# Highlighting System 3.0

## User Guide

# Table of Contents

# 1 Changelog

v3.0

- In this version, highlighting occluders doesn't work with Sprites! This might be fixed in the future releases of the Highlighting System, but you shouldn't upgrade in case you heavily rely on this feature in your project
- Mobile optimizations (9 FPS vs 25 FPS on iPhone 4)
- Highlighting occlusion feature (highlighters is now occluded with scene objects without the need to add highlighting occluders to all of them). Not compatible with hardware anti-aliasing!
- Per-*Highlighter* see-through mode (controls when the highlighting should be always visible)
- Hardware anti-aliasing (MSAA) support (highlighting buffer is now also anti-aliased. *RenderTexture* anti-aliasing support was introduced in Unity 4.2)
- Support for nested highlighted objects (previously it was causing an error)
- Invisible highlighted objects culling (they will not be affected by the material replacement routine. Scenes with huge amount of highlighted objects now should work faster)
- Depth *Offset Factor* and *Offset Units* settings added to avoid visual artifacts when *Dynamic Batching* is enabled in *Player Settings*
- Real stencil buffer is now used when rendering highlighting buffer (speeds up rendering. Stencil buffer access was introduced in Unity 4.2)
- Lightmaps compatibility
- *_CameraDepthTexture* / *_CameraDepthNormalsTexture* is no longer cleared when the *camera.depthTextureMode* property is set to *DepthTextureMode.Depth* / *DepthTextureMode.DepthNormals*
- *RenderTexture* restore operations avoided in most of the cases and "Tiled GPU perf. warning" is suppressed in all other cases. Enable *DEBUG_ENABLED* define in *HighlightingBase.cs* script to see when this happens
- Null reference exceptions now prevented in case highlighted *GameObject* or *Renderer* was removed, but *ReinitMaterials()* wasn't called
- Fixed one empty pixel border around highlighted objects on a devices without support for NPOT (non power of two) textures
- Fixed one texel vertical offset in Direct3D 9
- Coroutines, used in *HighlightableObject* (*Highlighter*) was replaced with simple frame number comparision
- Combined highlighting shaders. Fixed function states (ZWrite, ZTest, etc.) is now driven by the material parameters (feature was introduced in Unity 4.3)
- Events/delegates used to control *HighlightableObject*'s (*Highlighter*'s) state from *HighlightingEffect*'s (*HighlightingRenderer* / *HighlightingMobile*) was replaced with *Highlighter* components management
- Added *HighlightingSystem* namespace (to avoid potential name conflicts)

v2.0

- Linear blending of the highlighting and frame buffers (gives true highlighting colors)
- All shaders now compatible with the Highlighting System out of the box (no need to adapt each custom shader anymore)
- Batching and shared materials support
- Correct highlighting of transparent materials
- Highlighting occluders
- Handy highlighting effect quality and intensity controls with presets
- Effect inspector helpers (will help you correctly setup Highlighting System in your project)
- Bug fixes, shaders optimizations and other performance improvements

v1.1

- Improved folder structure (highlighting scripts moved to *Plugins* folder). Now it's possible to use Highlighting System from JavaScript and Boo (see *JSHighlightingController.js* and *BooHighlightingController.boo*).
- Fix: Highlighting System now highlights only *MeshRenderer*, *SkinnedMeshRenderer* and *ClothRenderer* components, because you probably doesn't want to see highlighted meshes created by *ParticleRenderer*, *ParticleSystemRenderer*, *LineRenderer* and *TrailRenderer* components.
- Fix: Now you can use highlighting with Hardware Anti-Aliasing without having highlights flipped, but Hardware AA smooths only framebuffer – outline glow will remain aliased as it uses additional render buffers, so i recommend you to continue using *AntialiasingAsPostEffect.js* for this.
- Fix: *Camera Clear Flags = Don't Clear* doesn't cause flipping anymore.
- Fix: Non-standard camera normalized viewport rects now works correctly
- Fix: Highlighting now doesn't affect alpha channel of framebuffer

v1.0

- Initial release

# 2 Upgrade guide

## 2.1 Upgrade notes from v1.0 to v2.0

When upgrading Highlighting System in your projects, to not loose all *HighlightableObject* and *HighlightingEffect* components references, please do the following:

1. Remove *Highlighting.Init()* calls from your code – this is not needed anymore.
2. Remove *Highlighting.cs* script from the *Plugins\HighlightingSystem\Scripts* folder.
3. Remove everything from your *Plugins\HighlightingSystem\Resources* folder (don't worry - adapting each custom shader is not needed anymore).
4. Import upgraded package from the Unity Asset Store. In the *Importing package* window click on *All*, then *Import*.
5. Choose any highlighting preset on each *HighlightingEffect* component by clicking on its button, or setup highlighting intensity and quality parameters by yourself.
6. May be you'll need to tune up highlighting colors, because now Highlighting System displays actual highlighting colors given to the highlighting methods.

