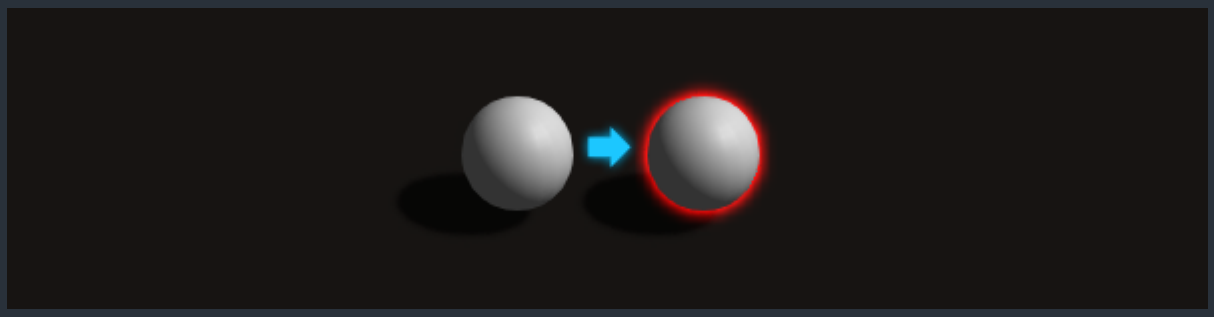
Highlighting System 5.0

User Guide



Highlighting System for Unity Engine available at: http://u3d.as/hUz

Online documentation available at: http://docs.deepdream.games/HighlightingSystem/5.0/

Contents

1 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.

1.1 Package overview

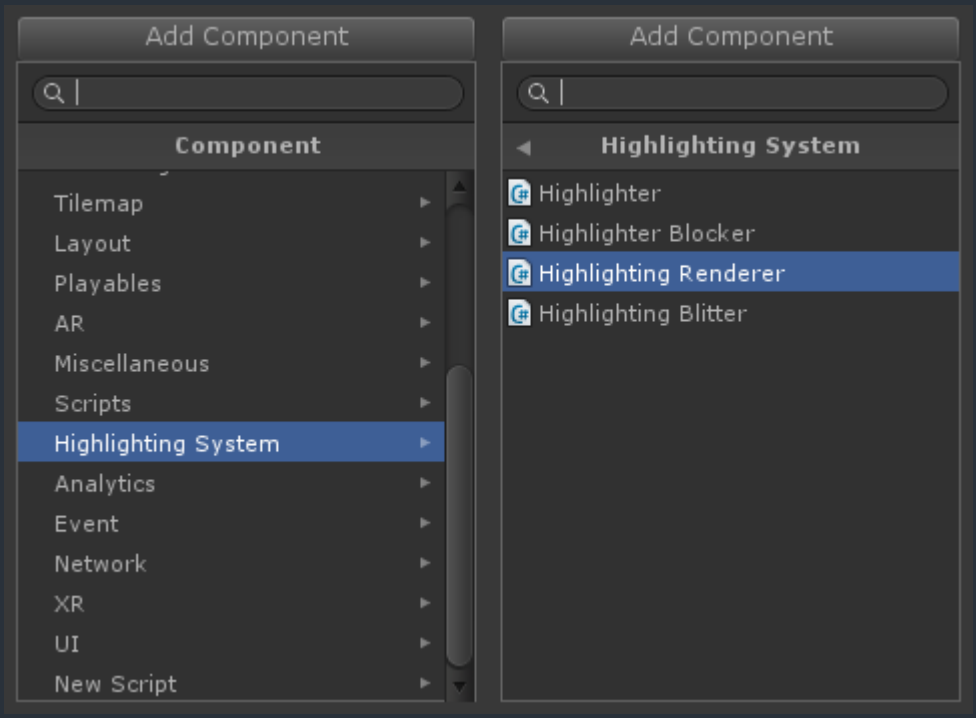
After the package installation, in the *Plugins\HighlightingSystem* folder you will find all the scripts and shaders required for the Highlighting System to work. All scripts in this folder are in the `HighlightingSystem` namespace.

In 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. All scripts in this folder are in the `HighlightingSystem.Demo` namespace.

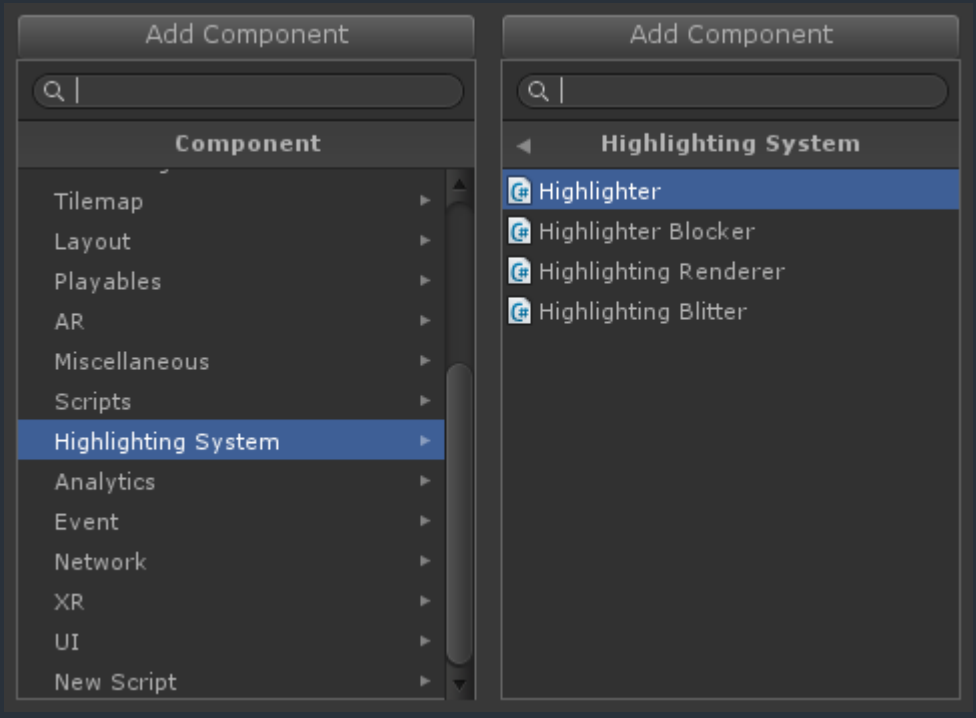
2 Integration to your project

2.1 Basic integration

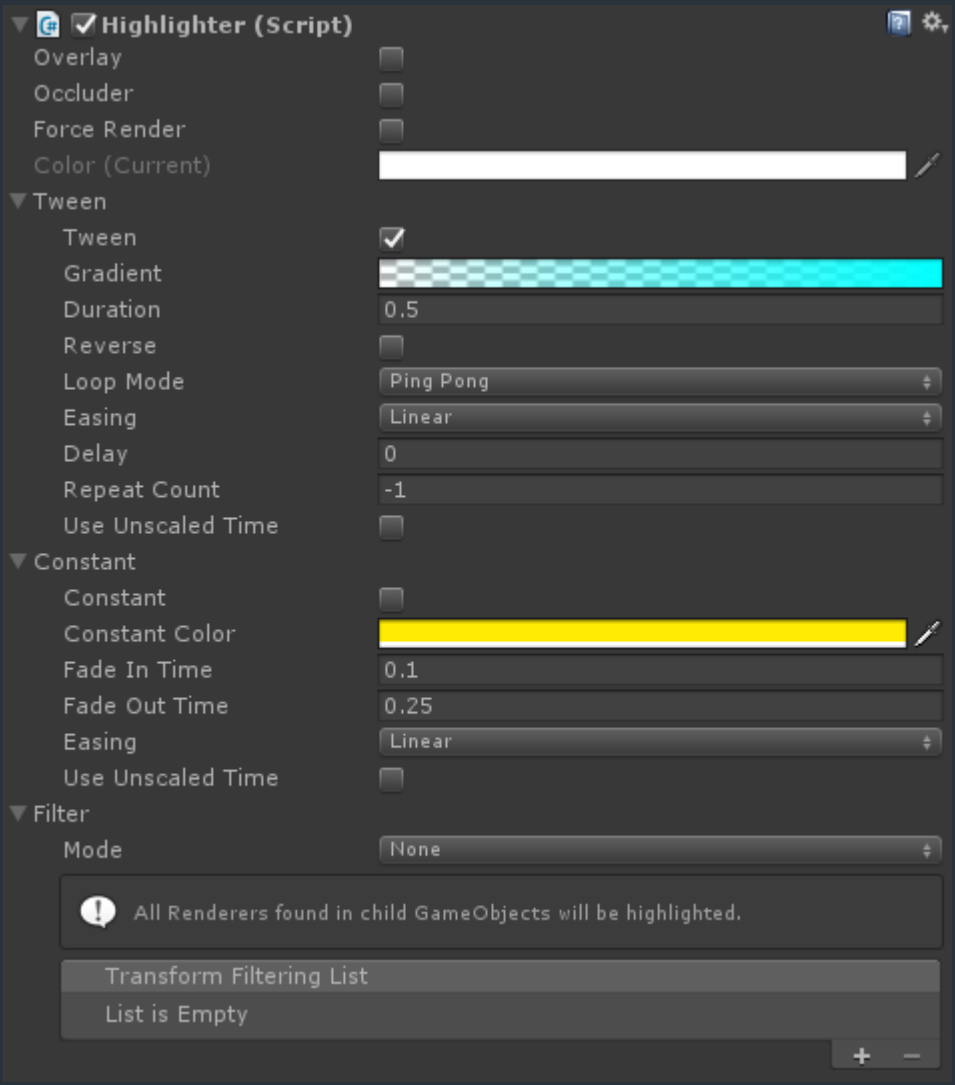
1. Import Highlighting System package from the Unity Asset Store to your project.
2. Add `HighlightingRenderer` component to the Camera. Order of this component (among other Image Effects on this Camera) defines when the highlighting buffer will be applied to the rendered frame.



