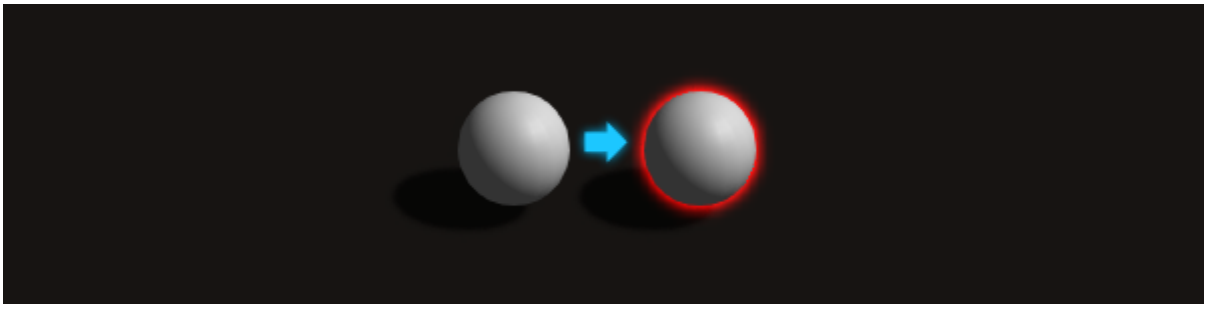


Highlighting System 4.3

User Guide

Table of Contents

1 Changelog....3
2 Upgrade notes....8
2.1 Upgrading from v1.0 to v2.0....8
2.2 Upgrading from v2.0 to v3.0.x....8
2.3 Upgrading from v3.0.x to v4.x....9
2.4 Upgrading from v4.x to v4.3....9
3 Overview....10
3.1 Package overview....10
4 Integration to your project....11
5 API....12
5.1 Highlighter API....12
5.2 HighlightingRenderer API....15
5.2.1 HighlightingRenderer Presets API....17
6 Important usage tips....19
6.1 Common tips....19
6.2 Using custom transparent shaders....20
6.5 Anti-aliasing....21
7 Limitations....23
8 Known issues....25
8 Support....26

1 Changelog

v4.3

- Virtual Reality *Single Pass (Fast)* Stereo Rendering Method is now supported
- Implemented support for highlighting geometry rendered with GPU instancing
- Static and Dynamic batching will no longer produce z-fighting artifacts under any circumstances. *Dynamic Offset* and *Factor* options have been removed from the *HighlightingRenderer* component
- Highlighting depth occlusion now works even if MSAA is enabled. This is no longer necessary to manually add *HighlighterOccluder* components to *GameObjects* (use *HighlighterOccluder* component only to achieve see-thorugh occluders). Non-see-through occluders is now never rendered, so that saves performance
- Added API for runtime control of highlighting presets. Now they are stored in *HighlightingRenderer* components (previously they were stored using Unity *EditorPrefs*)
- Exposed *Blur Directions* settings for *HighlightingRenderer* component (options: *Diagonal, Straight, All*) to allow finer control of solid highlighting modes
- Exposed *Anti Aliasing* settings for *HighlightingRenderer* component (options: *Use Value From Quality Settings*, *Disabled*, *2x Multi Sampling*, *4x Multi Sampling*, *8x Multi Sampling*). That will define the state of anti-aliasing for the highlighting buffer
- Added *forceRender* option to the *Highlighter* (to make it ignore frustum culling and occlusion culling)
- Improved cross-platform compatibility
- Fixed highlighting depth occlusion not working if *Camera Clear Flags* set to *Depth only* or *Don't clear* in *Forward* or *VertexLit* rendering paths
- Fixed no longer used highlighting materials kept in memory (turns out Unity is never releasing unreferenced materials without the *Resources.UnloadUnusedAssets()* call, so explicit *Destroy()* call is required) (https://trello.com/c/st8b6YZ9)
- Fixed 'gray tint instead of highlighting' bug (https://trello.com/c/58fK9qqd)
- Fixed *HighlightierRenderer.EndOfFrame* generating garbage every frame (https://trello.com/c/4lbbfKee)
- Fixed "WARNING: Shader Unsupported: 'Hidden/Highlighted/Blur' - Pass '' has no fragment shader" or even “EXC_BAD_ACCESS” exception when running on iOS Metal device (https://trello.com/c/yiDskT1i)
- Improved documentation

v4.2.1

- Fixed null reference exceptions if *Highlighter* has disabled *GameObject*s in hierarchy

v4.2

- Support for multiple cameras and different camera clear flags
- Support for *LODGroup* component
- Improved compatibility across all platforms

