UI Toolkit Blurred Background

Table of contents

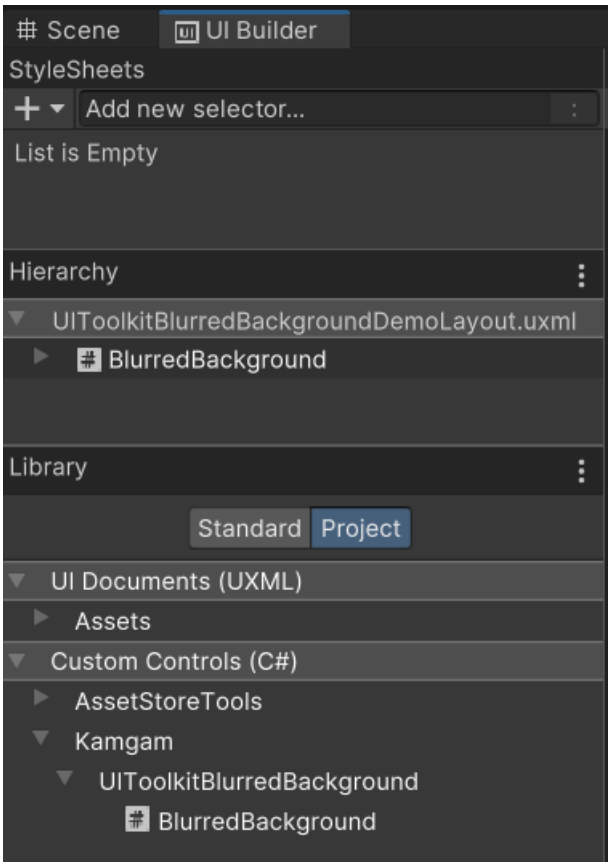
Requirements

Unity 2021.2 or higher is required since that is when Unity added the UI Toolkit Module for runtime use. If you can, please upgrade to the highest LTS version of Unity. The newer the version the less „glitches" the UI Toolkit has.

Keep in mind, UI Toolkit as a whole is still a work in progress and not quite ready for prime time. Unity itself still recommends using UGUI instead of UI Toolkit for runtime applications (source).

Creating a blurred background

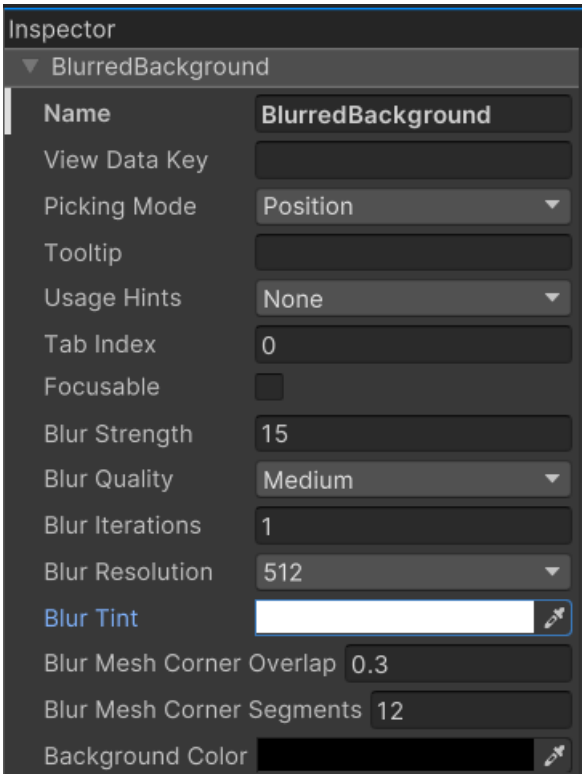
In the **UI Builder** window you can find the blurred background element under **Library > Project > Custom Controls > Kamgam > UIToolkitBurredBackground > Blurred Background**



