Sonic Ethers'

Natural Bloom & Dirty Lens

Image Effect for Unity3D

User Guide v1.0

Copyright © Sonic Ether 2014

1. Introduction

Thank you for purchasing SE Natural Bloom & Dirty Lens for Unity3D! You now have access to a cutting-edge image effect that can give your game an aesthetic boost and greatly improve the perception of high-dynamic-range lighting.

1.1 Summary

SE Natural Bloom & Dirty Lens (SENBDL) is an image effect that simulates the sub-surface scattering that occurs inside a lens. Traditional bloom shaders attempt to approximate this phenomenon by additively blending a simple Gaussian-blurred image of the original scene. SENBDL uses a much more intelligent approach, yet, renders nearly as quickly as Unity's included bloom effect. It achieves very large, perfectly smooth, and natural looking highlights from visible objects sufficiently brighter than surrounding objects without using physically incorrect techniques like "threshold".

SENBDL also utilizes the calculated scattered light to render a "dirty lens" effect. Included are 5 lens dirt textures, however, you can use any texture you'd like to represent lens dirtiness.

1.2 Compatibility

SENBDL works in any scene with any objects and any shaders within. All this effect requires is a high-dynamic-range of light to provide a large enough difference in brightness for the effect to be visible. This package includes an emissive shader that can output very bright colors. You will also find a tutorial in this guide on how to edit any surface shader to output brighter light if you are having trouble creating a wide enough range of brightness.

2. Usage Guide

This section will help you to setup SENBDL in your Unity game and will provide tips for implementation so that your game can make the most out of this image effect.

Even many professional game developers misuse and misunderstand bloom. Bad trends that started in the early 2000's due to hardware limitations are still alive and well today. Hopefully, this guide will dispel some popular myths in the industry regarding how bloom actually works in real-life and how to best approximate it.

2.1 Adding the Effect to Your Game

Follow these steps to get SE Natural Bloom & Dirty Lens up and running in your project!

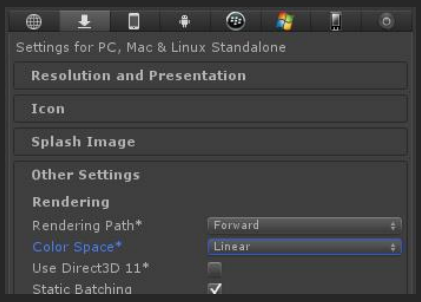
Step 1: Set Color Space to Linear

Calculating lighting in gamma-space is probably one of the most crucial mistakes visually a game can make. Make sure that your game's lighting isn't in gamma-space!

1. From the **Menu Bar**, go to **Edit** > **Project Settings** > **Player**

2. In the **Inspector**, select the icon respective to whether your game will be played using the **Unity Web Player**, or the **Standalone Player**

3. Under **Other Settings**, select the **Color Space** dropdown and select **Linear**

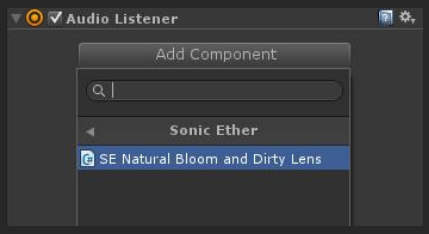


Step 2: Enable HDR on Your Main Camera

If your main camera is not rendering in HDR (High-Dynamic-Range), well, you may as well not be using SENBDL! The real world has a grand range of intensities of light, and so should your game! HDR makes this possible by increasing the precision of the color buffer on your main camera.

1. Select your **main camera** in the **Hierarchy** and, from the **Inspector**, click the **HDR** checkbox

Step 3: Add SE Natural Bloom & Dirty Lens Component to Your Main Camera



Any image effects placed before SENBDL will contribute to the overall effect properly. Make sure that you are not tonemapping before applying SENBDL.

1. Select your **main camera** in the **Hierarchy**, and from the Inspector, click the **Add Component** button.

2. In the **drop-down menu**, select **Image Effects** > **Sonic Ether** > **SE Natural Bloom and Dirty Lens**

3. If you already have other image effects added to your main camera, ensure that SENBDL is after any image effects that alter the contents of the scene directly like fog or reflections, and before any tonemapping.

Step 4: Select a Lens Dirt Texture

If you don't want a dirty lens, go ahead and skip this step.

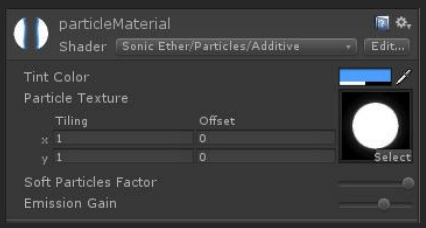
1. In the **Inspector**, in the main GUI of SENBDL, **select a texture** to represent lens dirt by pressing the icon to the right of the object window. All lens dirt textures that are included with SENBDL are located in the following **directory** in your project: **Sonic Ether/Natural Bloom/Lens Textures**

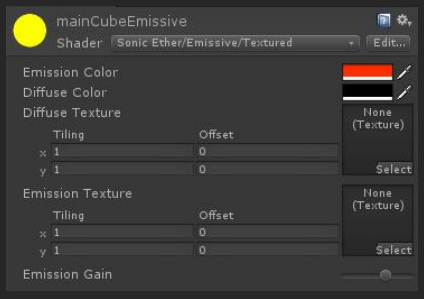
The names of the textures included begin with "lensDirt". It can be easier to find these textures by typing "lensDirt" into the search box at top of the texture selection window.

