

Manual **redLights**

Duisburger Straße 117

D-40479 Düsseldorf

Tel: +49 0211 17806259

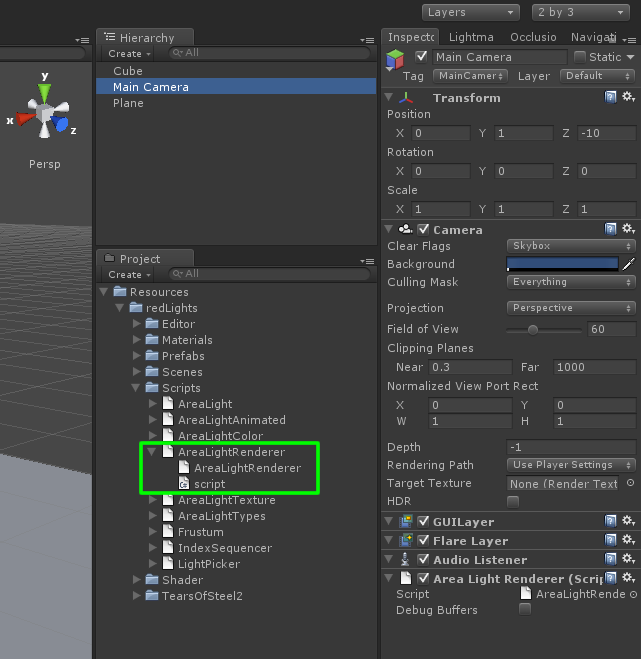
Fax: +49 0211 17807420

http://www.redPlant.de

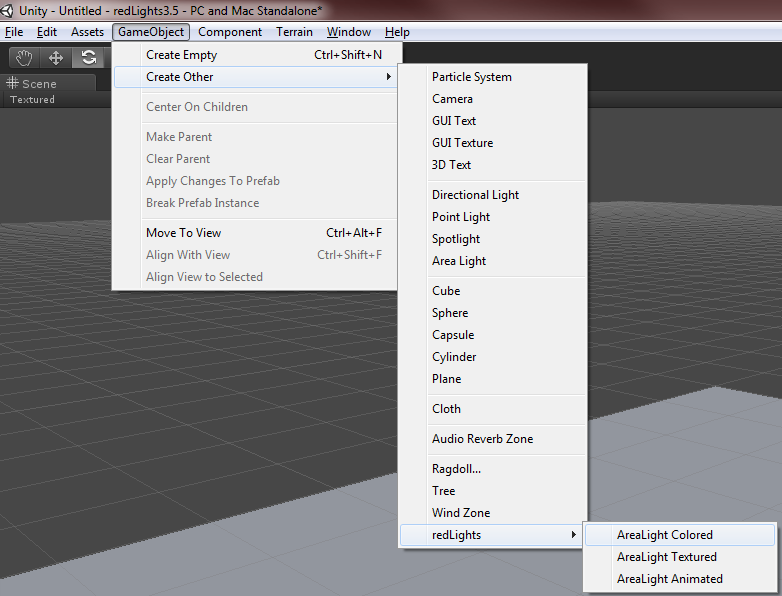
contact@redPlant.de

## How to add a single colored light to your scene

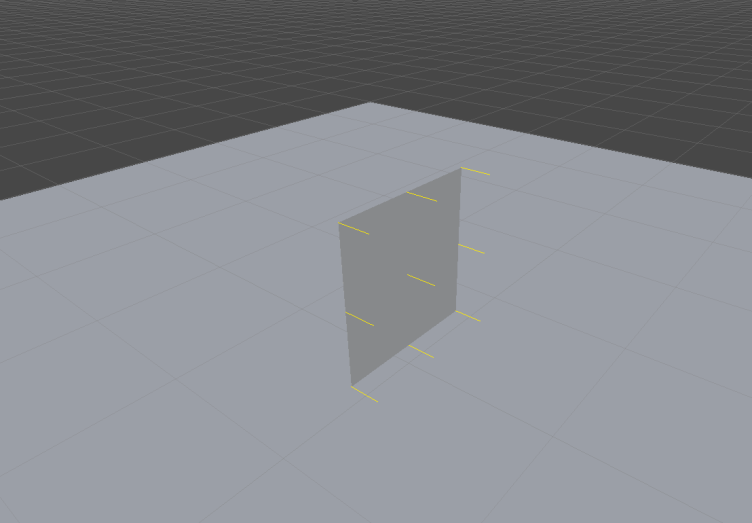
# Add the ***AreaLightRenderer*** component to your main camera, if there is none.