- Proper culling of invisible highlighting renderers
- Implemented optional *HighlightingBlitter* for blitting highlighting results to the screen using different camera
- Improved and refactored highlighting system core
- Implemented ability to dynamically change constant on/off transition times (no more common constantOnSpeed/constantOffSpeed static properties - use *ConstantOn(float time)* and *ConstantOn(Color color, float time)* instead)
- Disabling *GameObjects* with *Highlighter* components or *Highlighter* components directly in Unity Editor properly affects highlighting (toggling them from scripts at runtime works well as before)
- Fixed null reference exceptions after re-enabling *HighlightingRenderer* component
- Implemented *HighlighterBlocker* to prevent highlighting of specific objects in hierarchy
- Optimized memory allocation in *HighlighterInternal.GrabRenderers()* method
- Exposed *Highlighter.seeThrough* property. *SeeThrough(bool state)*, *SeeThroughOn()*, *SeeThroughOff()*, *SeeThroughSwitch()* methods deprecated and will be removed in the next version
- Exposed *Highlighter.occluder* property. *OccluderOn()*, *OccluderOff()*, *OccluderSwitch()* methods deprecated and will be removed in the next version
- Compatibility with iOS Metal graphics API
- Fixed iOS build crash when stripping engine code is enabled in Unity 5.2.0+
- Windows Store Apps: fixed compilation error caused by inability to access C# classes from Boo and JS scripts on WSA platforms (without checking .NET Core Partially in Compilation Overrides in *PlayerSettings*)
- Reworked all demo scripts and scenes
- Highlighter item revealer example added
- Added new demo scenes: LODGroup, Revealer, RenderTexture, ViewportRect, CustomBlitter, ClearFlags, FPSCamera1, FPSCamera2, Toggle, VR
- Changing properties of *HighlighterInteractive*, *HighlighterConstant*, *HighlighterFlashing* and other highlighter components in Inspector affects highlighting in realtime
- Other fixes and improvements

v4.1

- Implemented see-through mode for highlighting occluders (occluder in this mode won't receive any highlighting on it's area)
- *SpriteRenderer*s highlighting is now rendered from both sides. Makes highlighting visible if used on sprites with negative scale (mirrored sprites)
- Fixed producing a lot of *RenderTexture* garbage (previously causing out of memory issues on mobile devices)
- Fixed highlighting materials not being destroyed on loading scenes
- Fixed disabled / invisible renderers uninitialization
- Fixed improper highlighting offset on Android and iOS devices for solid highlighting

modes (when *HighlightingRenderer Downsampling* property set to *None*)
- Fixed improper highlighting when emulating OpenGL ES device (Android or iOS) in Unity Editor on Windows
- Fixed improper highlighting on Xbox
- Fixed artifacts when used with bloom image effect
- Other minor fixes and improvements

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 notes

2.1 Upgrading 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 Upgrading 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 Upgrading from v3.0.x to v4.x

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).

2.4 Upgrading from v4.x to v4.3

1. Highlighting presets now stored locally, in each instance of the *HighlightingRenderer* component using Unity's native serialization system (previously they were stored globally using *EditorPrefs*, under the "HighlightingSystem.Presets" key). See HighlightingRenderer Presets API for more info. Presets made in previous versions of the Highlighting System will be lost after upgrading, so make sure to remember their parameters in order to recreate them after upgrading.

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 and works on all major platforms, where Image Effects is supported.

3.1 Package overview

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

Inside of the *HighlightingSystemDemo* folder, you will find example scenes and scripts intended to demonstrate how to integrate and use Highlighting System in your own projects. Feel free to completely 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 on this Camera) defines the point at which highlighting buffer will be applied to the rendered frame.
3. To be able to access Highlighting System API from your own scripts – add '*using HighlightingSystem;*' directive to the beginning of your script.
4. Add *Highlighter* component to the objects you want to make highlightable or do so at runtime using *gameObject.AddComponent<Highlighter>()* call (see the *HighlightingSystemDemo\Scenes\06 Scripting* demo scene for an example).
5. At runtime, use Highlighter API to control the state of highlighting on a specific object.
6. Tweak settings on the *HighlightingRenderer* component to change the look of highlighting. Please note that highlighting presets stored in the component itself, so you can manipulate them at runtime using HighlightingRenderer Presets API.

5 API

5.1 Highlighter API

Four different highlighting modes available (listed in priority order):

1. **Once**
   Highlights object only for a single frame, so you can call it every frame for the object under the mouse cursor.

2. **Flashing**
   Useful to pay attention on a specific object (game tutorial item for example).

3. **Constant**
   Used to turn on/off constant highlighting on an object (for example, to highlight all pickable items on screen).

4. **Occluder**
   Object in this mode will become highlighting occluder. Actually, this is not the highlighting mode, but it will take effect only in case all other highlighting modes disabled and only if *seeThrough* flag is set (see below).

In case multiple highlighting modes enabled on the highlighter - mode with higher priority will take effect.

Use following methods and properties of the *Highlighter* components to control highlighting on a specific object:

- *bool seeThrough*
  See-through mode for highlighters or occluders. When set to true - highlighter in this mode will not be occluded by anything (except for see-through occluders). Occluder in this mode will overlap any highlighting.

- *bool occluder*
  Occluder mode. Use only in combination with *seeThrough* to turn object into see-thorugh occluder (the one which always occludes any highlighting over it's shape, see the '05 OccluderModes' demo scene for an example). Has no effect if *seeThrough* is disabled.

- *bool forceRender*
  Enables force-rendering mode. When rendering highlighting for this highlighter instance - no frustum culling or occlusion culling will be performed for it's renderers (though frustum clipping still takes place, since near and far frustum plane define depth buffer range in world space) and renderers from all LOD levels will be always

rendered for all cameras (only for the highlighting – that doesn't affect regular object rendering in any way). Please be considerate in enabling this mode, or you may experience performance degradation.