Settings

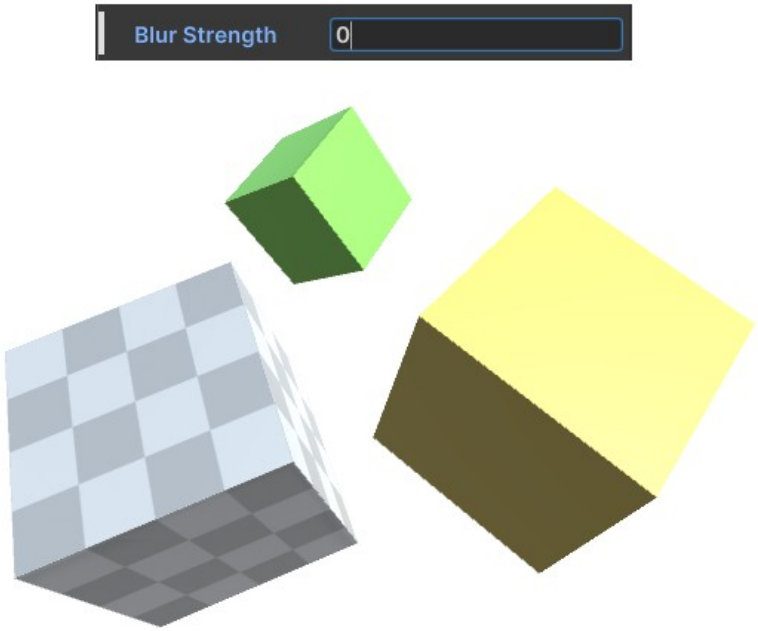
The blurred background can be controlled via the inspector in the UI Builder.

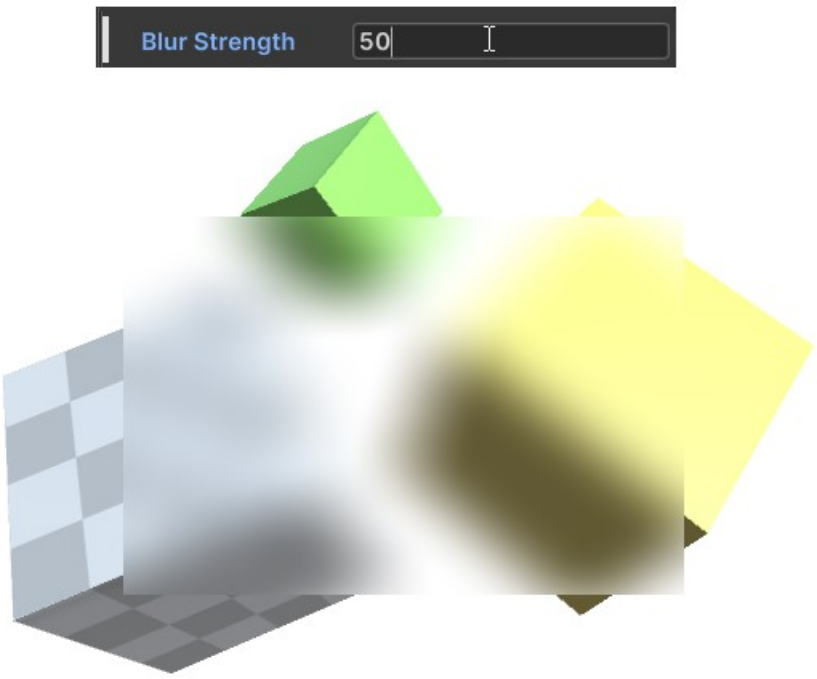
⚠ At the moment there is only **one global blur setting**. This means that all blurred backgrounds will use the same blur settings. - If you need more please let me know. If demand is high enough it will be added.