# Click on ***redLights*** / ***AreaLightColored***



# This will add a new single colored light to your scene with the name “***AreaLightColored***”.



# That’s it.

### How to add a textured light to your scene

### Add the *AreaLightRenderer* component to your main camera, if there is none

### Click on *redLights* / *AreaLightTextured*

### This will add a new single colored light to your scene with the name “*AreaLightTextured*”.

### Assign a texture.

### That’s it.

### How to add an animated light to your scene

### Add the *AreaLightRenderer* component to your main camera, if there is none

### Click on *redLights* / *AreaLightAnimated*

### This will add a new single colored light to your scene with the name “*AreaLightAnimated*”.

### Set the Image Path variable to the resource folder, where the image sequence is located. This path must be accessible at runtime, thus located in a Resource subfolder.

### That’s it.

**Description of values and parameters**

* **AreaLightColored Component**

This is the base component for the different area light types. It will use a single color for illumination.

**Color** and **Intensity** are probably self-explanatory. The other parameters will be explained below:

**Render Diffuse:** This checkbox will activate/deactivate the rendering of the diffuse light.

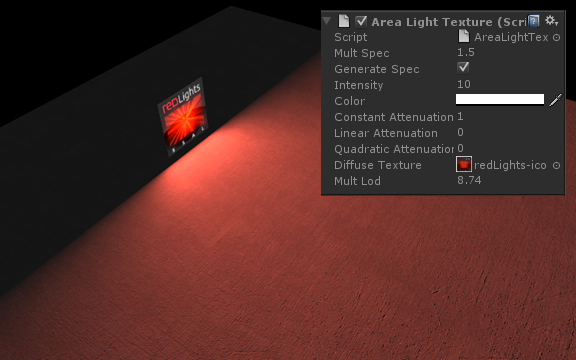
**Render Specular:** This checkbox will activate/deactivate the specular reflection of this light.

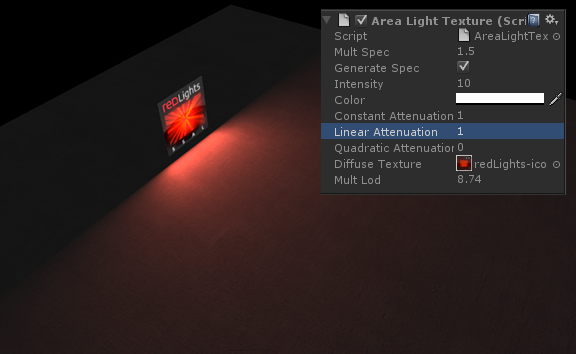


**Mult Spec**: This value can be used to control the strength of the specular reflection. Reflections will be generated on specular materials only.



**Constant, Linear, Quadratic:** The three attenuation parameters control how the light intensity diminishes with distance from the light source.





* **AreaLightTextured Component**

In addition to AreaLightColored, this component has two more parameters:

**Diffuse Texture:** This texture will be assigned to the emitter and is also used for illumination. It must have mip-maps enabled. During lighting, the diffuse texture is sampled and the color is multiplied by the light color.

**Mult Lod:** In order to get soft diffuse lighting without hard defined details, lighting calculation is accessing the diffuse texture mip maps. This parameter controls the mip map level.

* **AreaLightAnimated Component**

In addition to AreaLightColored, this component has four more parameters:

**Image Path:** This variable must point to a resource subfolder, where the image sequence is located. For example “TearsOfSteel2”. When starting, the images from that folder are loaded into an array and are then ready for playback. Note, the images from the sequence must have mip-maps enabled.