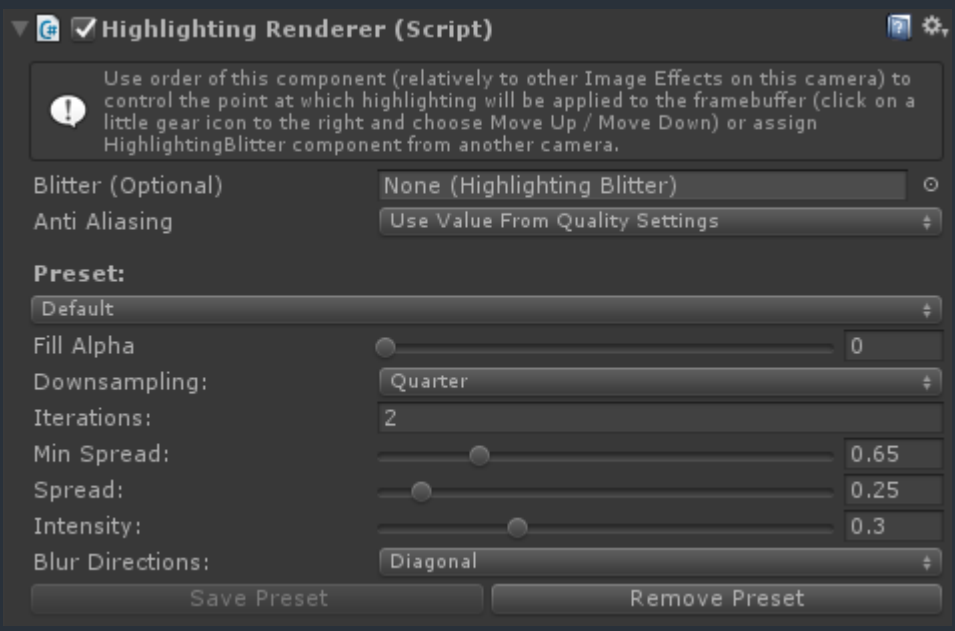
3. Add `Highlighter` component to the `GameObject`'s you want to make highlightable.



4. Setup `Highlighter` components as you need. Refer to the Highlighter API to figure out what each option means.

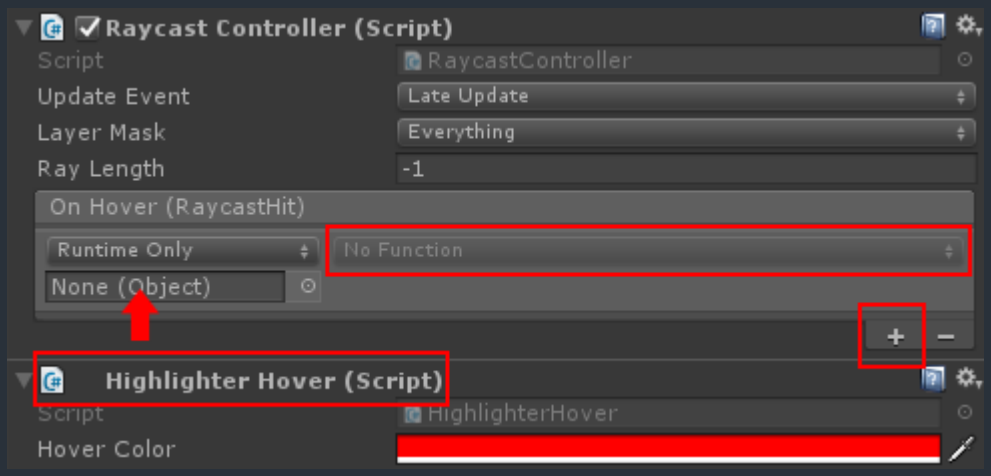


5. Tweak settings of the `HighlightingRenderer` component to change the look of highlighting. Please note that the highlighting presets are stored (serialized) in the component itself (will be saved in scenes and / or prefabs), and you can manipulate them at runtime using `HighlightingRenderer` API.



2.2 Hover highlight

1. Add `RaycastController` component to `Camera`.
2. Add `HighlighterHover` component to `Camera` or any other persistent `GameObject`.
3. Hook `HighlighterHover` to the `OnHover` event of the `RaycastController` by pressing the *+* button on the bottom right first, then dragging `HighlighterHover` component to the appeared *None (Object)* field and selecting *HighlighterHover > OnHover* function. You can also hook any other custom handlers to the `OnHover` event this way.



4. Set desired highlighting hover color in the `HighlighterHover` component.

See also `HighlighterInteractionDemo` class source for an example of more advanced interaction with the `Highlighter`'s.

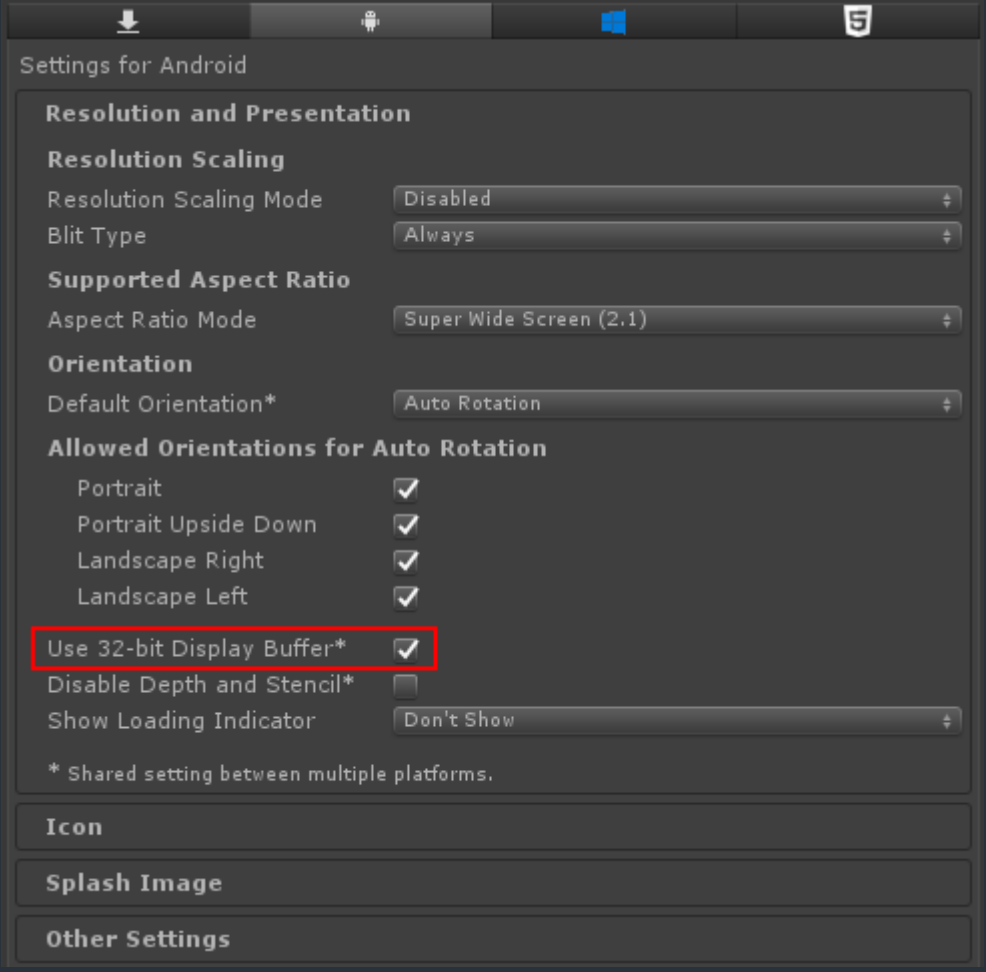
2.3 Quick tips

- *Tween* has higher priority than *Constant* highlighting, so if both modes are enabled on the `Highlighter` - you won't see any effects from tweaking *Constant* highlighting properties (you can change priorities in `Highlighter.UpdateHighlighting()` method source by simply rearranging pieces of code responsible for each highlighting mode).