Blur Strength

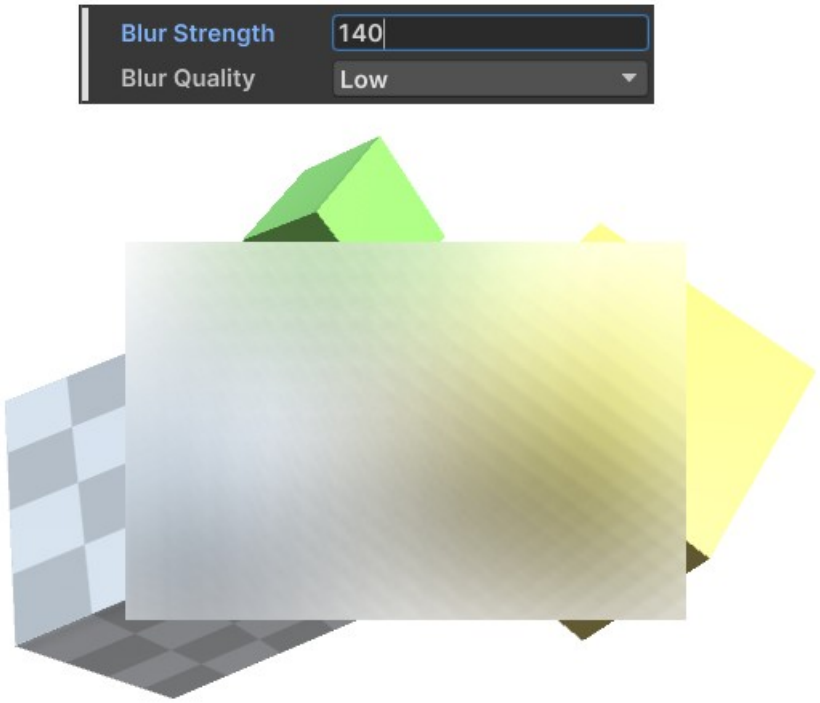
Defines how strong the blur effect will be. Notice that the quality may degrade more the higher the strength is (more on that below).