- *void ReinitMaterials()*
  Reinitialize *GameObject* renderers and materials. Call this method before or after your highlightable object has changed (added and/or removed) it's child objects or any materials and/or shaders (for example, when your game character has switched it's weapon). Feel free to call this method multiple times in a single update - reinitialization will occur only once at the rendering stage.

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

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

- *void On(Color color)*
  Turn on highlighting only in current frame using specified color.

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

- *void FlashingOn()*
  Turn on flashing.

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

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

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

- *void FlashingOff()*
  Turn off flashing.

- *void FlashingSwitch()*
  Switch flashing mode.

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

- *void ConstantOn(float time)*
  Fade in constant highlighting using specified transition duration.

- *void ConstantOn(Color color, float time)*
  Fade in constant highlighting using specified color and transition duration.

- *void ConstantOff(float time)*
  Fade out constant highlighting using specified transition duration.

- *void ConstantSwitch(float time)*
  Switch constant highlighting using specified transition duration.

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

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

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

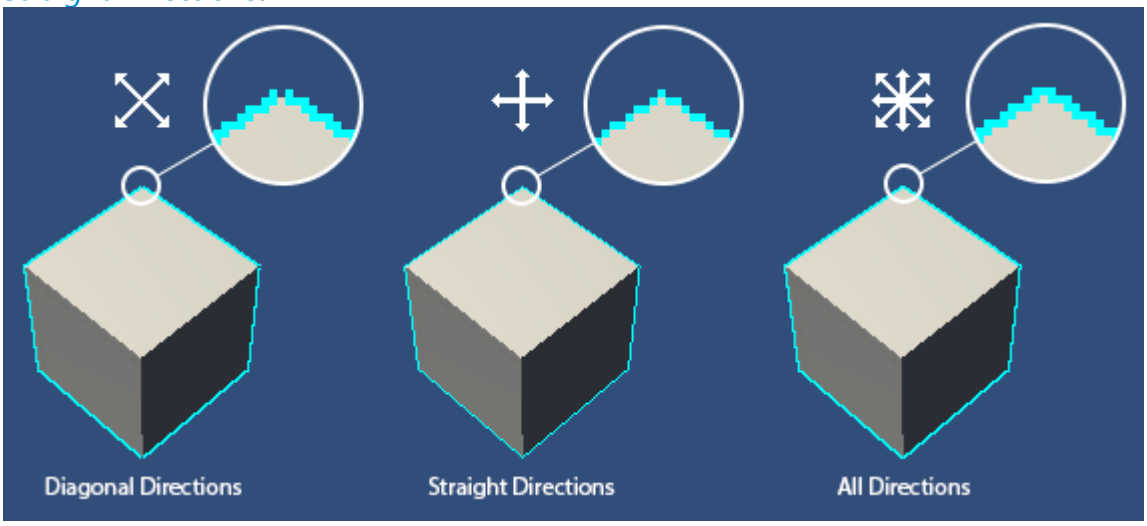
- *void ConstantSwitchImmediate()*
  Switch constant highlighting immediately (without fade in/out).

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

- *void Die()*
  Destroy the *Highlighter* component. Call this when you've done using highlighting on this object (for example, when character entered dying sequence).

5.2 HighlightingRenderer API

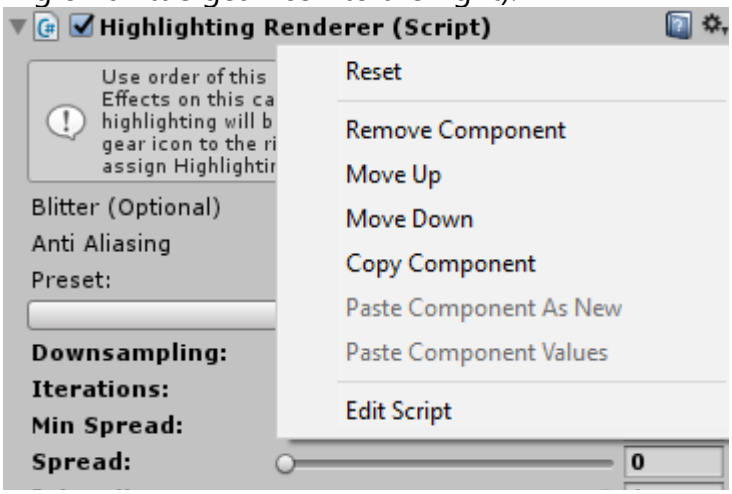
- *bool isSupported*
  Returns *true* if Highlighting System is supported on the current platform. Internally this checks for *SystemInfo.supportsImageEffects*, *SystemInfo.SupportsRenderTextureFormat(RenderTextureFormat.ARGB32)* and *isSupported* value for all highlighting shaders.
- *AntiAliasing antiAliasing*
  Anti-aliasing value for *highlightingBuffer* (options: *QualitySettings*, *Disabled*, *MSAA2x*, *MSAA4x*, *MSAA8x*). Set to *AntiAliasing.QualitySettings* by default (will use value from *QualitySettings.antiAliasing* in that case).
- *HighlightingBlitter blitter*
  Get or set *HighlightingBlitter* instance, which will be used to blit highlighting rendering results. Set to *null* to make *HighlighterRenderer* blit during the *OnRenderImage(RenderTexture src, RenderTexture dst)* callback (default behaviour for most Image Effects in Unity).
- *void Blit(RenderTexture src, RenderTexture dst)*
  Compose *highlightingBuffer* with *src RenderTexture* and output result to the *dst RenderTexture*. To be used only in custom scripts derived from *HighlightingRenderer* component to explicitly control highlighting blit. Make sure to also override *OnRenderImage* method. Please note that *highlightingBuffer* will be updated for the current frame only during the *BeforeImageEffectsOpaque* camera event, so calling this method earlier will probably lead to undesired results.
- *BlurDirections blurDirections*
  Defines directions in which highlighting buffer will be shifted/blurred (options: *Diagonal*, *Straight*, *All*. Default is *Diagonal*). This option allows finer control of solid highlighting modes. *All Directions* is more expensive than *Diagonal Directions* or *Straight Directions*:



- *float blurIntensity*
  Highlighting intensity. Internally defines the value by which highlighting buffer alpha channel will be multiplied after each blur iteration.
- *float blurMinSpread*
  Blur Min Spread. Lower values give better looking blur, but require more iterations to get large blurs. Pixel offset for each blur iteration is calculated as *blurMinSpread + blurSpread * Iteration Index*. Usually, the sum of *blurMinSpread* and *blurSpread* lies between 0.5 and 1.0.
- *float blurSpread*
  Blur Spread. Lower values give better looking blur, but require more iterations to get large blurs. Pixel offset for each blur iteration is calculated as *blurMinSpread + blurSpread * Iteration Index*. Usually, the sum of *blurMinSpread* and *blurSpread* lies between 0.5 and 1.0.
- *int downsampleFactor*
  Highlighting buffer downsampling factor. Allowed values are *1* (No downsampling), *2* (Half), *4* (Quarter). Internally defines the size of highlighting buffer by dividing frame buffer (screen) size with this value.
- *int iterations*
  Blur iterations. Number of blur iterations to be performed on the highlighting buffer. Larger number means more blur.

5.2.1 HighlightingRenderer Presets API

Highlighting presets stored locally, in each instance of the *HighlightingRenderer* component using Unity native serialization system. That means they are saved along with prefabs and/or scenes. Applying *HighlightingPreset* affects the following settings of the *HighlightingRenderer*: *downsampleFactor*, *iterations*, *blurMinSpread*, *blurSpread*, *blurIntensity*, *blurDirections*. To reset to default, copy and paste presets between instances of the *HighlightingRenderer* components or between projects – please use *Reset*, *Copy Component* and *Paste Component Values* context menu options correspondingly (you can access them by clicking on a little gear icon to the right):



Please use the following API to access and manipulate *HighlightingRenderer* presets at runtime:

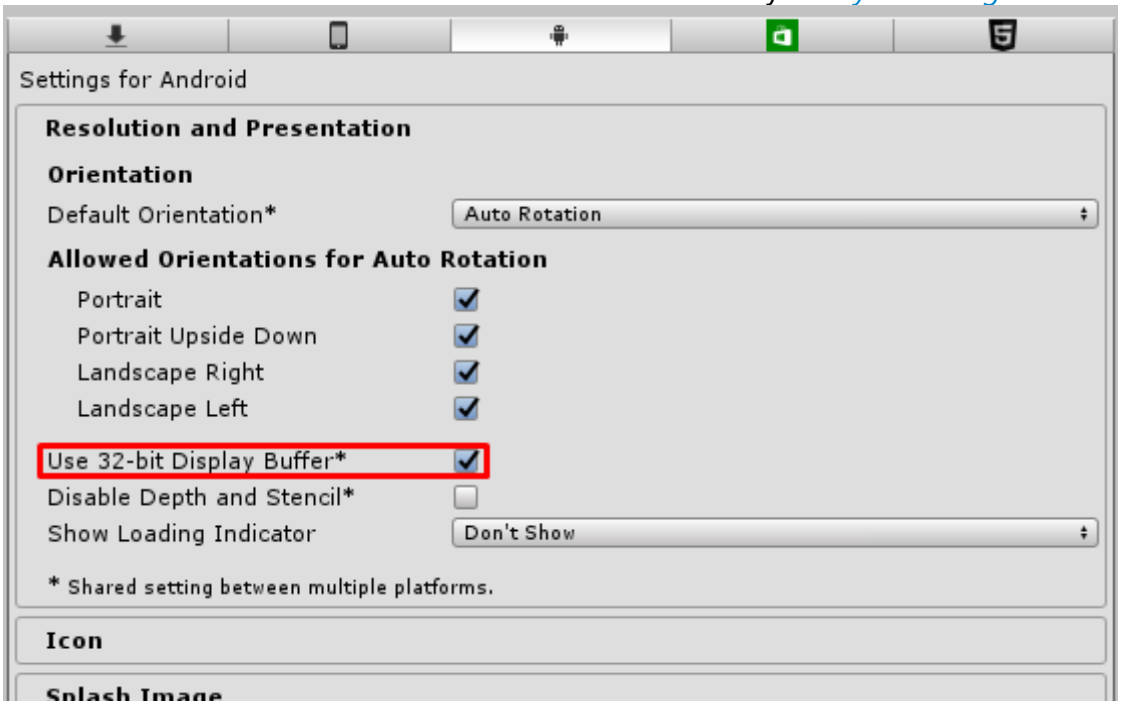
- *ReadOnlyCollection<HighlightingPreset> presets*
  Returns stored presets as *ReadOnlyCollection<HighlightingPreset>* (collection class is defined in *System.Collections.ObjectModel* namespace)
- *bool GetPreset(string name, out HighlightingPreset preset)*
  Get stored preset by name. Returns true if preset with this name has been found in the list of stored presets.
- *bool AddPreset(HighlightingPreset preset, bool overwrite)*
  Add (store) preset. Returns false if preset with this name already exists and overwrite flag is not set. Returns true otherwise.
- *bool RemovePreset(string name)*
  Find stored preset by name and remove it. Returns true if preset with this name has been found and removed. Returns false otherwise.
- *bool LoadPreset(string name)*
  Find stored preset by name and apply it's settings.
- *void ApplyPreset(HighlightingPreset preset)*
  Apply specified preset settings.

