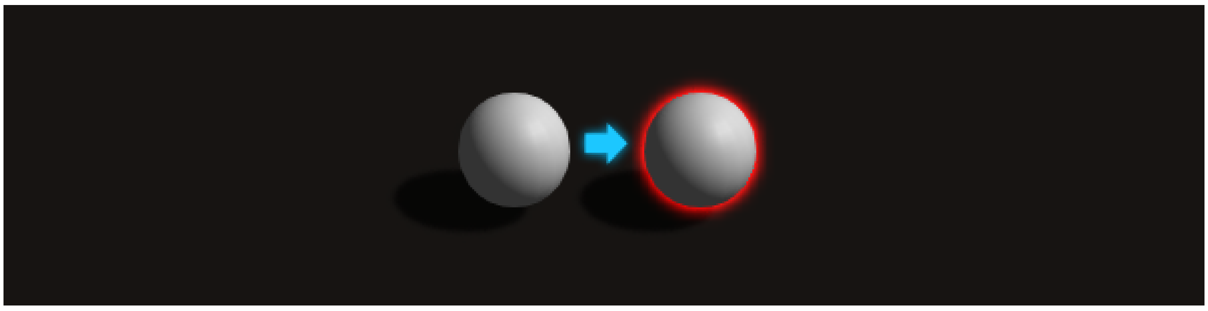


Highlighting System 2.0

User Guide

Table of Contents

1 Changelog

v2.0

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

v1.1

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

v1.0

- Initial release

2 Upgrade guide

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

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

3 Overview

Highlighting System package allows you to easily integrate outline glow effect for objects highlighting in your Unity3d project. It allows you to make any object highlightable. It designed with optimization in mind, so any material operations is performed only when that's really needed. This system must work on all major platforms, where post-processing effects are supported.