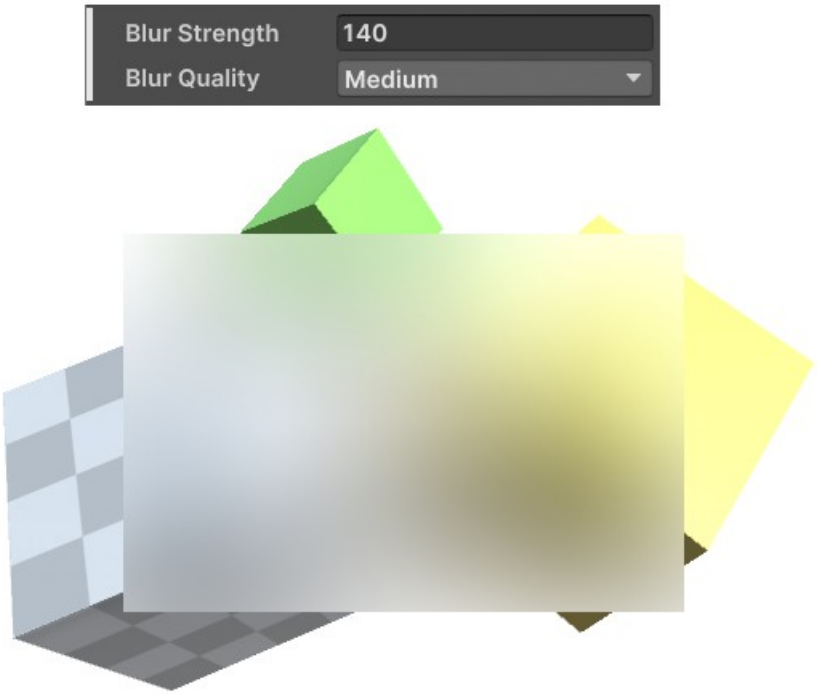



Blur Strength
50

Blur Quality

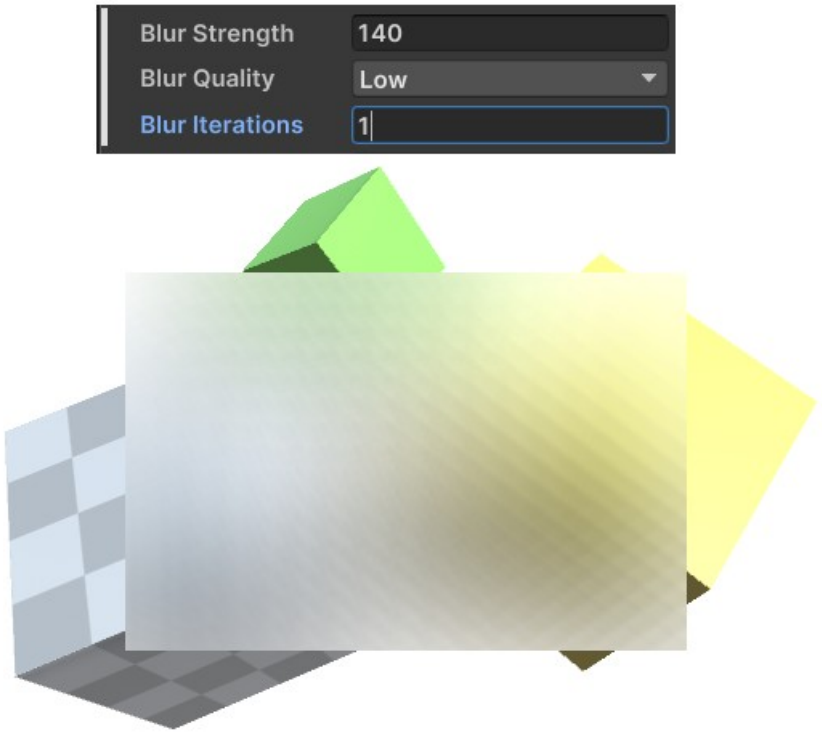
If high blur strengths are used then you may notice visible artefacts. To avoid these increase the quality. NOTICE: The higher the quality the more performance it will cost. As a rule of thumb (low = a cost of 1, medium = a cost of 3, high = a cost of 10).