- *void ClearPresets()*
  Clear all stored presets.

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* component - increasing blur iterations will help you to improve outline glow quality, but try to keep this value as low as possible for better performance.
- Any renderer component derived from Unity built-in *Renderer* class can be highlighted. By default - highlighting of the following renderers is enabled: *MeshRenderer*, *SkinnedMeshRenderer*, *SpriteRenderer*, *ParticleSystemRenderer*. Feel free to tune *types* list in the *Highlighter.cs* script as you need.

6.2 Using custom transparent shaders

In order to make 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 interpreted 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 *_MainTex* property of type 2D (Texture). Highlighting System will use texture assigned to this property to detect transparent areas by comparing texture alpha channel with threshold value, taken from:
   - *_Cutoff* (Float) property if your custom shader has it, or
   - *Highlighter*'s internal *transparentCutoff* variable otherwise (set to 0.5 by default. You can change this value in the *HighlighterRenderer.cs* script).

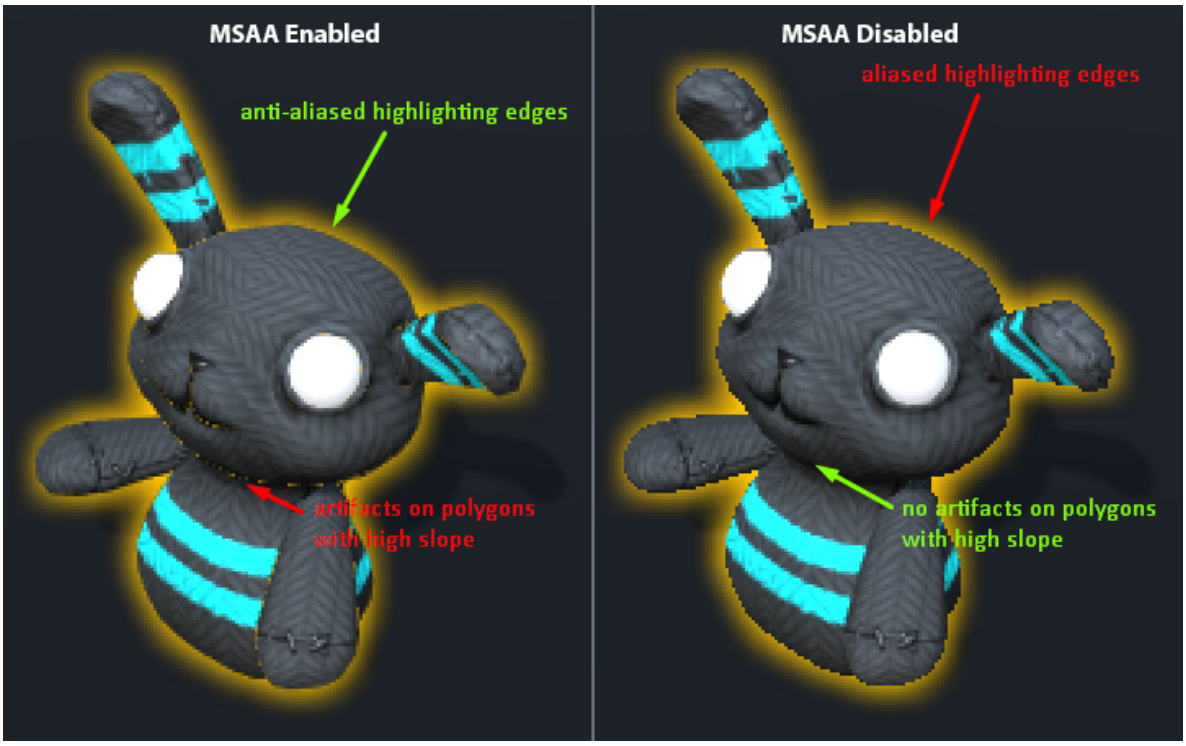
Note that the main texture with its offset and scale values is cached by the Highlighting System only on highlighter's initialization, which takes place after instantiating *Highlighter* component and after each call to *ReinitMaterials()*. Because of that, your changes to the main texture properties will not be reflected by the highlighting without the call to *ReinitMaterials()* method.
Also, please note that if your shader handles *_Cutoff* property differently (not with the default alpha clip *clip(alpha - _Cutoff)* expression) – the resulting higlighting may differ from what's rendered by your custom shader.