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

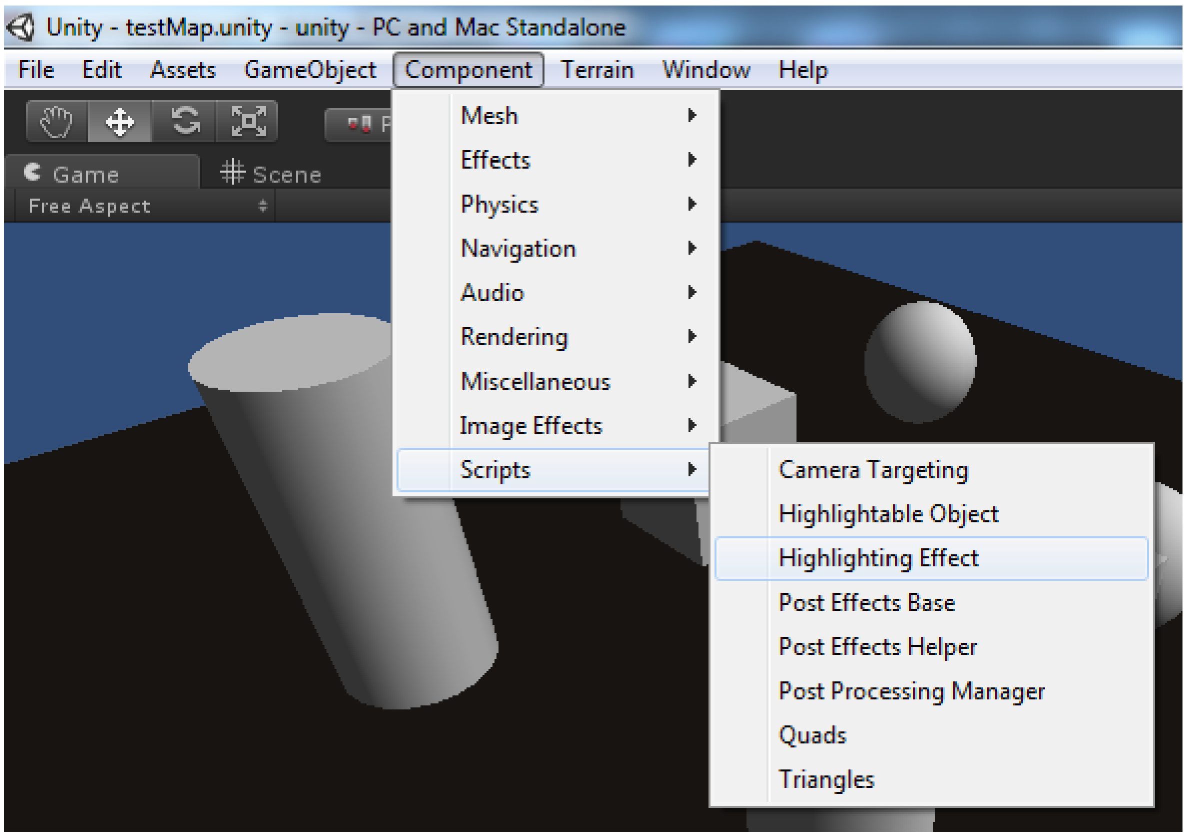
3.1 Package overview

After package installation, inside of *Plugins\HighlightingSystem* folder will be located all the scripts and replacement shaders of the HighlightingSystem. There's also *HighlightingSystemEditor.cs* custom editor script, inside of *Plugins\Editor* folder, which will help you to easily configure *HighlightingEffect* component on your camera(s).

Inside of *HighlightingSystemDemo* folder in your project, you'll have demonstration scene files. Use this as a reference to the integration in your project. You can remove this folder at anytime.

4 Integration to your project

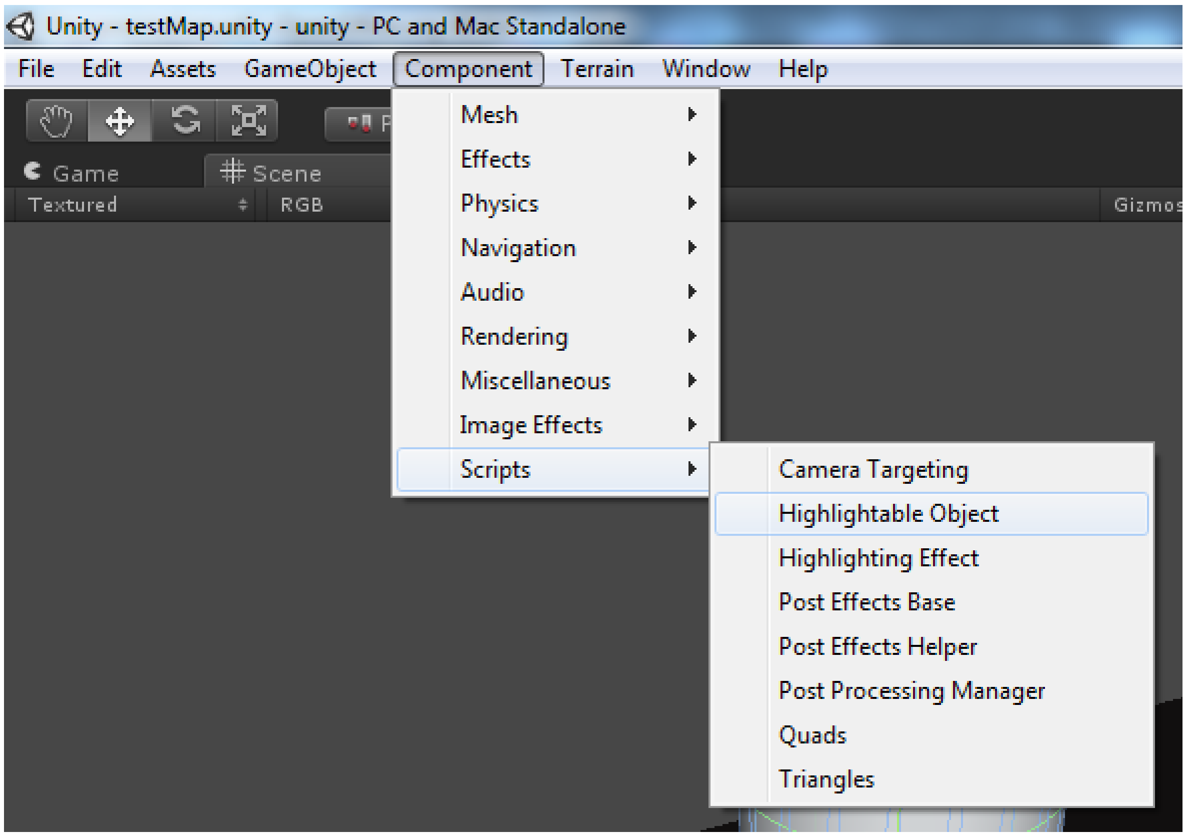
1. Import *HighlightingSystem* package from the Unity Asset Store to your project.
2. Add *HighlightingEffect.cs* component to your camera(s).