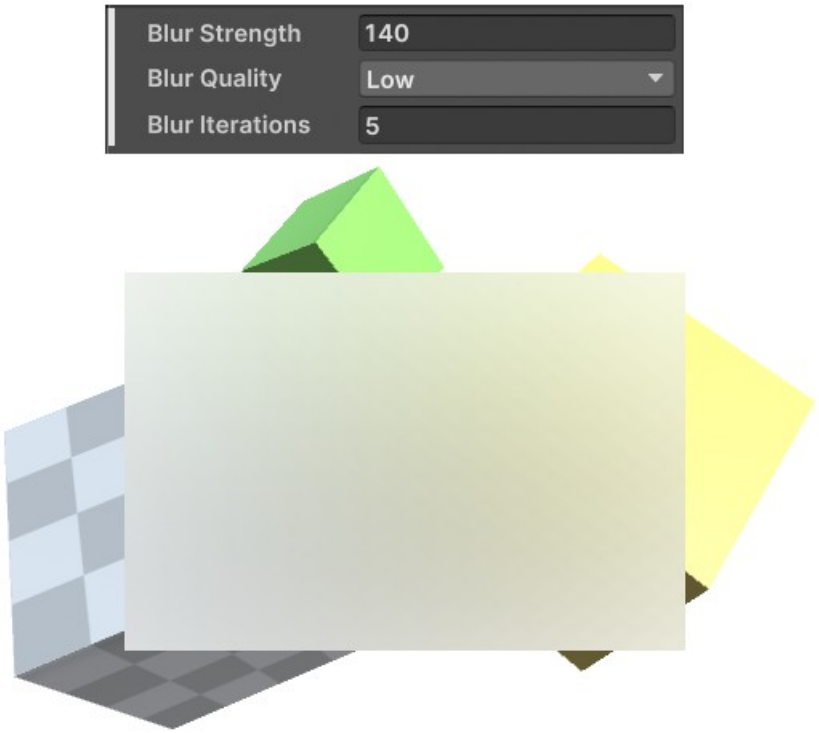




Blur Iterations

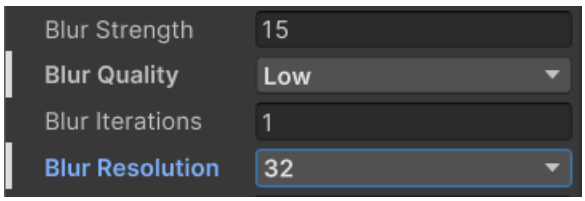
Blur iterations should be kept at 1. This defines how often the blur filter will be applied. In terms of performance this the most expensive setting you can increase. Use with care (avoid if you can).





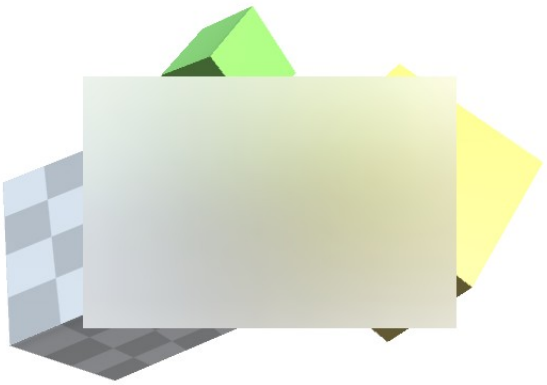
Blur Resolution

Reducing the resolution is a great way to increase the blurryness of your image while also saving a LOT of performance. Halfing the resolution usually makes the blur 4 times faster.

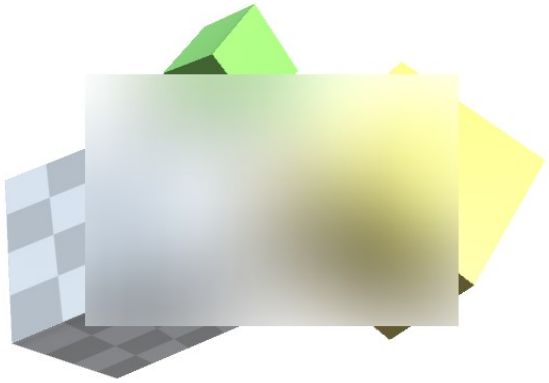


Using a low resolution can also allow you to get away with the quality set to low.