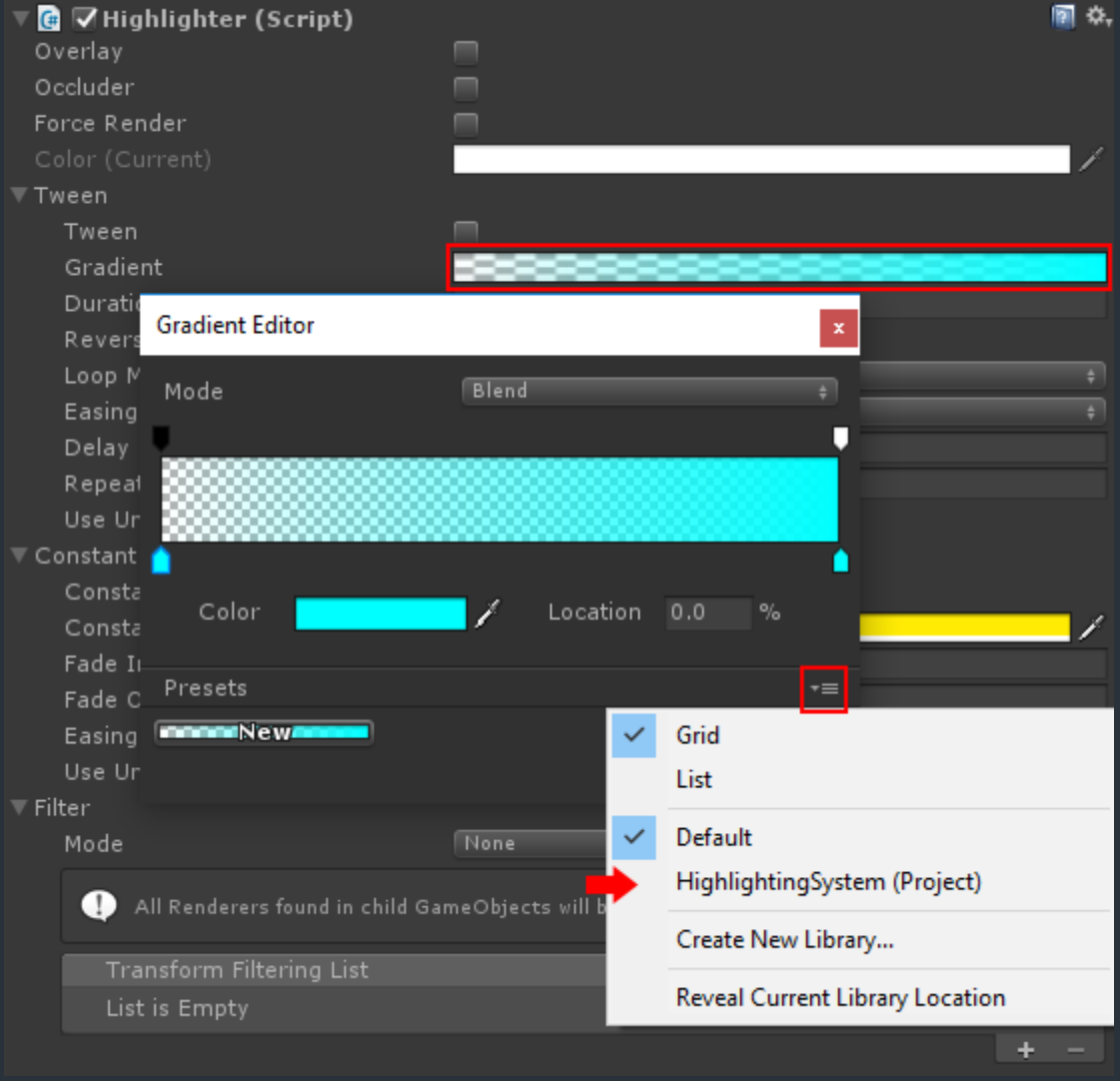
- You can control `Highlighter`'s directly from scripts by using their API. For example:

```
using UnityEngine;
using HighlightingSystem;

public class Example : MonoBehaviour
{
 // Assign Highlighter component to this field in Inspector
 public Highlighter highlighter;

 void Awake()
 {
   highligter.ConstantOn(Color.red);
 }
}
```

- Don't forget to add `using HighlightingSystem;` directive to the beginning of your scripts in order to be able to access Highlighting System API.
- If you doesn't want to use `HighlighterHover` - to manually highlight object only for a single frame call `Hover(Color.red)` on the target `Highlighter` (See *HighlighterHover.cs* and *HighlighterInteractionDemo.cs* scripts for an example).
- If you want to implement completely custom highlighting logic from scratch - derive your class from the `HighlighterCore`.
- When configuring `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's `Renderer` class can be highlighted. By default - highlighting of the following renderers is enabled: `MeshRenderer`, `SkinnedMeshRenderer`, `SpriteRenderer`, `ParticleSystemRenderer`. Feel free to tune types list globally in the `DefaultRendererFilter()` method source as you need, or implement custom `RendererFilter`.
- On mobile platforms, don't forget to set the *Use 32-bit Display Buffer* checkbox under the *Resolution and Presentation* section of the Unity *Player Settings*.



- Highlighting System comes with several built-in color gradients (located in *HighlightingSystemDemo/Editor/HighlightingSystem.gradients* gradients library file). In order to use them, you can switch to this library here:


Highlighter (Script)
Overlay
Occluder
Force Render
Color (Current)
Tween
Tween
Gradient
Gradient Editor
Mode
Blend
Color
Location
0.0
%
Presets
New
Grid
List
Default
HighlightingSystem (Project)
Create New Library...
Reveal Current Library Location
Constant
Filter
Mode
None
All Renderers found in child GameObjects will b
Transform Filtering List
List is Empty

3 API

3.1 Highlighter

class in `HighlightingSystem` / Inherits from: `HighlighterCore`

Description:
Main component to use on the objects you want to make highlightable. Three different highlighting modes available (listed in descending priority order):

1. **Hover**

   Highlights object only for a single frame, so you can trigger it every frame for the object under the mouse cursor.

2. **Tween**

   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**

   Not actually a highlighting mode, but highlighter will turn into occluder only if no other modes are active.

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

```
using UnityEngine;
using HighlightingSystem;

public class Example : MonoBehaviour
{
  private Highlighter highlighter;

  void Awake()
  {
    highlighter = gameObject.AddComponent<Highlighter>();
    highlighter.ConstantOn(Color.red);
  }
}
```

See also: `HighlighterCore`

Properties:

- **`bool constant`**
  Enables constant highlighting.
- **`Color constantColor`**
  Constant highlighting color.
- **`Easing constantEasing`**
  Defines alpha channel curve used to perform constant highlighting fade in / out transitions. See `Easing`.
- **`float constantFadeInTime`**
  Time in seconds it will take for constant highlighting to fade in. (`color.a` will be changing from `0` to `constantColor.a` during that time).
- **`float constantFadeOutTime`**
  Time in seconds it will take for constant highlighting to fade out. (`color.a` will be changing from `constantColor.a` to `0` during that time).
- **`float constantFadeTime`**
  Shortcut to set `constantFadeInTime` and `constantFadeOutTime` at once.
- **`bool constantUseUnscaledTime`**
  Use `Time.unscaledTime` instead of `Time.time` when performing constant fade in / out transitions.
- **`List<Transform> filterList`**
  List of `Transform` components to use in filtering. Make sure to trigger `SetDirty()` after modifying this list in order for changes to take effect. See also: `filterMode`.
- **`RendererFilterMode filterMode`**
  Defines Renderers filtering mode. Does nothing if custom rendererFilter is assigned. See `RendererFilterMode`.
- **`bool occluder`**
  When enabled - if object is not highlighted, it will turn into highlighting occluder. (See '05 OccluderModes' demo scene for an example). Most likely you'll want to enable this on your main character. See also: `HighlighterMode.Occluder`.
- **`bool overlay`**
  When enabled - highlighting for this object will be rendered on top of any other geometry. See also: `HighlighterMode.Overlay`

- **`bool tween`**
  Enables tween highlighting.

- **`float tweenDelay`**
  Delay in seconds before tween will start playing. Can also be used to shift tween start position by setting to negative value (e.g. `tweenDelay = -Random.value * tweenDuration;`).

- **`float tweenDuration`**
  Time in seconds for tween to playback once.

- **`Easing tweenEasing`**
  Defines how tween highlighting color gradient value is evaluated. See `Easing`.

- **`Gradient tweenGradient`**
  Tween color gradient.

- **`LoopMode tweenLoop`**
  See `LoopMode`.

- **`int tweenRepeatCount`**
  Number of times tween will be played (or −1 to play forever). This value will be taken into account only if `tweenLoop` is set to `LoopMode.Loop` or `LoopMode.PingPong`.

- **`bool tweenReverse`**
  Tween will play in reverse when this flag is set.

- **`bool tweenUseUnscaledTime`**
  Use `Time.unscaledTime` instead of `Time.time` for tween playback.

Public Methods:

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

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

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

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

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

- `void ConstantOnImmediate()`
  Turn on constant highlighting immediately (without fade in).
- `void ConstantSet(float fadeTime, bool value)`
  Base method for setting constant highlighting mode.
- `void ConstantSwitch(float time)`
  Switch constant highlighting using specified transition duration.
- `void ConstantSwitchImmediate()`
  Switch constant highlighting immediately (without fade in/out).
- `void Hover(Color color)`
  Turn on highlighting only in current frame using specified color.
- `void Off()`
  Turn off all highlighting modes.
- `void TweenSet(bool value)`
  Base method for setting tween highlighting mode.
- `void TweenStart()`
  Shortcut for `TweenSet(true)`
- `void TweenStop()`
  Shortcut for `TweenSet(false)`

Static Methods:

- `Color HSVToRGB(float hue, float saturation, float value)`
  Converts hue, saturation and value parameters into corresponding color.

3.2 HighlighterBlocker

class in `HighlightingSystem` / Inherits from: `MonoBehaviour`

Description:

`Renderer`'s on `GameObject`'s with this component (or any of it's children) will never be highlighted.