6.5 Anti-aliasing

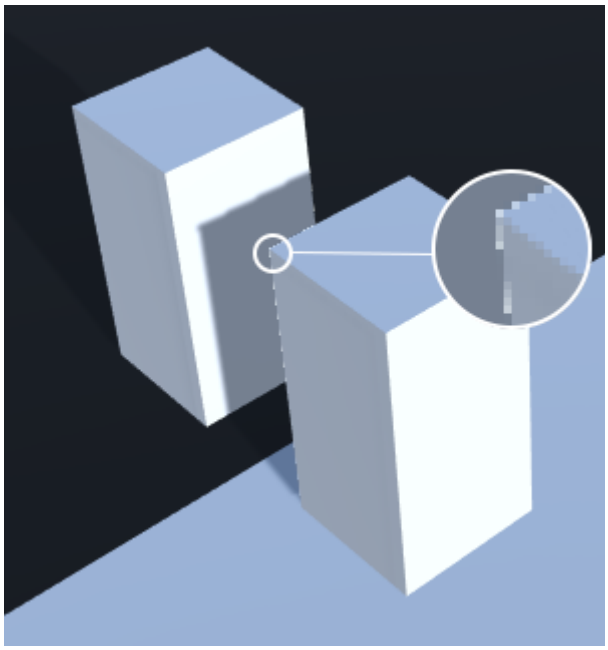
Hardware anti-aliasing (or MSAA, Multi-Sample Anti-Aliasing) is enabled in Unity if *Anti Aliasing* property is not set to *Disabled* in *Edit > Project Settings > Quality* settings. Note that there are multiple quality levels all with their own anti-aliasing settings.

Hardware anti-aliasing has several significant drawbacks:

- It is not compatible with Legacy Deferred Lighting and Deferred Shading rendering paths
- It is not compatible with HDR rendering
- There is no way in Unity to access and use non-MSAA-resolved *_CameraDepthTexture* in Image Effects. So if you enable anti-aliasing for the highlighting buffer - imprecisions between anti-aliased color buffer and non-anti-aliased depth texture will produce rendering artifacts (as seen on the left side of this image):



Same issue affects shadows rendering in Unity, so it seems there is currently no way to fix that:

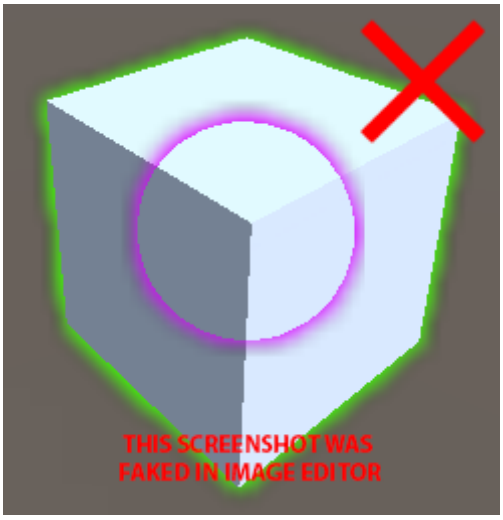


Due to all of the above – it is not recommended to use hardware anti-aliasing in your project. You can replace it with *Antialiasing* Image Effect from the Unity Standard Assets package.

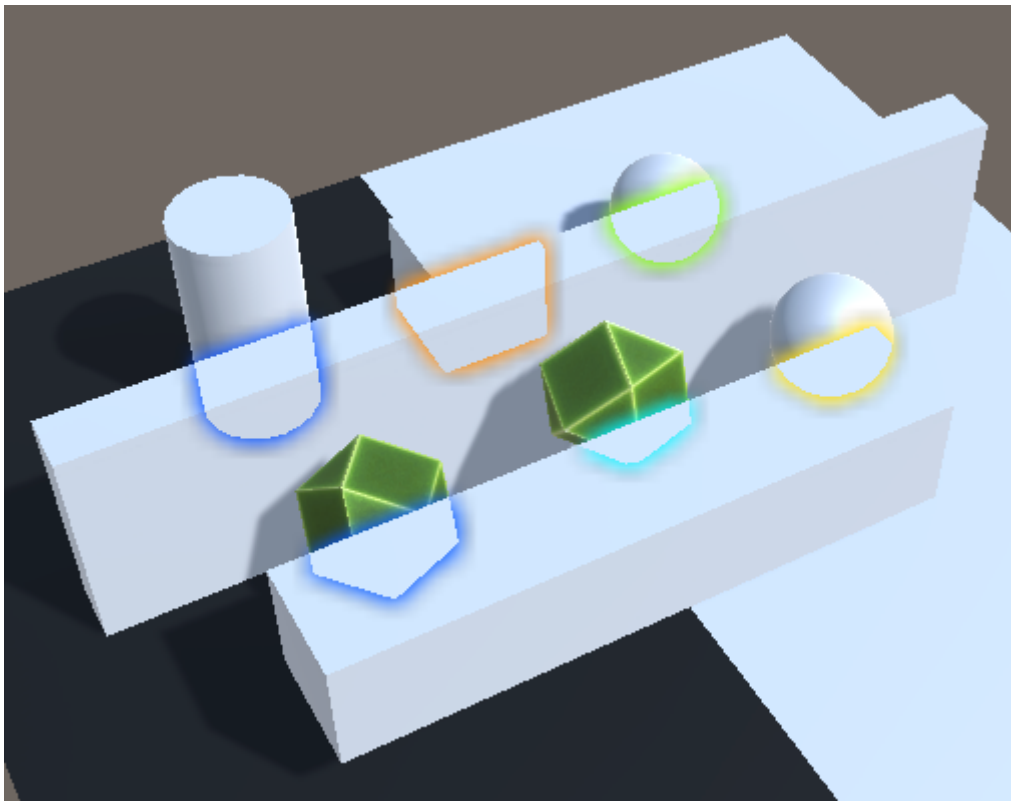
7 Limitations

Due to the Image Effect nature of the Highlighting System – it has several limitations:

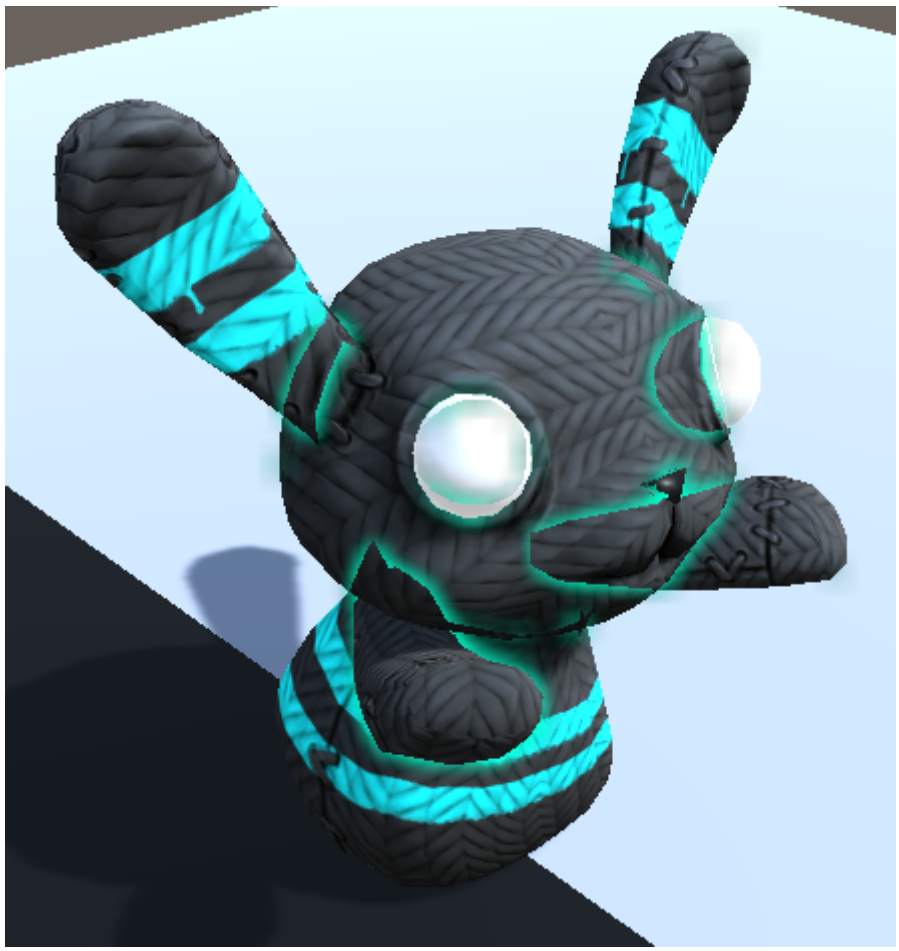
1. Multi-layer highlighting. This isn't possible to show highlighting of an object which is obscured by other highlighted object.



2. Inverse highlighting occlusion (highlight only parts obscured by other objects).



Despite the fact that this can be implemented for simple convex geometry such as this shown on the image above – complex arbitrary meshes will occlude their own parts, so they will become highlighted:



3. Mixing *HighlightingRenderer* settings.

This isn't possible to use different highlighting settings on a per-object basis. *HighlightingRenderer* settings always apply to the whole image only.

8 Known issues

1. For VR, it is not possible to render highlighting with one Camera and blit (using *HighlightingBlitter*) with another Camera. Highlighting buffer rendered for the left eye is overwritten with right eye highlighting rendering results. Currently (in Unity 5.5.0f3) no API is exposed to properly support *MultiPass*, *SinglePass* and *Instancing* stereo rendering paths in Image Effects. Specifically - all non-temporary renderTextures should be (in accordance to this document):
    - *MultiPass* - duplicated (one *RenderTexture* for each eye)
    - *SinglePass* - turned into double-wide *RenderTexture*s
    - *Instancing* - turned into RenderTexture array (*RenderTexture.dimension* = *TextureDimension.Tex2DArray*)

    But there is no way to check at runtime which stereo rendering path is currently active, and even the *StereoRenderingPath enum* is only in *UnityEditor* namespace, so it cannot be used outside of the Editor.
2. Unity Editor may hang or crash on high-resolution displays (such as Retina® displays used in MacBook®) when hardware anti-aliasing is enabled. Hardware anti-aliasing option (enabled by default for new Unity projects) basically acts as a multiplier for your game view resolution, so in case you're experiencing this issue – you are probably running out of video memory and/or exceeding maximum allowed *RenderTexture* size. To fix this - either disable hardware anti-aliasing in Quality Settings or reduce your game view screen resolution using Aspect Drop-down.
3. Highlighting doesn't work properly on iOS platform if hardware anti-aliasing is enabled. There is a very low chance that someone needs this considering the device screen DPI and performance drop that comes from enabling this option on a mobile device.