**Mult Lod:** In order to get soft diffuse lighting without hard defined details, lighting calculation is accessing the diffuse texture mip maps. This parameter controls the mip map level.

**FPS:** Frames per second. The speed of the playback.

**Play On Start:** Wether or not the animated texture starts to play on application start. If deactivated, playback can be controlled via the public methods (Play, Stop, Rewind, Reset).

* **AreaLightSpherical Component**

This component acts as an omnidirectional light, that emits light equally in all directions. Right now it supports only monochrome color. In a future version we might implement support for textured spherical emitter.

* **AreaLightIESPoint Component**

This component acts as an point emitter. In addition to the basic distance falloff it also takes into account IES light profiles, that define how much light is present at certain angle around a given lightsource. In order to use your own IES files, you must make sure, that they are formatted in a way, that is compatible with our parser, since we do **not** support **all** of the various version of IES formats. To convert an IES into the appropriate format, download iesgen\_4 from [http://rip3d.net/Downloadmodels/**iesgen\_4**.rar](http://rip3d.net/Downloadmodels/iesgen_4.rar).

Use this tool to open and then export your file. While exporting iesgen\_4 will format the input data properly so can be used with the redLights.

The converted files are expected to be located in the “redLights/Resources/IES/” subdirectory. To apply an imported IES file to an AreaLightIESPoint component, simply type in its name into the “IES File“ input field.

**How to integrate custom shaders with redLights**

* For an example of a custom shader/material take a look at the files in the “redLights/Resources/CustomMaterials” subdirectory.
* Define a custom rendertype tag in your own custom shader

Tags { "RenderType"="CustomRendertype" }

* Add new custom rendertype to replacement shader for generating renderbuffer (normal, albedo, transparency and depth)

**Normal-Buffer**

* Open the file “redLights/Resources/Shader/WorldNormal.shader”
* Add an appropriate subshader for your custom rendertype.

**Albedo-Buffer**

* Open the file “redLights/Resources/Shader/UnlitAlbedo.shader”
* Add an appropriate subshader for your custom rendertype.

**Transparency-Buffer**

* Open the file “redLights/Resources/Shader/UnlitTransparency.shader”
* Add an appropriate subshader for your custom rendertype.

**Depth-Buffer**

This last one is a bit more involved, since you need to manipulate one of the internal shader files, that ship with unity.

* Download the appropriate built in shader package for your unity version from <http://unity3d.com/unity/download/archive>
* Find the file „Camera-DepthTexture.shader“ and copy it into your unity project folder.
* For example directly into the redLights folder “redLights/Resources/Shader”.
* Restart Unity3D.
* After restart Unity will use “Hidden/Camera-DepthTexture” shader from the source file that it finds inside the project folder and ignore the built-in hidden one.
* Add an appropriate subshader for your custom rendertype.
* **Make sure that you stick to the standard naming convention for material properties.**

For example, **the main color** of your material should be called:

“\_Color ("", Color) = (1,1,1,1)”

and not

“\_Colour ("", Color) = (1,1,1,1)” or

\_MainColor ("", Color) = (1,1,1,1) and so on…

The **main texture** should be called

The **normal map** should be called

“\_BumpMap ("Normalmap", 2D) = "bump" {}”

**Shininess** should be called:

“\_Shininess ("Shininess", Range (0.01, 1)) = 0.01”

**Configuration**

* **Location of arealights in menu**

If you would like to change the default menu location “redLights” to something different such as “GameObject/Create Other/AreaLights” open the file *AreaLightFactory.cs* and adjust the variable *RootMenu* accordingly.

**How to use redLights with the Oculus Rift**

* Download the OculusUnityIntegration.unitypackage from<http://everythingoculus.co.uk/oculus-sdk/>
* Replace your main camera with either OVRCameraController or OVRPlayerController prefab.
* Inside the prefab locate the two gameobjects “CameraLeft” and “CameraRight” and add the AreaLightRendererRift component to both of them.
* That’s it, you can now use our lights as you normally would.
* **Known Issue**: In the editor mode while not being in play mode the lights might be switched vertically. If you switch to play mode everything should work correctly. This is a temporary issue and will be fixed in a future version.