3.3 HighlighterCore

class in `HighlightingSystem` / Inherits from: `MonoBehaviour`

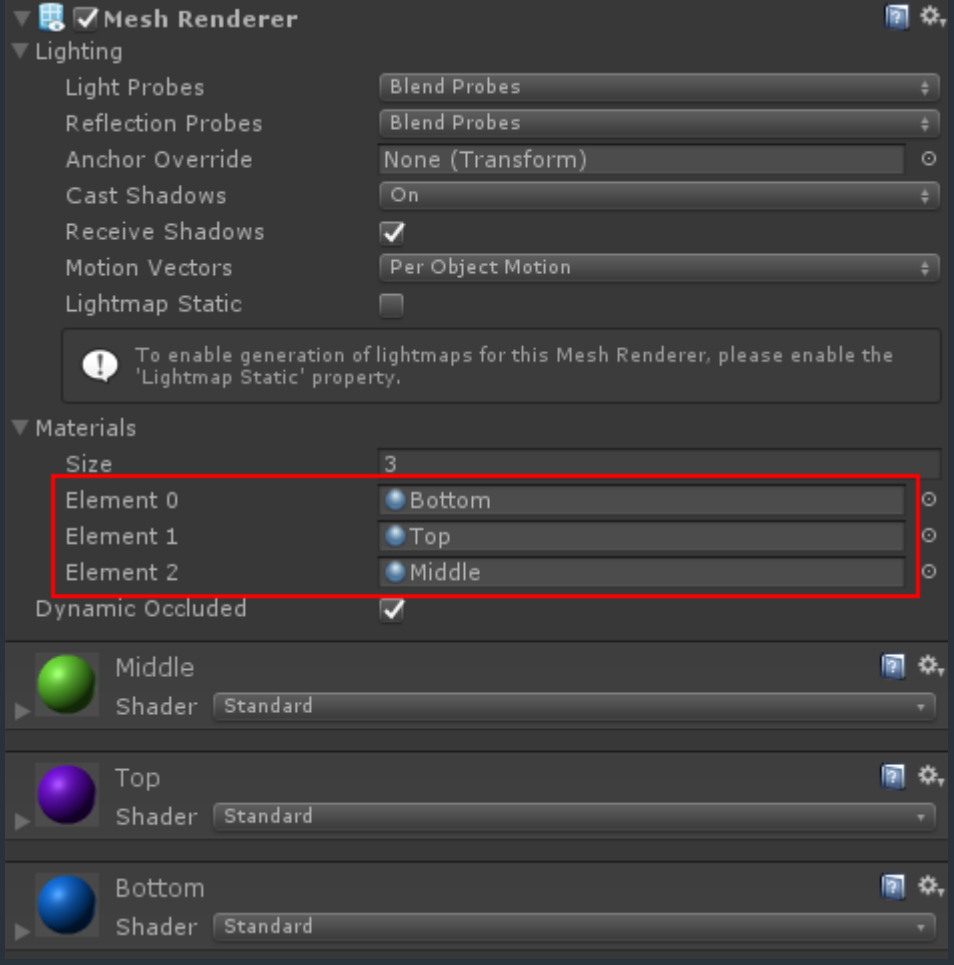
Description:

Base class for all highlighters. If you doesn't want to use provided `Highlighter` component and want to implement custom highlighting logic instead - inherit from this class and override `UpdateHighlighting()` method where you should update `mode` and `color` properties. See `Highlighter` class sources for an example. Please note that for safety (to avoid exceptions), this class hides `MonoBehaviour`'s `Awake()`, `OnEnable()`, `OnDisable()`, `OnDestroy()` methods from all inheritors, so instead you should override `AwakeSafe()`, `OnEnableSafe()`, `OnDisableSafe()`, `OnDestroySafe()` methods correspondingly.

Delegates:

- **`bool RendererFilter(Renderer renderer, List<int> submeshIndices)`**
  Delegate to use for `globalRendererFilter` and `rendererFilter`, which will be triggered after each `SetDirty()` call to update renderers which should be highlighted.

  In your own implementation of this delegate - return true to highlight renderer passed as an argument and fill submeshIndices list with the list of submesh indices which should be highlighted (or −1 to highlight all of them). Submesh indices correspond to the material indices in Materials list of the Renderer component:



Make sure to keep this delegate implementation as simple as possible or you may experience performance degradation. Refer to the `DefaultRendererFilter` for an implementation example.

Properties:

- **`Color color`**
  Color to use for highlighting.

- **`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 planes 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.

- **`HighlighterMode mode`**
  See `HighlighterMode`.

- **`RendererFilter rendererFilter`**
  Renderer filter to use for this `HighlighterCore` instance. If set to none - `globalRendererFilter` will be used instead.

Public Methods:

- **`void SetDirty()`**
  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.

Static Properties:

- **`RendererFilter globalRendererFilter`**
  Global renderer filter which is going to be used if `rendererFilter` is not explicitly assigned.

- **`ReadOnlyCollection<HighlighterCore> highlighters`**
  Collection of all enabled highlighters. Make sure to add `using System.Collections.ObjectModel;` directive in your scripts if you want to use this collection.

Static Methods:

- **`bool DefaultRendererFilter(Renderer renderer, List<int> submeshIndices)`**
  Default renderer filter implementation which is going to be used if `globalRendererFilter` is not assigned.

```
static public bool DefaultRendererFilter(Renderer renderer, List<int> submeshIndices)
{
  // Do not highlight this renderer if it has HighlighterBlocker in parent
   if (renderer.GetComponentInParent<HighlighterBlocker>() != null) { return false; }

  bool pass = false;

  if (renderer is MeshRenderer) { pass = true; }
  else if (renderer is SkinnedMeshRenderer) { pass = true; }
  else if (renderer is SpriteRenderer) { pass = true; }
  else if (renderer is ParticleSystemRenderer) { pass = true; }

  if (pass)
  {
    // Highlight all submeshes
    submeshIndices.Add(-1);
  }

  return pass;
}
```

3.4 HighlightingBase

class in `HighlightingSystem` / Inherits from: `MonoBehaviour`

Description:

Internal base class which implements core `Highlighter`'s rendering functionality. In most cases you should use `HighlightingRenderer` instead.

Properties:

- **`AntiAliasing antiAliasing`**
  Anti-aliasing value for the highlighting `RenderTexture`. Set to `AntiAliasing.QualitySettings` by default.

- **`HighlightingBlitter blitter`**
  Get or set `HighlightingBlitter` instance, which will be used to blit highlighting rendering results. Set to null to make `HighlightingBase` blit during the `OnRenderImage(RenderTexture src, RenderTexture dst)` callback (default behaviour for most Image Effects in Unity).

- **`BlurDirections blurDirections`**
  Defines directions in which highlighting buffer will be shifted/blurred (default is `BlurDirections.Diagonal`). This option allows finer control of solid highlighting modes. `BlurDirections.All` is more expensive than `BlurDirections.Diagonal` or `BlurDirections.Straight`. See Also: `BlurDirections`

- **`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 the highlighting buffer by dividing frame buffer (screen) size with this value.

- **`float fillAlpha`**
  Global inner fill alpha value. Currently, there is no way to make it work on a per-instance basis, so this setting will affect all `Highlighter`'s.

- **`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 of all highlighting shaders.

- **`int iterations`**
  Blur iterations. Number of blur iterations to be performed on the highlighting buffer. Larger number means more blur.

Public Methods:

- **`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 `BeforeImageEffectsOpaque` camera event, so calling this method earlier will probably lead to undesired results.

3.5 HighlightingBlitter

class in `HighlightingSystem` / Inherits from: `MonoBehaviour`

Description:

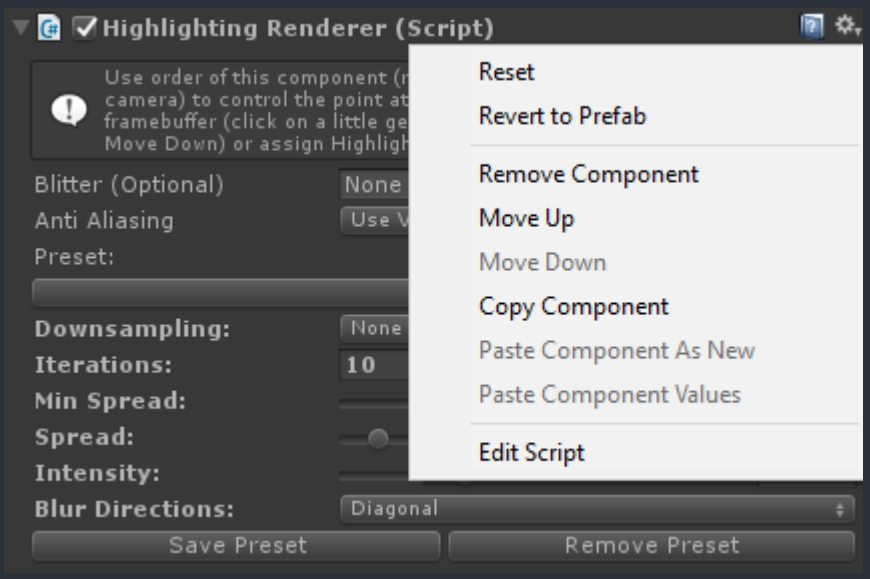
You should add this component to the current camera if you want to apply the highlighting buffer rendered on another camera during current camera rendering. You should assign this component instance to the `blitter` field in that case. Order of this component (among other Image Effects on this Camera) will define the point at which highlighting buffer will be applied to the rendered frame. Make sure that the Camera with `HighlightingBlitter` component has higher depth than the Camera with `HighlightingRenderer` which is using this `HighlightingBlitter` component instance.

3.6 HighlightingRenderer

class in `HighlightingSystem` / Inherits from: `HighlightingBase`

Description:

Main component to assign to the Camera. This class, on top of the core `HighlightingBase` implements functionality to manipulate highlighters rendering presets. 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. 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):



Properties:

- **`ReadOnlyCollection<HighlightingPreset> presets`**
  Returns stored presets as `ReadOnlyCollection<HighlightingPreset>`. Make sure to add `using System.Collections.ObjectModel;` directive in your scripts if you want to use this collection.

Public Methods:

- **`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.

- **`void ApplyPreset(HighlightingPreset preset)`**
  Apply specified preset settings.

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

- `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 LoadPreset(string name)`
  Find stored preset by name and apply it's settings.

- `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.

Static Properties:

- `List<HighlightingPreset> defaultPresets`
  Readonly list of default presets.

3.7 HighlightingPreset

struct in `HighlightingSystem` / Inherits from: `ValueType`

Description:

Struct used to store `HighlightingRenderer` (`HighlightingBase`) settings.

