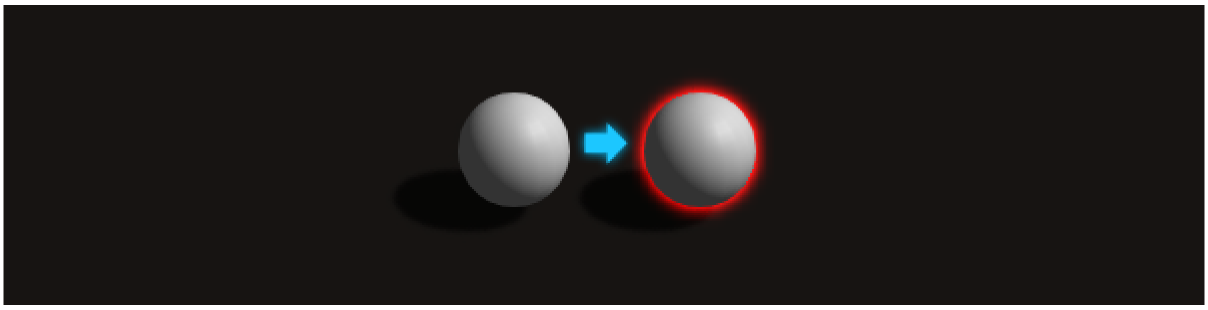


Highlighting System 4.0

User Guide

Table of Contents

1 Changelog

v4.0

- Unity 5 compatibility
- Windows Phone 8 compatibility
- Highlighters now rendered via *CommandBuffer*s. Simplified setup and usage – only *HighlightingRenderer* component is required on Camera.
- Highlighters now culled on the CPU before rendering
- Ability to save custom highlighting Presets in editor (Presets shared between projects)
- Added support for ParticleRenderer (Legacy) and ParticleSystemRenderer highlighting
- Other improvements and performance optimizations

v3.0.1

- Fixed possible screen darkening when *Color Space* is set to *Linear* in *Player Settings*
- Improved documentation

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 during highlighting buffer rendering (speeds up rendering. Stencil buffer access was introduced in Unity 4.2)
- Fixed lightmapped objects highlighting
- *_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. Uncomment *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*) were 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*) were replaced with *Highlighter* components management
- Added *HighlightingSystem* namespace (to avoid potential name conflicts)

v2.0

- Linear blending of the highlighting and frame buffers (gives correct highlighting colors)
- All shaders are 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 don'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 work correctly

- Fix: Highlighting doesn't affect alpha channel of framebuffer now

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 lose 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 – you don't have to adapt your custom shaders anymore).
4. Import upgraded package from the Unity Asset Store. In the *Importing package* window click on *All*, then *Import*.
5. Choose one of the highlighting Preset on each *HighlightingEffect* component by clicking on its button, or setup highlighting intensity and quality parameters by hand.
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.x

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 split into two versions:
   - *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 Integration to your project section for more info)
   - *HighlightingMobile* (optimized version for mobile devices)
4. Refer to the highlighting occlusion section of this document to consider completely removing manually added highlighting occluders from your project.

2.3 Upgrade notes from v3.0.x to v4.0

1. *HighlightingMobile* and *HighlightingBlitter* components have been removed. Use

only *HighlightingRenderer* instead. It is no longer necessary to keep *HighlightingRenderer* as a first Image Effect on Camera - order of this component (among other Image Effects) defines the point at which highlighting will be applied to the screen (this replaces removed *HighlightingBlitter* component functionality).

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 necessary. This system works on all major platforms, where post-processing effects are supported.

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 you'll be integrating Highlighting System to your project. Feel free to 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. Add *HighlightingRenderer* component to the Camera. Order of this component (among other Image Effects applied to the Camera) defines the point at which highlighting buffer will be applied to the rendered frame.
3. 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).
4. At runtime, call any highlighting methods on the *Highlighter* components.
5. Refer to the Dynamic Batching section of this documentation to properly setup *Depth Offset Factor* and *Depth Offset Units* properties of the highlighting renderer component. Optionally - set custom values for intensity and quality settings, or select any Preset from the Preset drop-down list.

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 - 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.