It is also highly recommended to add a tonemapping image effect after SENBDL, though, it is not necessary. Tonemapping can help to compress a high dynamic range of light to suit the limited range of light that display devices can output.

2.2 Using the Included Shaders

The included shaders will help you create objects and particles bright enough to glow brilliantly in your scenes.





The included emissive shader can be found when selecting a shader for a material in **Sonic Ether** > **Emissive** > **Textured.** The included additive particle shader can be found when selecting a shader for a material in **Sonic Ether** > **Particles** > **Additive.**

Both of these shaders have an "Emission Gain" property that boosts the emissive output of the shader.

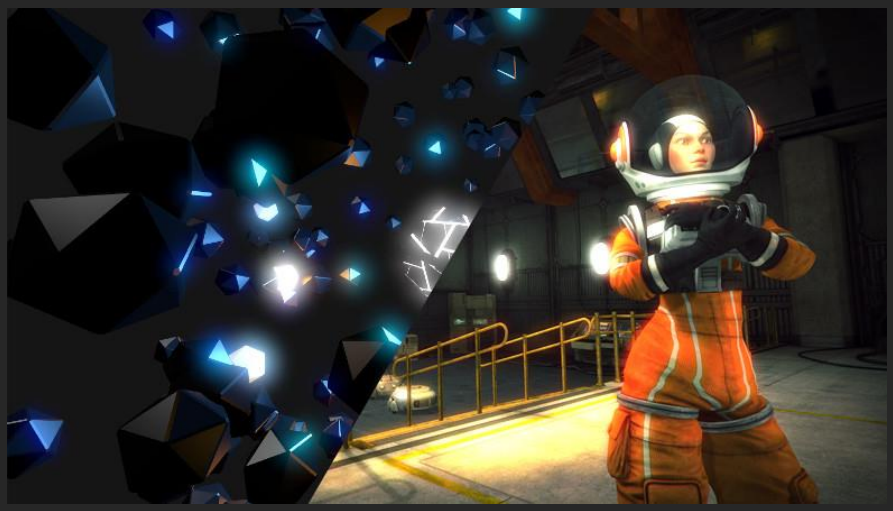
At the slider's leftmost position, the output luminosity is unaffected (1.0). Dragging the slider to the right will increase the brightness of the particle or emission. The surface shader includes a texture input for a diffuse and an emission texture. The emission texture will control RGB intensity of emission. You may choose not to use a diffuse or emission texture. Both default to white when no texture is selected.

It is important to be careful with the Emission Gain slider on the particle shader. Unity doesn't seem to like when particles get too bright, and the whole screen will go black. Adjust this parameter with caution. If you find yourself needing brighter particles than Unity will allow, it is recommended that you lower the brightness of everything else in your scene and increase exposure.

2.3 Effectively Utilizing SENBDL

This section includes things that you should know that will help you make the most of a high-quality bloom solution like SENBDL in your games and help you to understand where most go wrong so you can make the right decisions as a game developer.

Bloom doesn't have a very good reputation. In the early 2000's, when bloom was first being used in games with low-precision color buffers (8 bits per channel), a lot of shortcuts had to be made. Low-precision color buffers did not allow for high-dynamic-range lighting in a scene. Back then, you pretty much had to make the best use of the entire range of the color buffer that you could, which meant that having objects that are much brighter than most others was infeasible. Game developers overcame this limitation by developing bloom effects that utilize a threshold or "bright-pass" that ensured that objects under a certain brightness would not affect the bloom shader. To make matters worse, any color clipping that occurred (colors exceeding the maximum limit) meant hue-shifting and lost information made more apparent by bloom. This causes a look that I think we've all learned to hate.