Properties:

- **`BlurDirections blurDirections`**
  See `HighlightingBase.blurDirections`.
- **`float blurIntensity`**
  See `HighlightingBase.blurIntensity`.
- **`float blurMinSpread`**
  See `HighlightingBase.blurMinSpread`.
- **`float blurSpread`**
  See `HighlightingBase.blurSpread`.
- **`int downsampleFactor`**
  See `HighlightingBase.downsampleFactor`.
- **`float fillAlpha`**
  See `HighlightingBase.fillAlpha`.
- **`int iterations`**
  See `HighlightingBase.iterations`.
- **`string name`**
  Preset name.

3.8 AntiAliasing

enum in `HighlightingSystem` / Inherits from: `Enum`

Description:

Anti-aliasing settings to use when rendering highlighting.

Static Properties:

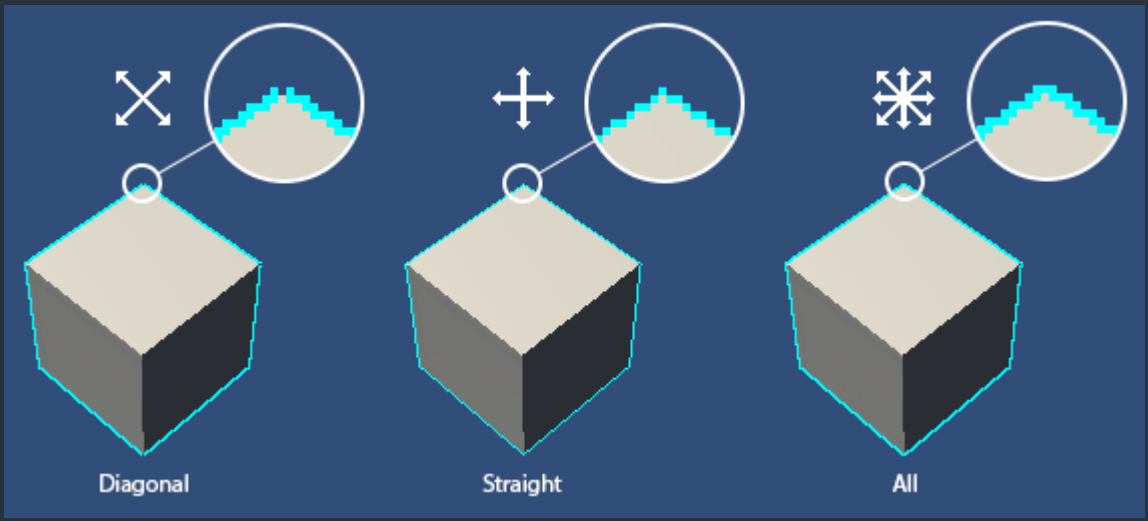
- **`QualitySettings`**
  Use `QualitySettings.antiAliasing` value (or `RenderTexture.antiAliasing` of `Camera.targetTexture` when rendering to texture).
- **`Disabled`**
  Disable multisample anti-aliasing.
- **`MSAA2x`**
  2x Multisample anti-aliasing.
- **`MSAA4x`**
  4x Multisample anti-aliasing.
- **`MSAA8x`**
  8x Multisample anti-aliasing.

3.9 BlurDirections

enum in `HighlightingSystem` / Inherits from: `Enum`

Description:

Defines blur directions of the highlighting buffer.



Static Properties:

- `Diagonal`
  Blur in diagonal directions (top-left, top-right, bottom-left, bottom-right).
- `Straight`
  Blur in straight directions (top, bottom, left, right).
- `All`
  Blur in diagonal and straight directions (top, bottom, left, right, top-left, top-right, bottom-left, bottom-right).

3.10 Easing

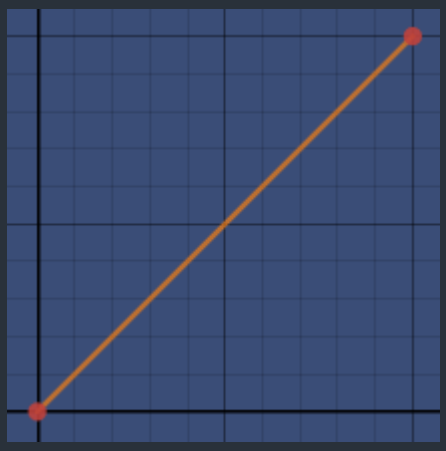
enum in `HighlightingSystem` / Inherits from: `Enum`

Description:

Defines how tween highlighting color gradient value is evaluated or how constant highlighting alpha channel value is eased during fade in and out transitions.

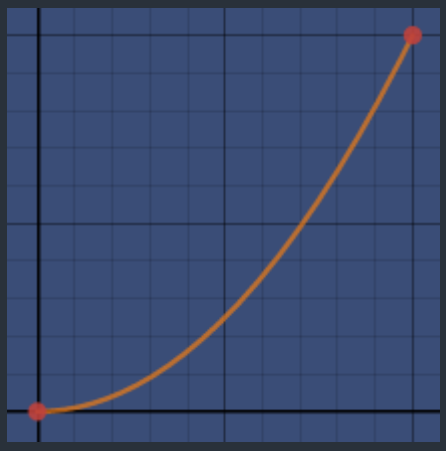
Static Properties:

- `Linear`



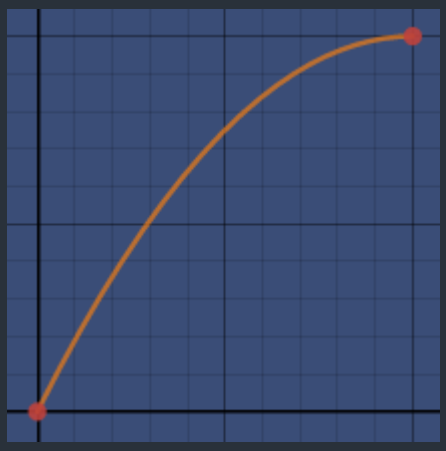
```
y = x;
```

- `QuadIn`



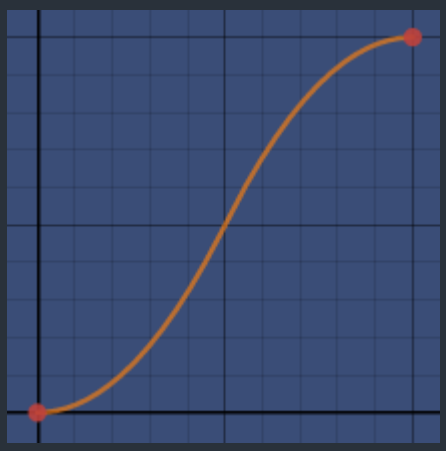
```
y = x * x;
```

- QuadOut



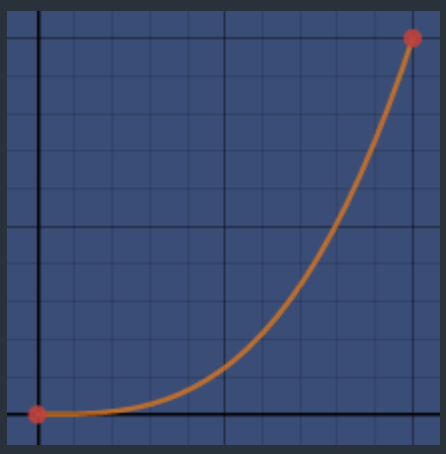
```
y = -x * (x - 2f);
```

- QuadInOut



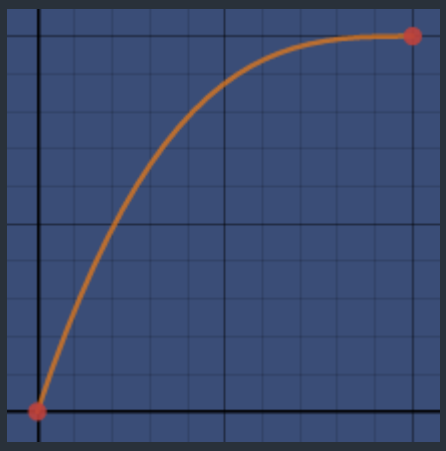
```
y = x < 0.5f ? 2f * x * x : 2f * x * (2f - x) - 1f;
```

- CubicIn



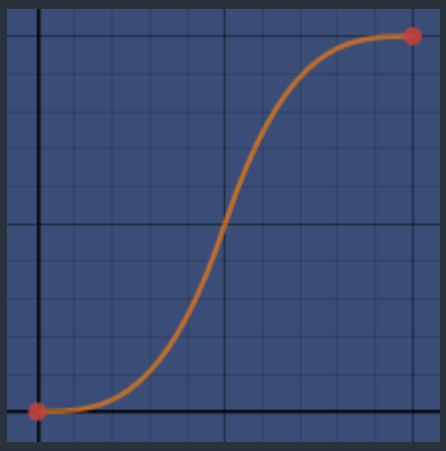
```
y = x * x * x;
```

- **CubicOut**



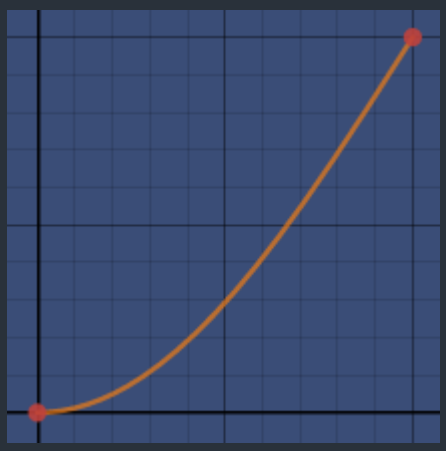
```
x = x - 1f;
y = x * x * x + 1f;
```

- **CubicInOut**



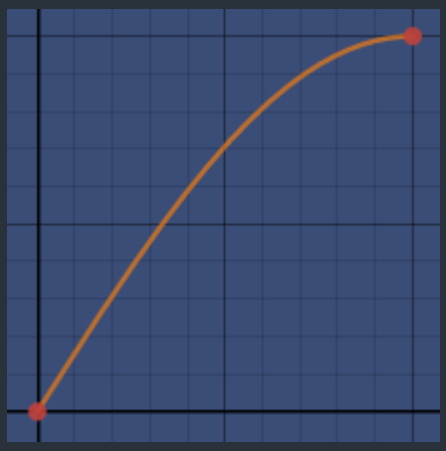
```
if (x < 0.5f)
{
  y = 4f * x * x * x;
}
else
{
  x = 2f * x - 2f;
  y = 0.5f * (x * x * x + 2f);
}
```