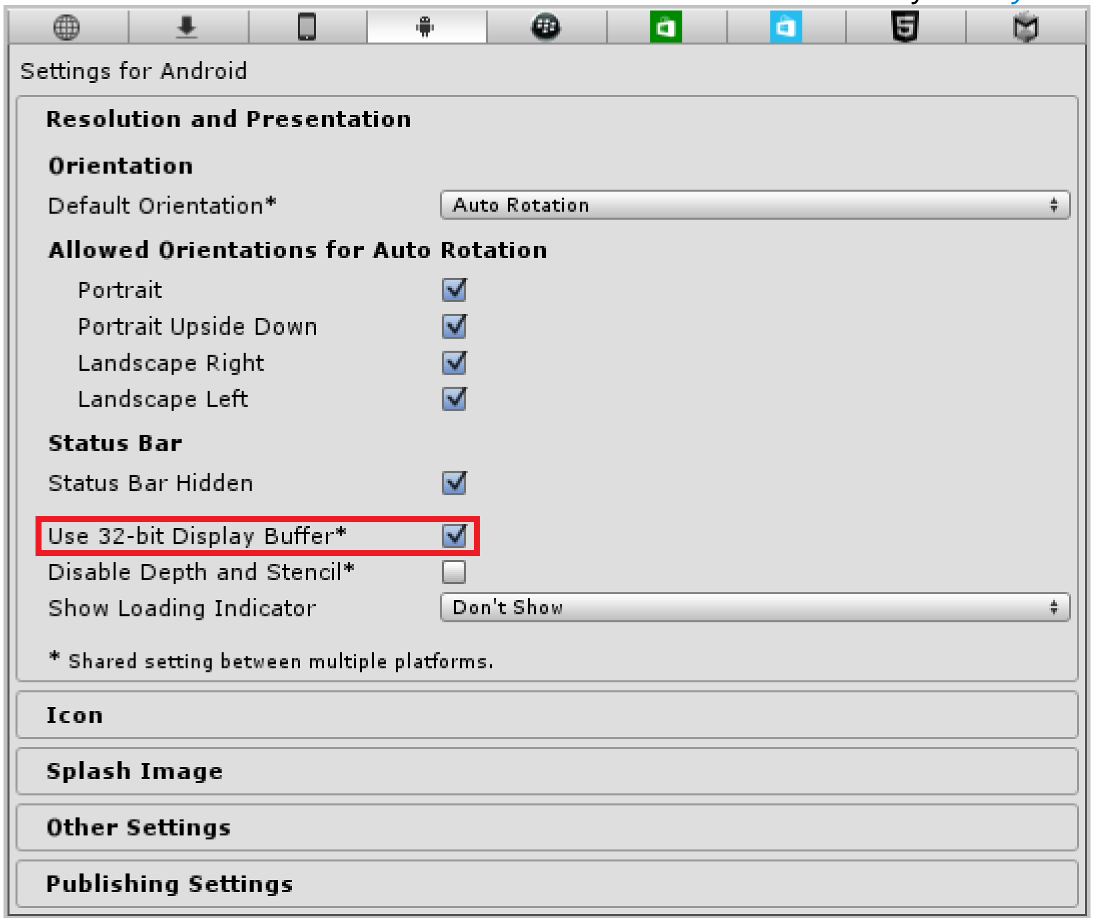
- *Off()*
  Turn off all highlighting modes.

- *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*.



Settings for Android

Resolution and Presentation

Orientation

Default Orientation*	Auto Rotation

Allowed Orientations for Auto Rotation

Portrait	✓
Portrait Upside Down	✓
Landscape Right	✓
Landscape Left	✓

Status Bar

Status Bar Hidden	✓
Use 32-bit Display Buffer*	✓
Disable Depth and Stencil*	☐
Show Loading Indicator	Don't Show

* Shared setting between multiple platforms.

Icon

Splash Image

Other Settings

Publishing Settings

- When configuring your *HighlightingRenderer* component - 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 are used together – it will be better to keep the amount of highlighting occluders as low as possible, because in this case occluders rendered in additional pass as well as any other highlighted objects.

