Blur Behind UI manual

v.2.1 | Online manual | Store page

1. Designation

This package is designed to blur a background under semi-transparent UI elements.

It is done by multipass gaussian blur filter, processed after the selected camera has finished its rendering. The product of this process is stored as a global texture and can be used by multiple UI images or texts in overlay canvas almost at the cost of ordinary sprite. It is great when you draw a big amount of panels and buttons on top of the main game view (eg. HUD), but it also means that one UI element will not automatically blur another by simply overlapping it (this effect is possible, though, and described in section *5. Canvas render modes*).

2. Compatibility

The effect uses image effect functionality, so the target platform should at least support render textures. It generally works on desktops, WebGL, iOS and Android.

If it's not supported, it will be automatically disabled without causing catastrophic damage.

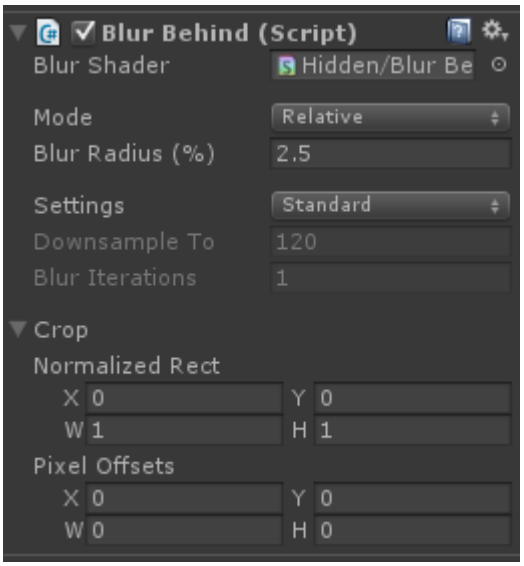
3. Setup

After importing the package you can examine an example scene from the package. It uses most of presented features.

To make this effect work from scratch you'll need to:

1. Add the *Blur Behind* component to a camera (by dragging the script file from the *Blur Behind* folder to that gameobject, or via *Component/Image Effects/Blur Behind* menu).
2. Apply *UI* material from the *Blur Behind* folder to some UI Image or Text. Alpha channel of its sprite/font will define the shape of blurred area.
3. Put your actual semi-transparent sprite or text on top of it.

4. Settings



Mode defines how the blur radius and downsampling is set up. *Absolute* mode is designed for UI with constant pixel size, and *Relative* mode is for UI that scales to fit the screen size.

Blur Radius is measured in pixels with *Absolute* mode and in screen size percents with *Relative*. Hereinafter, size means the long side of an image.

Settings field determines how the downsampling and blur iterations count will be defined. The *Manual* lets you set all values by yourself. The *Standard* calculates them automatically, and is good for constant or fast changing blur radius — it has high performance, but can produce some image vibrations when blur radius changes slowly. The *Smooth* uses a higher iterations count, which suppress most animation artifacts, but has a higher system requirements.

Downsampling reduces the input image size before blur calculations, which is good both for performance and reducing blur artifacts. The value of this property has a different meaning for different modes. In *Absolute* mode it will divide input image size *by the value*. In *Relative* mode it will reduce input image size *to the value*.

Blur Iterations is the count of blur calculation passes. The higher value increases the overall quality greatly, but has a heavy performance impact.

Crop section lets you define, which part of the camera view will be actually blurred and stored. If your UI covers only a small part of the screen, limiting the blur processing to that area can greatly improve performance of the effect.

Normalized Rect sets the crop frame in relative scale, pretty much like the camera Viewport Rect property does. *Pixel Offsets* adds up to the *Normalized Rect*, but in absolute scale.

Crop setting examples:

- Effect with *Normalized Rect* set to (X=0.5, Y=0, W=0.5, H=0.5) and all *Pixel Offsets* set to 0 will process the bottom-right quarter of the input image.
- Effect with *Normalized Rect* set to (X=0, Y=1, W=1, H=0) and Pixel Offsets set to (X=0, Y=-50, W=0, H=50) will process only the top 50 pixels of the input image.

Keep in mind, that for the best result the crop frame should extend over UI by the blur radius.

5. Canvas render modes

Screen Space – Camera

This package is primarily designed to work with *“Screen Space – Overlay”* render mode.

However, if you understand the rendering order enough, you can achieve additional effects by using *“Screen Space – Camera”* render mode with multiple cameras. Eg. blurring one UI layer by another *(see the example scene)*.

The thing is, the *Blur Behind* component on camera blurs all content the camera has rendered, with all image effects placed above the *Blur Behind* component in gameobject inspector. It also includes all UI canvases with *“Screen Space – Camera”* mode, which have this camera as target.

Therefore, to create a multilayer blur effect you will need the following:

1. First camera (lowest depth property) with a *Blur Behind* component. To render the scene and save it blurred version into a texture.
2. Canvas with *“Screen Space – Camera”* mode, containing UI layer.
3. Additional camera, targeted by that canvas, with clear flags set to “Depth Only”, culling mask to UI-only, and a *Blur Behind* component attached. To render the UI layer with blurred content of previous camera and replace the stored texture with new blurred fullscreen image for later use. Repeat #2 and #3 for all additional layers.
4. Final canvas with *“Screen Space – Overlay”* mode, which contains all UI elements that should not be blurred by anything on top of it.

If you are not sure about rendering order you've got, use the built-in Unity *Frame Debugger* tool to inspect the whole rendering process, step by step. You can see there, when exactly the blurred image is updated and what it contains.

World Space

To properly render the blurred UI in the world space you'll also need the multicamera setup (even for a single-layer UI):

1. First camera with the culling mask set to non-UI, and a *Blur Behind* component. To render the scene without UI and save it blurred version into a texture.
2. Additional camera, with identical transform and settings (field of view and clipping planes are crucial) to the first one, but with "Don't Clear" clear flags, and UI-only culling mask. To render the UI with blurred content of previous camera, taking into account the depth buffer of the scene (occlude UI with geometry).

Non-fullscreen viewport

If the camera that renders a canvas has a non-fullscreen viewport and has no *Blur Behind* component, you should attach a *Blur Behind Viewport* component (from the *Blur Behind/Scripts* folder) to that camera. It will set up a correct screen coordinates for UI shader and will reset them back afterward for overlay canvases.

6. Troubleshooting

- If you use a perspective projection to render the UI and rotate some blurred UI element, you'll see the distortion of a background image. This issue will be fixed in the next update. Until then you can replace *Blur Behind/Shaders/UI.shader* with this.
- If instead of desired blurred image you see just one color or some flickering, check if the crop setting of the Blur Behind component dropped to zeros due scene loading or version control failure.

7. Upgrade from v.1

If you used the first version of *Blur Behind UI* and want to update it in your project, be aware, that some things will not be compatible.

Firstly, you will have to replace all *Blur Behind* components with the new ones, since their settings are very different now.

Secondly, there is no *Text* material anymore. The *UI* material do both jobs now. To preserve the references you may keep old materials and just set the new shaders for them.

8. Contacts

If you have any problems or suggestions, please contact me at a.a.saraev@yandex.ru.

Or you can leave a comment right in this online document, so I can publicly answer your question and improve the manual.

Also, it would be great, if you rate the asset or even write a review on the store page.

And let me know if your product goes live with my effect in it.