- **SineIn**



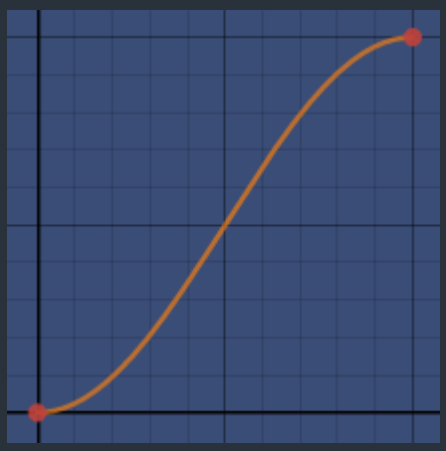
```
y = 1f - Mathf.Cos(x * Mathf.PI * 0.5f);
```

- **SineOut**



```
y = Mathf.Sin(x * Mathf.PI * 0.5f);
```

- **SineInOut**



```
y = -0.5f * (Mathf.Cos(x * Mathf.PI) - 1f);
```

3.11 HighlighterMode

enum in `HighlightingSystem` / Inherits from: `Enum`

Description:

Determines how to render each `HighlighterCore` instance.

Static Properties:

- **`Disabled`**
  Highlighting will not be rendered in this mode.
- **`Default`**
  Highlighting will be occluded by other geometry.
- **`Overlay`**
  Highlighting will not be occluded by other geomtry.
- **`Occluder`**
  Turn object into highlighting occluder (the one which always occludes any highlighting over it's shape).

3.12 LoopMode

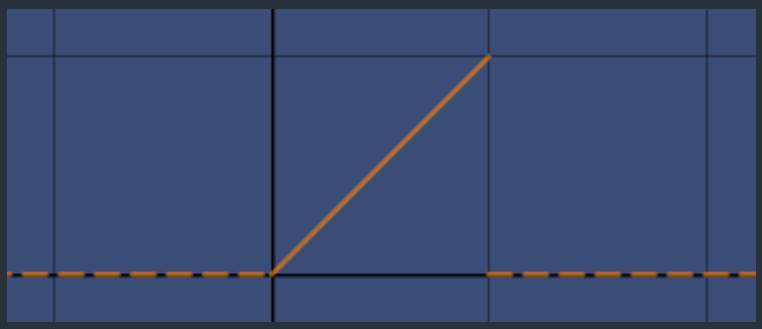
enum in `HighlightingSystem` / Inherits from: `Enum`

Description:

Determines how `tweenGradient` is sampled outside of a single tween iteration.

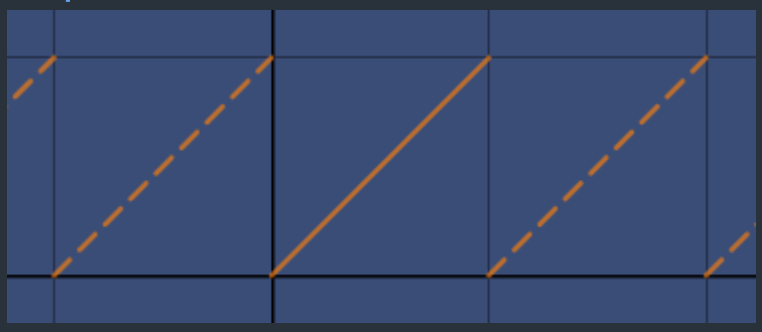
Static Properties:

- **Once**



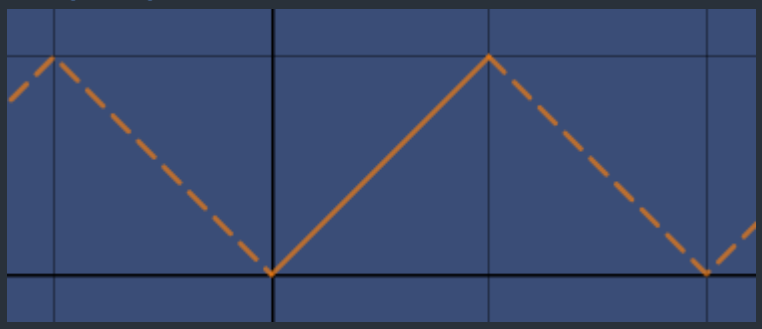
When tween reaches the end of a single iteration - it will automatically stop playing.

- **Loop**



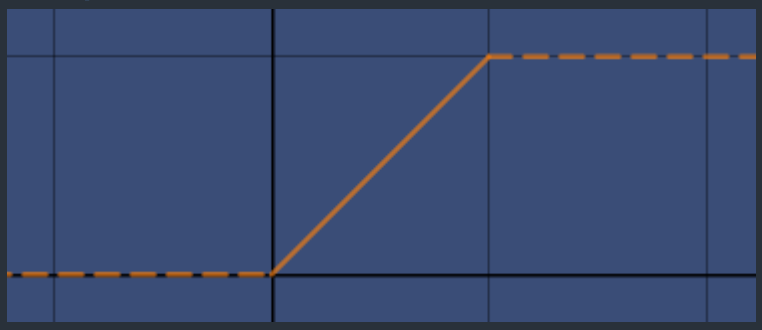
When tween reaches the end of a single iteration - it will start over from the beginning.

- **PingPong**



When tween reaches the end of a single iteration - it will ping pong back between beginning and end.

- `ClampForever`



When tween reaches the end of a single iteration - it will stay at the last color value specified in gradient forever.

3.13 RendererFilterMode

enum in `HighlightingSystem` / Inherits from: `Enum`

Description:

Defines how `Highlighter` will utilize `filterList` to perform `Renderer`'s filtering.

Static Properties:

- **None**
  All `Renderer`'s found in child `Transform`'s will be highlighted.
- **`Include`**
  `Renderer`'s only on `Transform`'s (and any of their children) specified in `filterList` will be highlighted.
- **`Exclude`**
  `Renderer`'s on `Transform`'s (and any of their children) specified in `filterList` will be excluded from highlighting.

4 Advanced tips

4.1 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 the 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 `SetDirty()`. Because of that, your changes to the main texture properties will not be reflected by the highlighting without the call to `SetDirty()` 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.

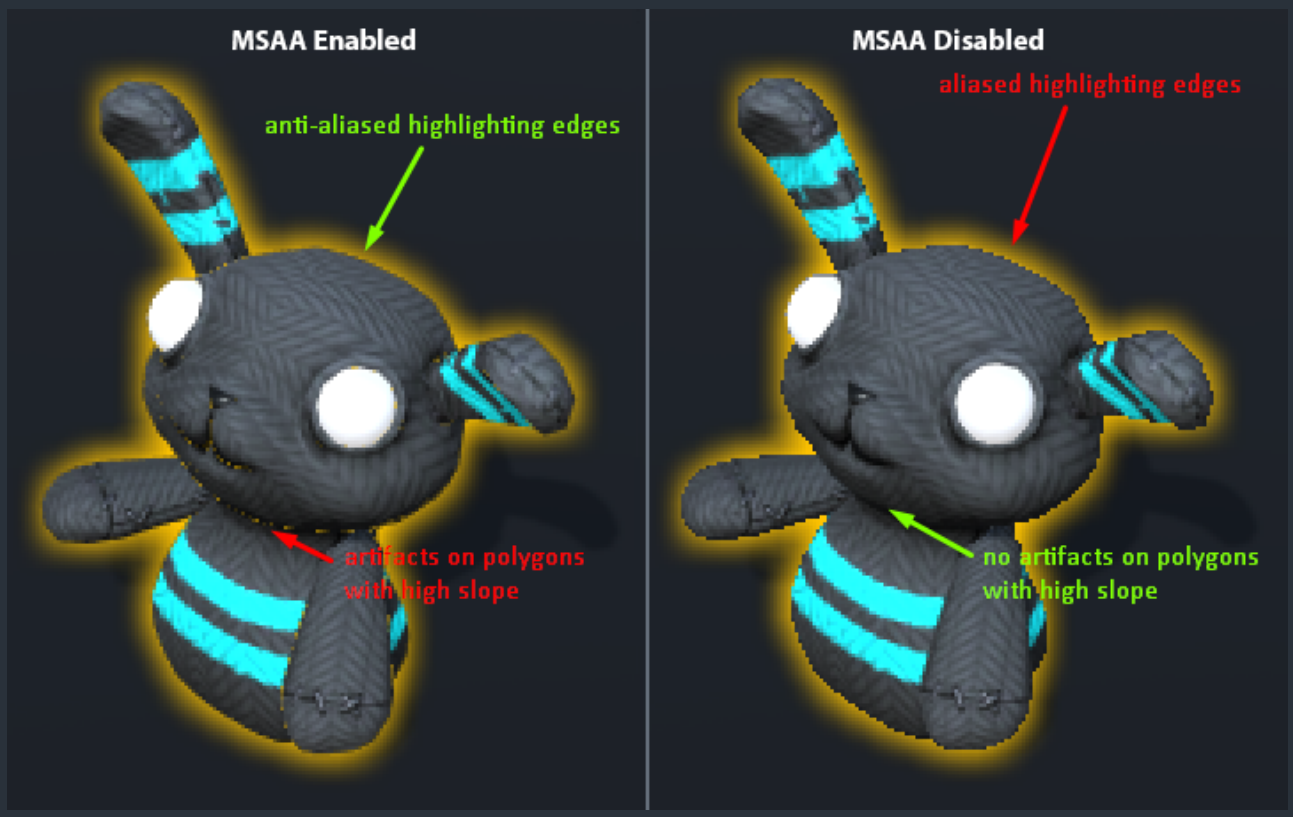
4.2 Anti-aliasing

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.