## 2.2 Upgrade notes from v2.0 to v3.0

1. Namespace *HighlightingSystem* has been added to avoid potential name conflicts with your own code. Add these directives to your scripts if they are referenced to any of the Highlighting System classes (you can find an example in *HighlightingSystemDemo\Scripts\Basic* folder):

   • for C# scripts: *using HighlightingSystem;*

   • for UnityScript (JavaScript) scripts: *import HighlightingSystem;*

   • for Boo scripts: *import HighlightingSystem*

2. *HighlightableObject* was renamed to *Highlighter*.

3. *HighlightingEffect* component has been separated to:

   • *HighlightingRenderer* + *HighlightingBlitter* (to be used mainly for desktop applications and in combination with other Image Effects, where precise control over the point at which highlighting buffer will be applied to the generated frame is required. See <u>Integration to your project</u> section for more info)

   • *HighlightingMobile* (optimized version for mobile devices)

4. Refer to the <u>highlighting occlusion</u> section of this document to consider completely removing manually added highlighting occluders from your project.

# 3 Overview

Highlighting System package allows you to easily integrate outline glow effect for objects highlighting in your Unity project. It allows you to make any object highlightable. It designed with optimization in mind, so any material operations is performed only when that's really needed. This system must work on all major platforms, where post-processing effects are supported.

This package requires Unity Pro version, as it uses post-processing.

## 3.1 Package overview

After package installation, inside of the *Plugins\HighlightingSystem* folder will be located all the scripts and replacement shaders required for the Highlighting System to work. There's also a bunch of Editor scripts can be found in *Plugins\Editor* folder, intended to simplify the workflow with the Highlighting System components.

Inside of the *HighlightingSystemDemo* folder in your project, you will find demonstration scene files. Use them as a reference when integrating Highlighting System to your project. You can remove this folder at any time.

# 4 Integration to your project

1. Import Highlighting System package from the Unity Asset Store to your project.
2. Choose one of the highlighting renderer components between *HighlightingRenderer* and *HighlightingMobile*:
   - *HighlightingRenderer* compoent always works only in combination with *HighlightingBlitter*. Position of the *HighlightingBlitter* component (among other Image Effects applied to the camera) determines the point at which highlighting buffer will be applied to the generated frame. *HighlightingBlitter* can even be applied to the camera other than the camera with *HighlightingRenderer* component.
   - *HighlightingMobile* - optimized version for mobile devices. Applies highlighting to the generated frame immediately after rendering.
3. Add chosen component to your camera above any other Image Effects (or the highlighting occlusion feature may not work!).
4. In case you are using *HighlightingMobile* – skip this step. Otherwise – add *HighlightingBlitter* component to the camera and assign corresponding *HighlightingRenderer* to it's 'Highlighting Renderer' slot.
5. Add *Highlighter* component to the objects you want to make highlightable or do that at runtime with *gameObject.AddComponent<Highlighter>()* method (see the *HighlightingSystemDemo\Scenes\Scripting* demo scene).
6. At runtime, call any highlighting methods on the *Highlighter* components.
7. Refer to the Dynamic Batching section of this document to properly setup *Depth Offset Factor* and *Depth Offset Units* properties of the highlighting renderer component, added in step 3. Optionally - set custom values for intensity and quality settings, or choose any existing preset by clicking on its button.

Remainders and notes on *HighlightingMobile*, *HighlightingRenderer* and *HighlightingBlitter* components will help you properly integrate and setup Highlighting System in your project.

# 5 Methods reference

Four different highlighting modes allowed (sorted in priority order):

1. **Once**
   Useful for highlighting objects under mouse cursor.

2. **Flashing**
   Can be used if you need to pay attention on some object (game tutorial item for example).

3. **Constantly**
   Can be used to turn on/off constant highlighting on object (for example, to highlight pickable items or selected objects).

4. **Occluder**
   Used only when hardware anti-aliasing is enabled. Object in this mode will become highlighting occluder. Actually, this is not a highlighting mode, but it will take effect only in case all other highlighting modes are inactive.

At runtime, use these methods of your *Highlighter* components to control the highlighting of an object:

- *ReinitMaterials()*
  Object renderers and materials reinitialization. Call this method before or after your highlightable object has changed (added and/or removed) his child objects or changed any of its materials and/or shaders (for example, when your game character has changed his weapon *GameObject* for example). Can be called multiple times per update - meshes reinitialization will occur only once.