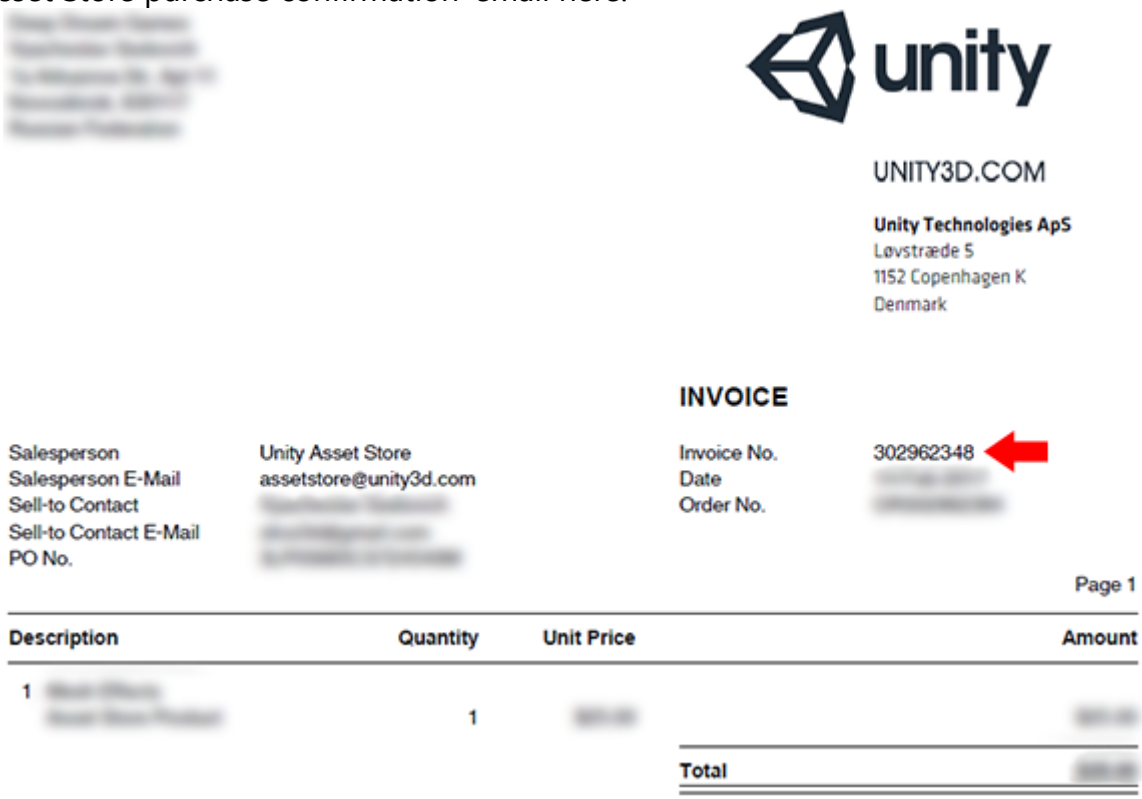
8 Support

Here you can find Highlighting System development board which you can use to check if a specific bug fix or a feature is already known or in development:
https://trello.com/b/GmwO3VNJ

New hot fixes and tips are always posted here: https://trello.com/c/lTRmC9Yv

Please feel free to send your bug reports, feedback, suggestions, questions or feature requests to: support@deepdreamgames.com
In order to help me resolve your issue faster – please make sure to provide the following information in your email:

1. Invoice number in case you're asking for support for the first time. This isn't 100% necessary, but I'm prioritizing emails from users with known invoice numbers, since that allows me to instantly send them modified scripts and/or shaders of the Highlighting System, or even the whole package as soon as I have a solution for any particular issue. You can find your invoice number in the PDF attached to the 'Unity Asset Store purchase confirmation' email here:



unity

UNITY3D.COM

Unity Technologies ApS
Løvstræde 5
1152 Copenhagen K
Denmark

INVOICE

Salesperson	Unity Asset Store	Invoice No.	302962348
Salesperson E-Mail	assetstore@unity3d.com	Date	
Sell-to Contact		Order No.	
Sell-to Contact E-Mail			
PO No.			

Page 1

Description	Quantity	Unit Price	Amount
1	1		
		Total	

2. Unity version
3. Highlighting System version (v4.3 in case this Documentation is provided to you along with the package)
4. Operating system version (e.g. Windows 10 64-bit)
5. Graphics card model (e.g. NVIDIA GeForce GTX 980)
6. Mobile device version in case of mobile-related issues (please find your device on

http://www.gsmarena.com/ and include the link. For example: http://www.gsmarena.com/asus_zenfone_3_ze552kl-8106.php)

7. (Optional) Screenshots or videos depicting the problem. This is optional, but often 1 screenshot worth 1000 words ;)
8. (Optional) Archived example project to reproduce your issue (either attached to email if it's below 20Mb, or a link to any file sharing service). Please note that this is not necessary to include the *Library* folder (only *Assets* and *ProjectSettings* is required).