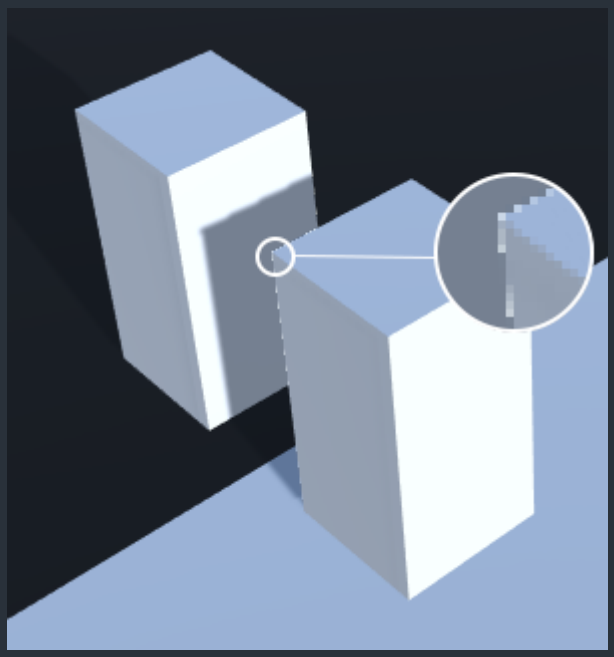
Hardware anti-aliasing has several significant drawbacks:

- 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 like there is currently no way to fix that:

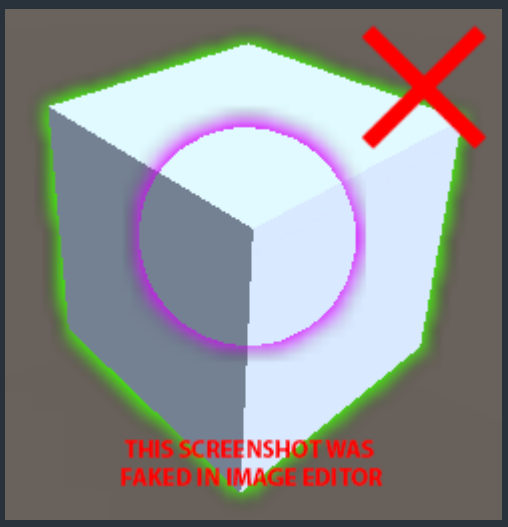


Due to all of the above – it is not recommended to use hardware anti-aliasing in your project. You can replace it with *Antialiasing* Image Effect from the Unity Standard Assets package.

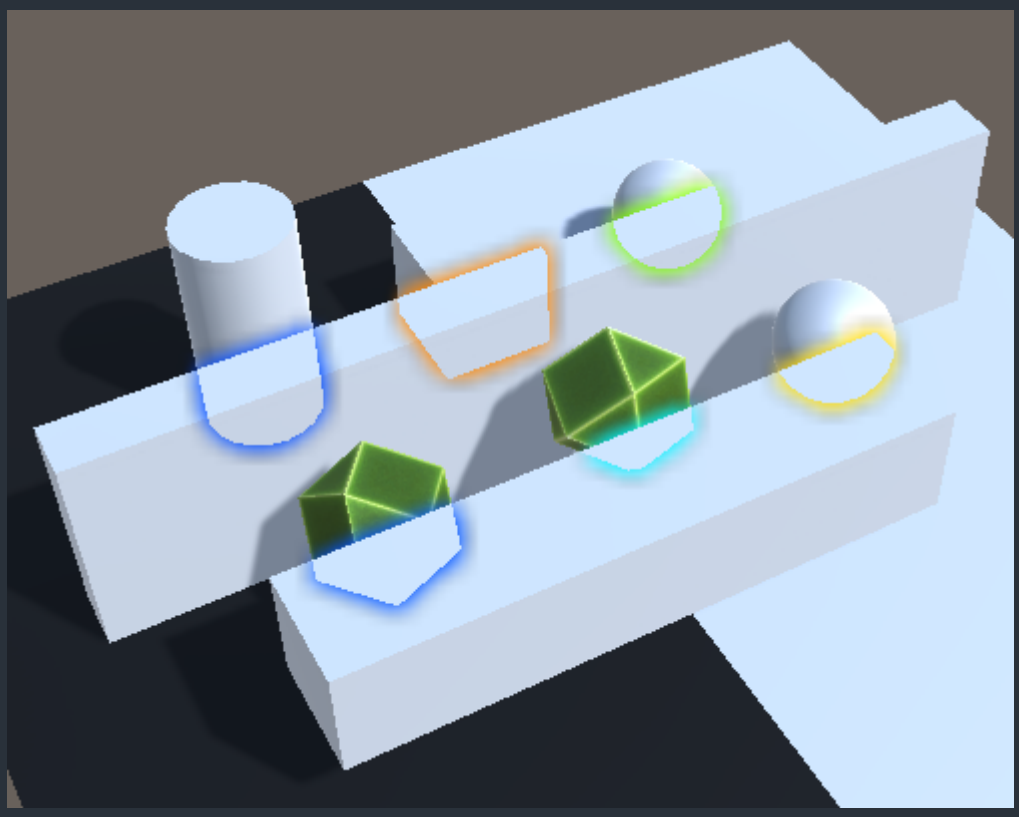
5 Limitations

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

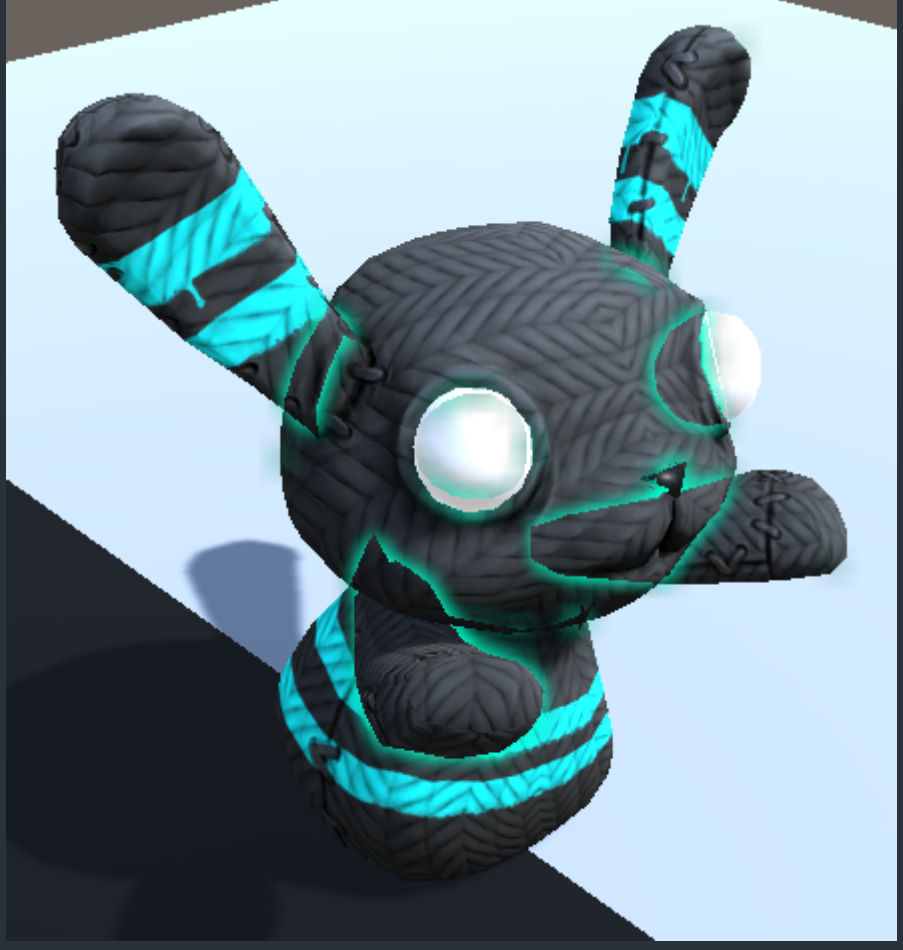
1. Multi-layer highlighting. This isn't possible to show highlighting of an object which is obscured by other highlighted object.



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.

6 Known issues

1. Currently (in Unity 2018.1.0f1), there is no API exposed to properly support *Single Pass Instanced (Preview)* stereo rendering path for Image Effects.
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.