3. Configure parameters of the *HighlightingEffect* component(s). Check the *Use Z-Buffer* option if you plan to use highlighting occluders in your project. Keep it unchecked otherwise.

   [Optional] Set custom parameters for the *HighlightingEffect* quality and intensity controls, or choose any highlighting preset by clicking on its button.

4. Add *HighlightableObject.cs* component to the roots of the objects you want to make highlightable or do that in runtime with *GameObject*'s *AddComponent<HighlightableObject>()* method (see the *HighlightingSystemDemo\Scenes\Scripting* demo scene).



5. At runtime, call any highlighting methods on the *HighlightableObject*'s components.
6. When you'll be ready to release – don't forget to completely remove the *HighlightingSystemDemo* folder.

5 Important usage tips

- When using multiple cameras with the *HighlightingEffect* component applied - never use different *Use Z-Buffer* settings for them in one scene (this option must be set to the same value on each active *HighlightingEffect* component in scene). Otherwise, highlighting materials will be constantly initialized with different z-writing settings once per frame by each active *HighlightingEffect* component. This is bad for the performance.
- 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*.
- If you use custom transparent shaders in your project and want to make them properly highlightable:
    1. Make sure that the *RenderType* shader tag is set to *Transparent* or *TransparentCutout* (check http://docs.unity3d.com/Documentation/Components/SL-ShaderReplacement.html for more info). Otherwise – it will be recognized by the Highlighting System as an opaque material and will not take into account the alpha channel of your material's main texture.
    2. Make sure that your custom shader has the *_MainTex* property. Highlighting System will use texture applied to that property to detect transparent areas by comparing this texture alpha channel with the threshold value, taken from:
        - the *_Cutoff* property if your custom shader has it, or
        - the *HighlightableObject*'s internal *transparentCutoff* variable otherwise (set to 0.5 by default, you can change this in *HighlightableObject.cs*).

  Note that main texture with its offset and scale values is cached by the Highlighting System only once, when highlighting materials is initialized (at first frame rendering stage and after each *ReinitMaterials()* call). Because of that, your changes to the main texture parameters will not be reflected by the highlighting without *ReinitMaterials()* call.
- Never use completely black color for the highlighting (*Color.black* or *new Color(0,0,0,1)*). Use at least 1/255 (or 0.004) for at least one color channel to avoid visual artifacts. Completely black color is recognized by the Highlighting System as the absence of anything highlighted in stencil (highlighting) buffer.
- When configuring your *HighlightingEffect* components - increasing blur iterations will help you to improve outline glow quality, but try to keep this value as low as possible in terms of performance.
- If you use highlighting occluders in your project – keep their amount as low as possible, because occluder objects rendered in additional pass as well as all highlighted objects.

6 Methods reference

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

1. **Occluder**
   Object in this mode will become highlighting occluder. Actually, this is not a highlighting mode but it will override any other modes (has the highest priority).

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

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

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

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

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

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

- *On()*
  Turn on highlighting for this frame.

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

- *FlashingParams(Color color1, Color color2, float freq)*
  Set flashing parameters – starting/ending 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).

- *OccluderOn()*

Turn object into highlighting occluder.

- *OccluderOff()*
  Disable occluder mode.

- *OccluderSwitch()*
  Switch occluder mode.

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

- *Die()*
  Destroy the *HighlightingObject* 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).

7 Anti-aliasing

Hardware anti-aliasing doesn't affect *RenderTexture* used by the Highlighting System (this is a common problem for all post-processing effects in Unity). As a result, if you enable hardware anti-aliasing, you'll still see aliased glowing outline edges. Because of that, it is recommended to use *AntialiasingAsPostEffect.js* instead from the Unity's standard *Image Effects (Pro only)* package.

8 Support

Send your bug reports, feedback and questions to support@deepdreamgames.com