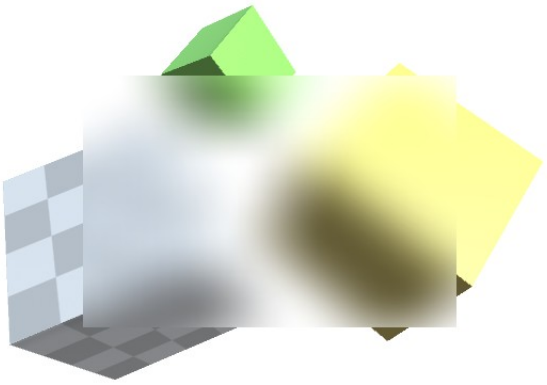
32 x 32



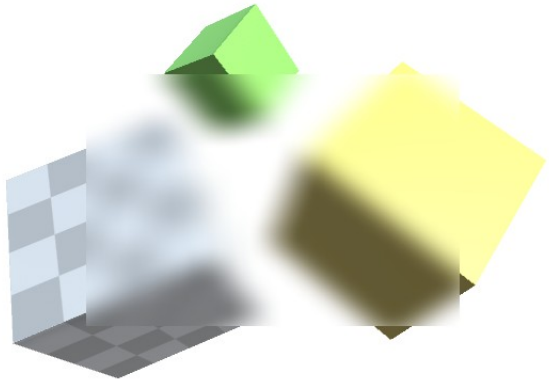
64 x 64



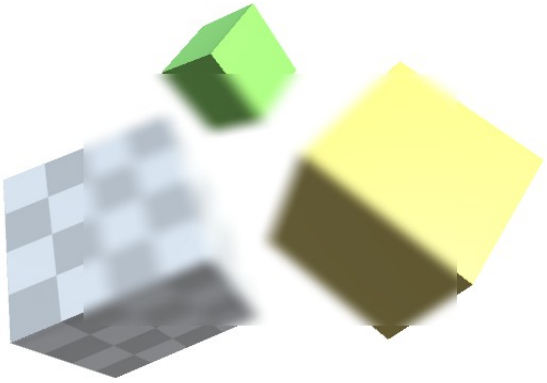
128 x 128



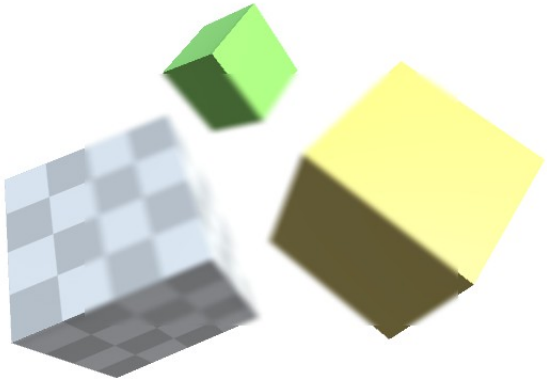
256 x 256



512 x 512

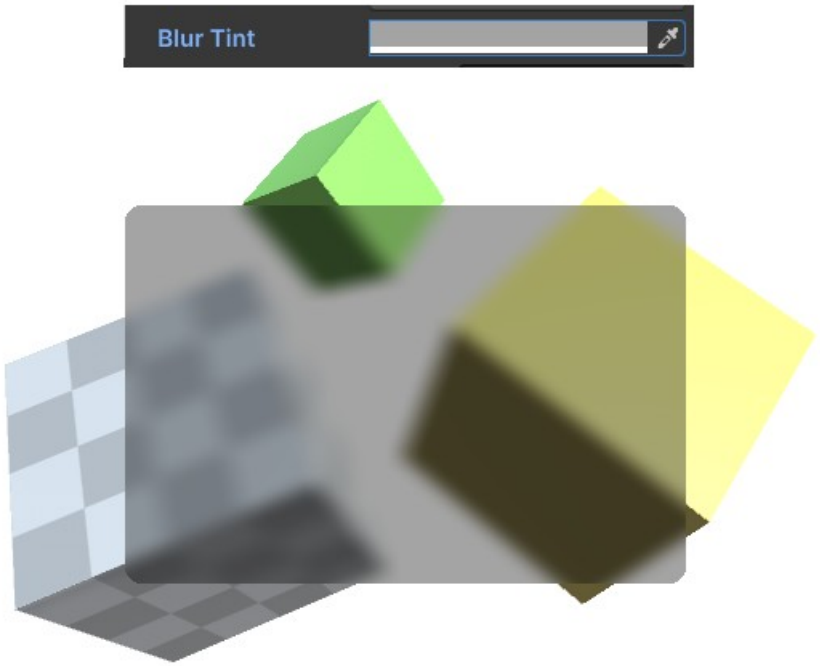


1024 x 1024

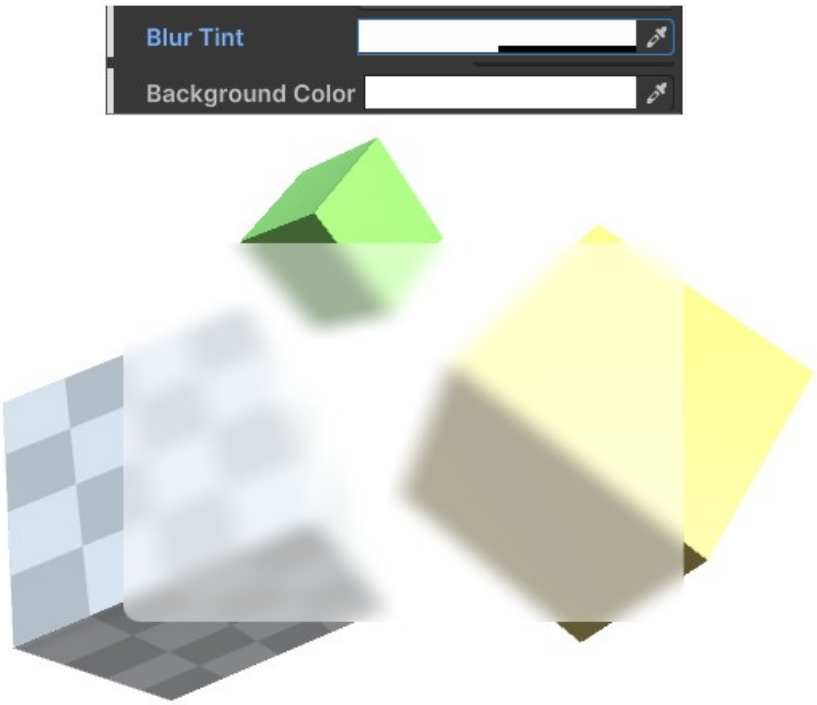


Blur Tint

By tinting the color you can make your blur look like an overlay.



You may have noticed that tinting it white will not make it more white. To achieve a milky glass like overlay you have to reduce the