7 Support

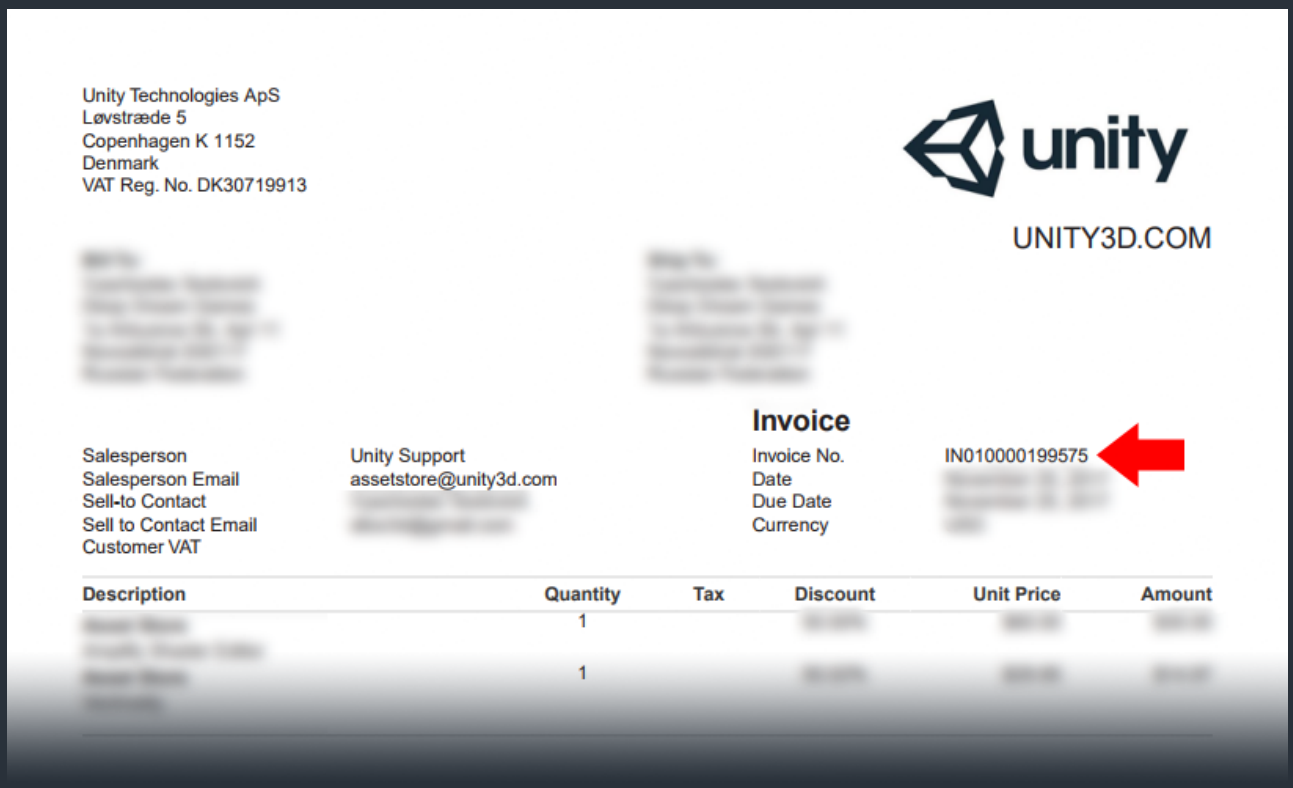
Here you can find Highlighting System development board which you can use to check if a specific bug fix or a feature is already known or in development: http://trello.com/b/GmwO3VNJ

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@deepdream.games

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 Technologies ApS
Løvstræde 5
Copenhagen K 1152
Denmark
VAT Reg. No. DK30719913

unity

UNITY3D.COM

Invoice

Salesperson	Unity Support	Invoice No.	IN010000199575
Salesperson Email	assetstore@unity3d.com	Date	
Sell-to Contact		Due Date	
Sell to Contact Email		Currency	
Customer VAT			

Description	Quantity	Tax	Discount	Unit Price	Amount
	1				
	1				

2. Unity version
3. Highlighting System version (v5.0 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) 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).
8. (Optional) Screenshots or videos depicting the problem. This is optional, but often 1 screenshot worth 1000 words ;)

8 Changelog

8.1 v5.0

- Fully reworked and improved `Highlighter` component which is now capable of covering more than 90% of all highlighting use-cases without any scripting.
- Now you can inherit from the `HighlighterCore` class to implement custom highlighting logic without having to worry about breaking compatibility with the future versions of the package.
- `Highlighter` / `HighlighterCore` has been optimized and is no longer using `Update()` or Coroutines. (See 10000 update calls Unity Blog post for more info)
- Flashing highlighting mode has been replaced with fully-featured tween, which supports color gradients.
- Implemented ability to specify Renderers and submeshes of each Renderer to highlight (`RendererFilter` 's).
- Fully reworked demo scripts intended to teach you the best and most performant ways of using Highlighting System in your own projects.
- Implemented solid `fill alpha` value control (affects all `Highlighter` 's).
- Orthographic Camera Projection mode compatibility.
- Fixed rendering to texture when Stereo Rendering is enabled.
- Improved documentation, which is now also available online.
- All demo scripts have been moved into the HighlightingSystem.Demo namespace to avoid type name collisions. You will have to add `using HighlightingSystem.Demo;` directive to reference them from your own scripts.
- Fixed highlighting of negatively scaled Renderers. https://trello.com/c/nUOlKQ6S
- Fixed `NullReferenceException` in `Highlighter.FillBufferInternal`. https://trello.com/c/zIfJdCkY
- Fixed setting material properties (e.g. `_Color`) via `MaterialPropertyBlock` also affects highlighting if property name matches. https://trello.com/c/WUfBXE8B
- Fixed *Dither.shader* won't compile as of Unity 2017. https://trello.com/c/QIM6kksZ

9 Upgrade notes

9.1 Upgrading from v4.3 to v5.0

1. Open Unity Asset Store and locate Highlighting System asset. Press *Update* button.

2. In the appeared *Import Unity Package* window - press *All* and then *Import* buttons.

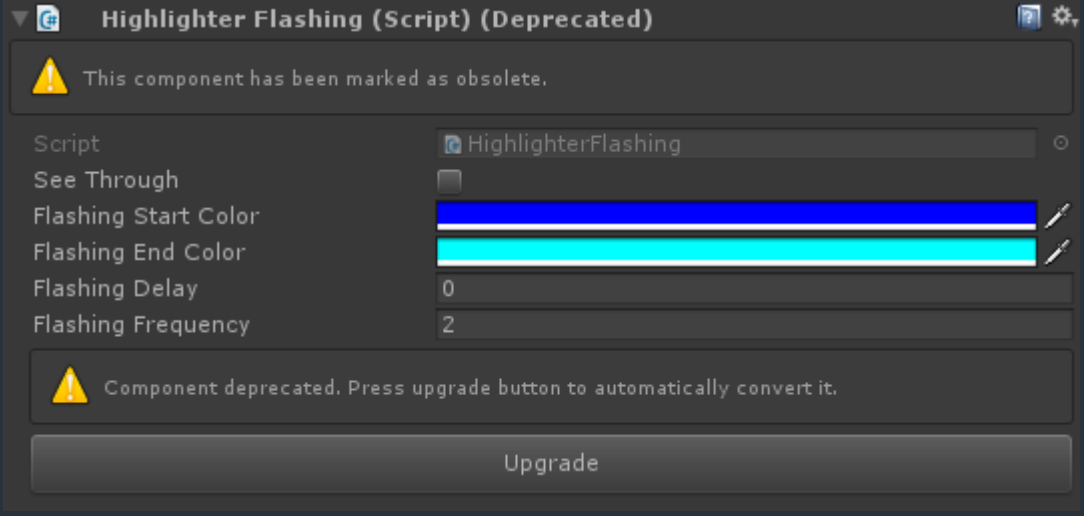
3. Remove the following files (you can use *Alt + Click* to expand all subdirectories recursively):

```
Assets
├──HighlightingSystemDemo
│  ├──Scripts
│  │  ├──Advanced
│  │  │  └──HighlighterItem.cs
│  │  ├──Basic
│  │  │  ├──BooHighlighterController.boo
│  │  │  └──JSHighlighterController.js
│  │  └──Service
│  │     ├──ColorTool.cs
│  │     ├──ScreenSpaceCanvas.cs
│  │     └──WorldSpaceCanvas.cs
│  └──Documentation.pdf
└──Plugins
   └──HighlightingSystem
      ├──Scripts
      │  └──Internal
      │     └──HighlighterInternal.cs
      └──link.xml
```

4. Rename *CameraTargeting* script (in the *Assets/HighlightingSystemDemo/Scripts/Advanced/* folder) to *HighlighterInteractionDemo*.

5. The following scripts have been deprecated: `HighlighterBase` `HighlighterConstant` `HighlighterFlashing` `HighlighterInteractive` `HighlighterOccluder` `HighlighterSpectrum`. If you haven't used these demo components in your project - simply remove them completely along with `HighlightingUpgrade` script.

   But if you do - you can automatically upgrade them to use `Highlighter` component now. In order to upgrade prefabs - press *Upgrade* button when you see such message, and save prefab:



In order to upgrade *GameObject*'s in the currently opened scene - open *Tools/HighlightingSystem/Upgrade scene from v4.3 to v5.0* utility window, press *Upgrade Current Scene* (all upgraded components will be logged to console) and save it.

6. Upgrade your custom scripts to use new API if you see similar warnings in console:

```
Assets/HighlightingSystemDemo/Scripts/Basic/BooHighlighterController.boo(18,15
):                                BCW0012:                                WARNING:
'HighlightingSystem.Highlighter.FlashingOn(UnityEngine.Color,
UnityEngine.Color, single)' is obsolete.
```

Refer to the updated `Highlighter`, `HighlightingBase` and `HighlightingRenderer` API.

7. Other important changes to take into account when upgrading:
   - `seeThrough` property of the `Highlighter` renamed to `overlay`.
   - `ReinitMaterials()` method of the `Highlighter` renamed to `SetDirty()`;
   - Removed deprecated `Highlighter` API methods: `SeeThrough(bool state)`, `SeeThroughOn()`, `SeeThroughOff()`, `SeeThroughSwitch()`, `OccluderOn()`, `OccluderOff()`, `OccluderSwitch()`
   - Deprecated `Die()` method of the `Highlighter`. Use `Destroy(highlighter);` instead.

In memory of my mother Nina and my friend Darwin.