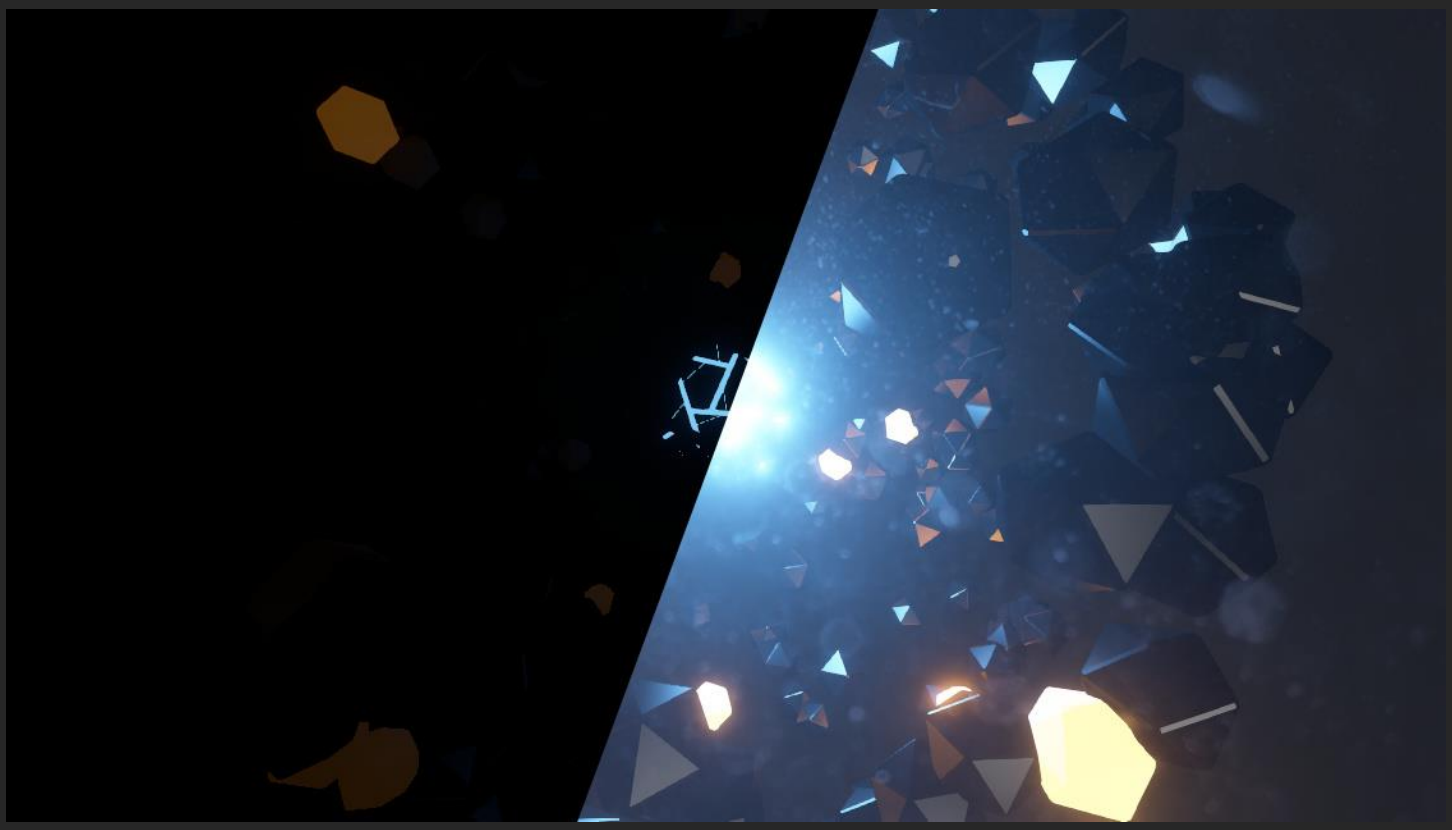
Beautiful, if you like hue-shifted globules all over your game. Takes me back to the good ol' days of LDR rendering.

It's time for a change. HDR color buffers have been a possibility for nearly 8 years, so why are they still not being utilized properly? Perhaps it can be likened to the saying, "Old habits die hard." I think the main cause is a general lack of understanding of the phenomenon that bloom is attempting to approximate.

Bloom is an approximation of light scattering inside a lens. It happens in the human eye, as well as cameras (to varying degrees). Your camera/eye doesn't care how bright light is. It's all scattered, reflected, and refracted the same no matter how bright or dim it is. *Reality doesn't use threshold bloom.* So, why is bloom only visible from bright objects? Well, technically, it's "visible" at all times, but with light not sufficiently brighter than surrounding light, it is unnoticeable to the eye.

The real world displays an expansive range of light intensity, from candlelight to sunlight. Display devices are only able to output a fraction of that range. Many game developers and lighting artists tend to generally pick intensities of light that fit well within the limited range of light that a monitor can output. The problem that this creates is that, when applying photorealistic phenomena like bloom or similar effects, there simply isn't enough range of light to provide a realistic look. **If you're going to want a photorealistic or filmic look to your game, the first and most important thing to get right is high-dynamic-range lighting.**

Here's the bottom line. If you load up a scene and slap SENBDL on the main camera, and the scene was not crafted with the above in mind, you probably won't even be able to see the effect. This is, in fact, a good thing! You don't see bloom everywhere you look, you only see it in situations where something is sufficiently brighter than its surroundings. *If you want something to glow or make lens dirt visible using SENBDL, you have to make it bright!*



Consider this comparison image. Right of the split is the intended exposure level, and left of the split is a very low exposure. You can see just how much brighter the blue emitting from the center cube is in comparison to the surrounding objects. The unilluminated icosahedrons are pretty much invisible because they are so much darker than the center object. The center cube shines so brilliantly because it is much brighter than the surrounding objects.

3. Support

If you need support for this product or wish to provide suggestions, please email me (Cody Darr aka Sonic Ether) at cody@sonicether.com and I'll be happy to assist you!