- *OnParams(Color color)*
  Set color for one-frame highlighting mode.

- *On()*
  Turn on highlighting in this frame only.

- *On(Color color)*
  Turn on highlighting in this frame only with given color.

- *FlashingParams(Color color1, Color color2, float freq)*
  Set flashing parameters – colors and frequency.

- *FlashingOn()*
  Turn on flashing.

- *FlashingOn(Color color1, Color color2)*
  Turn on flashing from given color1 to color2.

- *FlashingOn(Color color1, Color color2, float freq)*
  Turn on flashing from given color1 to color2 and flashing frequency.

- *FlashingOn(float f)*
  Turn on flashing with given frequency.

- *FlashingOff()*
  Turn off flashing.

- *FlashingSwitch()*
  Switch flashing mode.

- *ConstantParams(Color color)*
  Set the constant highlighting color.

- *ConstantOn()*
  Fade in constant highlighting.

- *ConstantOn(Color color)*
  Fade in constant highlighting with given color.

- *ConstantOff()*
  Fade out constant highlighting.

- *ConstantSwitch()*
  Switch constant highlighting.

- *ConstantOnImmediate()*
  Turn on constant highlighting immediately (without fading in).

- *ConstantOnImmediate(Color color)*
  Turn on constant highlighting with given color immediately (without fading in).

- *ConstantOffImmediate()*
  Turn off constant highlighting immediately (without fading out).

- *ConstantSwitchImmediate()*
  Switch constant highlighting immediately (without fading in/out).

- *SeeThroughOn()*
  Turn on see-through mode. In this mode highlighting for this object will not be occluded by anything (will be always visible).

- *SeeThroughOff()*
  Turn off see-through mode

- *SeeThroughSwitch()*
  Switch see-through mode

- *OccluderOn()*
  Turn object into highlighting occluder. Note that highlighting occluders will be enabled only in case frame depth buffer is not available!

- *OccluderOff()*
  Disable occluder mode.

- *OccluderSwitch()*
  Switch occluder mode.

- *Off()*
  Turn off all highlighting modes.

- *Die()*
  Destroy the *Highlighter* component. Call this when you want to stop using highlighting on this object (for example, in case of highlightable enemy character has entered dying sequence).

# 6 Important usage tips

## 6.1 Common tips

- On mobile platforms, don't forget to set the *Use 32-bit Display Buffer* checkbox under the *Resolution and Presentation* section of the Unity's *Player Settings*.
- When configuring your *HighlightingRenderer* or *HighlightingMobile* components - increasing blur iterations will help you to improve outline glow quality, but try to keep this value as low as possible in terms of performance.
- When hardware anti-aliasing and highlighting occluders is used together – it will be better to keep the amount of highlighting occluders as low as possible, because in this case occluder objects rendered in additional pass as well as any other highlighted objects.
- Highlighting System uses one layer which shouldn't be used by anything else in the project. By default – this is layer number 7, which is normally inaccessible in Unity Editor and can be used only from scripts. This value can be changed in *Highlighter.cs* script, but in most cases this will never be required.

## 6.2 Using custom transparent shaders

In case you'd like to make your custom transparent shaders properly highlightable:

1. Make sure that *RenderType* shader tag is set to *TransparentCutout* or *Transparent* (check [this](#) for more info). Otherwise – such shader will be recognized by the Highlighting System as an opaque shader and alpha channel of your material's main texture will not be taken into account.
2. Make sure that your custom shader has the *_MainTex* property. Highlighting System will use texture applied to that property to detect transparent areas by comparing this texture alpha channel with the threshold value, taken from:
   - the *_Cutoff* property if your custom shader has it, or
   - the *Highlighter*'s internal *transparentCutoff* variable otherwise (set to 0.5 by default. You can change this value in *Highlighter.cs* script).

Note that the main texture with its offset and scale values is cached by the Highlighting System only on initialization, which will take place on first frame rendering and after each *ReinitMaterials()* call. Because of that, your changes to the main texture parameters will not be reflected by the highlighting without the *ReinitMaterials()* call.

## 6.3 Dynamic Batching

In case *Dynamic Batching* is disabled in *Player Settings* - always set *Depth Offset Factor* and *Depth Offset Units* parameters to 0 to avoid rendering artifacts!

