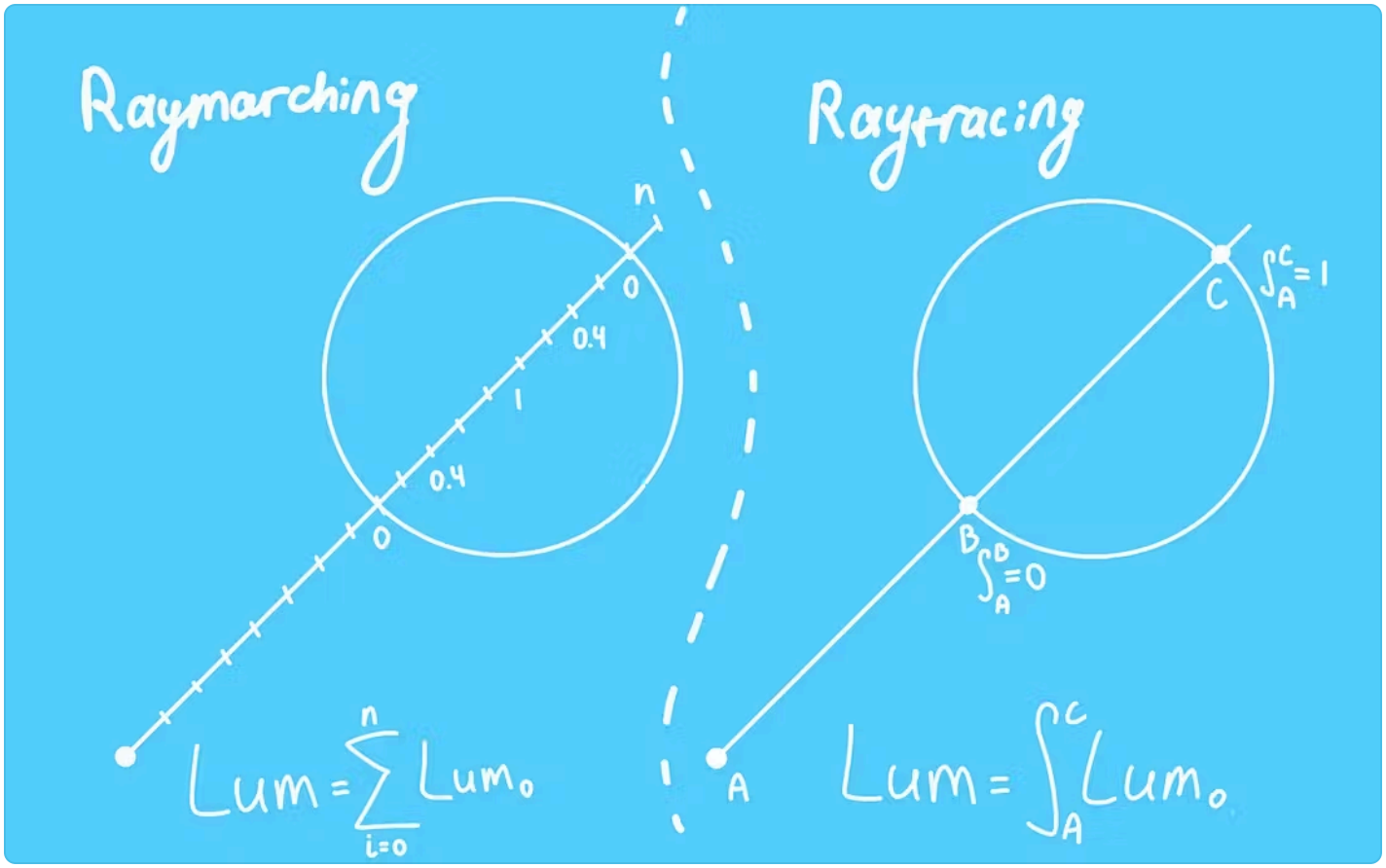
❔ **I have a suggestion!**
❕ Please use GitHub issues repository to describe the suggestion.

📖 Concepts

Analytical Lighting Overview

Volumetric lighting can be described as a screen-space visual effect based on the property of thick air to interact with visible light rays, creating volumetric shapes of light.

Simply put, this effect can be depicted as a lighting equation for each of the per-pixel camera rays. This equation should be integrated, and result can be used as the final color blended into the camera color texture (camera color target). The integration can happen basically in two ways: **raymarching** and **raytracing**.



Traditionally, volumetric lighting is calculated using raymarching technique, which is based on traversing the light ray iteratively. This is a quite simple and straightforward approach which also can be used for complex light equations accounting for shadows and procedural noise. Though, it is also quite computationally heavy.

AVL employs the concept of mathematical analysis, or in other words, raytracing. Instead, lighting equation can be integrated, and no further iterations are needed. This heavily increases the performance as well as fixes some other raymarching problems such as noise filtering and draw distance. But some complex light equations cannot be reasonably integrated.