6.2 Using custom transparent shaders

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

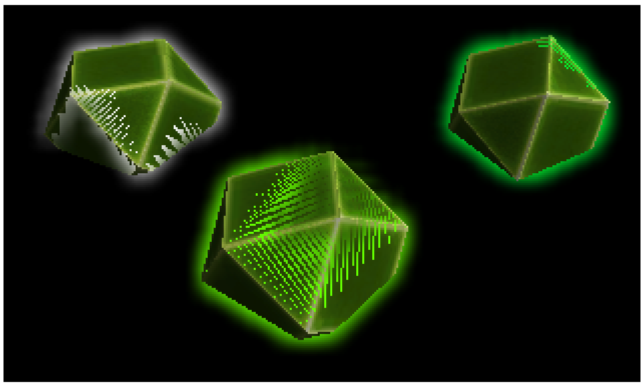
1. Make sure that *RenderType* shader tag is set to *TransparentCutout* or *Transparent* (check this for more info). Otherwise – such shader will be recognized by the Highlighting System as opaque shader and alpha channel of your material's main texture will not be taken into account.
2. Make sure that your custom shader has the *_MainTex* property. Highlighting System will use texture assigned 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!

Highlighting System supports *Dynamic Batching*, but in some cases when different sets of rendered objects being batched – depth comparision imprecision can take place, what will lead to an artifact, called 'z-fighting':



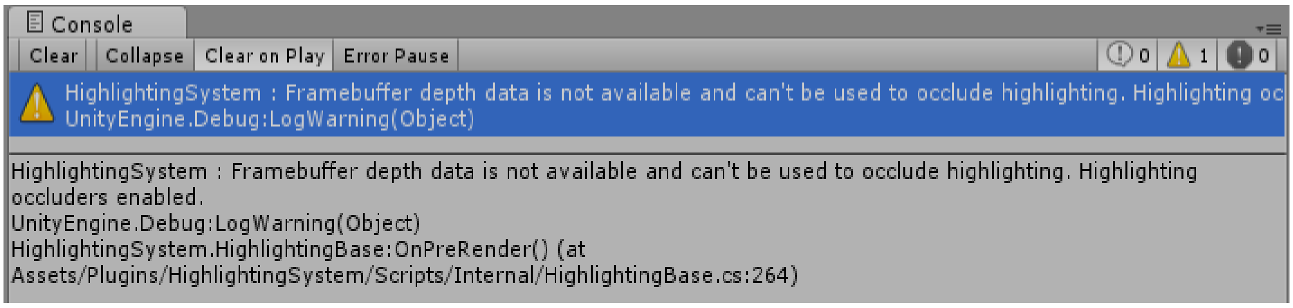
You have at least three options to eliminate this:

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. Tune *Depth Offset Factor* and *Depth Offset Units* parameters on the *HighlightingRenderer* component to offset depths of the highlighted objects when rendering them to the highlighting buffer. 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 equally on all devices. Also, using depth offset leads to another visual artifact, that you may notice when highlighted object intersects another object:

However, if your intersecting highlighted objects doesn't differ much by contrast, and you have the ability to test and tune *Depth Offset Factor* and *Depth Offset Units* parameters for most of your target platforms – this can turn into a good option.

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, (mostly when hardware anti-aliasing 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 this event to the *Console*):



```
HighlightingSystem : Framebuffer depth data is not available and can't be used to occlude highlighting. Highlighting oc
UnityEngine.Debug:LogWarning(Object)
```

```
HighlightingSystem : Framebuffer depth data is not available and can't be used to occlude highlighting. Highlighting
occluders enabled.
UnityEngine.Debug:LogWarning(Object)
HighlightingSystem.HighlightingBase:OnPreRender() (at
Assets/Plugins/HighlightingSystem/Scripts/Internal/HighlightingBase.cs:264)
```

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 not set to *Disabled* in *Edit > Project Settings > Quality* settings. Note that there are multiple quality levels with their own anti-aliasing settings.

Hardware anti-aliasing has two significant drawbacks:

- It is not compatible with Legacy Deferred Lighting and Deferred Shading rendering paths
- When it is turned on – depth buffer rendered directly to the backbuffer and is not accessible for Image Effects. That said – 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 *Antialiasing* Image Effect instead from the Unity's *Standard Assets* package when you need to have anti-aliased image as a result.

7 Known issues

v3.0.x:

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 rendered frames may stuck on screen
- Unity Editor may crash

It seems like this happens because Unity Editor does not 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 built projects – only in Unity Editor on Mac OS X.

8 Support

Send your bug reports, feedback, suggestions, questions, feature requests to:
support@deepdreamgames.com