Both main and internal highlighting camera supports *Dynamic Batching*, but in cases when main camera and highlighting camera batches different sets of objects – depth comparision imprecision can take place, what will lead to an artifact, called 'z-fighting':
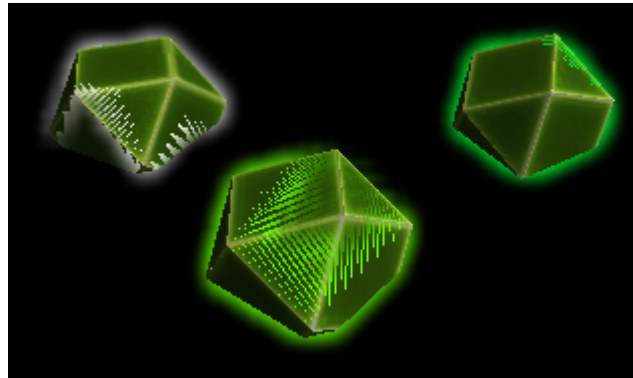


You have at least three options to eliminate this:

1. Force disable batching on such objects by duplicating and assigning different instances of the same material to them (recommended).

2. Disable *Dynamic Batching* in *Player Settings*. There's nothing bad in this option, due to a bunch of significant criterias which should be fulfilled for objects, before *Dynamic Batching* will be applicable on them, and the performance gain from this feature will not be significant in most of the cases. As an argument to use this option can also be the fact, that Unity's *camera.RenderWithReplacementShaders()* method (widely used by other Image Effects) doesn't supports *Dynamic Batching* at all. Don't forget to set *Depth Offset Factor* and *Depth Offset Units* parameters to 0 after disabling *Dynamic Batching*.

3. Offset depths of the highlighted objects when rendering them to the highlighting buffer by tuning *Depth Offset Factor* and *Depth Offset Units* parameters on the *HighlightingMobile* / *HighlightingRenderer* components. This will set *Offset Factor, Units* for all highlighting shaders simultaneously. This is the least recommended option because shader depth offset feature (glPolygonOffset, Depth Bias) is implementation-dependent (depends on hardware and graphics API used), so the same settings will not work evenly on all devices. Also, using depth offset leads to another visual artifact, visible when highlighted object intersects another object:

However, if you don't have too contrasty intersecting objects, which should be highlighted, and you have the ability to test and tune *Factor* and *Units* parameters for most of your target platforms – this can turn into a good option for you.

## 6.4 Highlighting occlusion

In case frame depth buffer is available – it will be used to occlude highlighting for objects with disabled see-through mode. Otherwise, (mainly when <u>hardware anti-aliasing</u> is turned on) highlighting occluders (*GameObjects* with *Highlighter* components for which *OccluderOn()* method was called) will be enabled to replace highlighting occlusion functionality (Highlighting System will log about that change to *Console*).

So if you don't use hardware anti-aliasing in your project – you shouldn't manually add highlighting occluders to any of your scenes (never use any of the *OccluderOn()*, *OccluderOff()* and *OccluderSwitch()* methods).

But if in doubt – just add them to make sure that your project will work identically even when hardware anti-aliasing will be enabled.

## 6.5 Anti-aliasing

Hardware anti-aliasing (or MSAA, MultiSample Anti-Aliasing) is enabled in Unity if *Anti Aliasing* property is set to something different than *Disabled* in *Edit > Project Settings > Quality* settings. Note that there are multiple quality levels all with their own anti-aliasing settings.

Hardware anti-aliasing has two significant drawbacks:

- it is not compatible with [Deferred Lighting rendering path](#).

- When it is turned on – depth buffer is rendered directly to the backbuffer and is not accessible as *depthBuffer* part of a source *RenderTexture* when *OnRenderImage()* is called. For this reason – it is not compatible with [highlighting occlusion](#) feature.

Because of these limitations – it is recommended not to use hardware anti-aliasing and replace it with *AntialiasingAsPostEffect.js* instead from the Unity's standard *Image Effects (Pro only)* package when achieving anti-aliased image is required.

# 7 Known issues

v3.0:

When switching hardware anti-aliasing modes or rendering paths with enabled hardware anti-aliasing in Unity Editor on Mac OS X – several issues may appear:

- Highlighting System may stop working
- highlighting buffer or one of the previously redered frames may stuck on screen
- Unity Editor may crash

It seems like this happens because Unity Editor does not always really switches anti-aliasing internally for the framebuffer and *RenderTexture*s. To force-update this – you can maximize/minimize *Game* window.

Such issues have never been spotted on builded projects – only in Unity Editor on Mac OS X.

# 8 Support

Send your bug reports, feedback, suggestions, questions, feature requests to:
[support@deepdreamgames.com](mailto:support@deepdreamgames.com)

**Unity, Unity Pro, Unity Asset Store are Copyright © 2014 Unity Technologies.
Highlighting System are Copyright © 2014 Deep Dream Games.**