ALPHA of the tint and use a white, opaque color as the background. Like this:

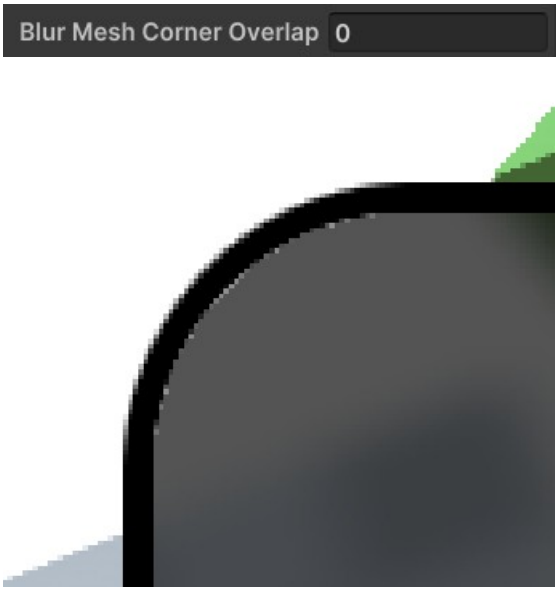


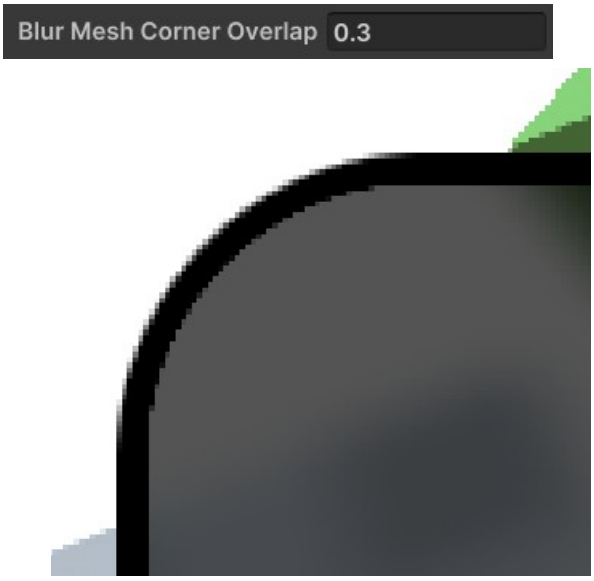
Mesh Corner Overlap

Sadly we can not (yet) modify the mesh of a visual element in UI Toolkit. That's why the blurred background has to be drawn OVER the normal mesh.

This means it has to draw a new mesh matching the borders an curves of the underlying mesh. The result of this is that sometimes there are gaps between the blurred overlay mesh and the normal border.

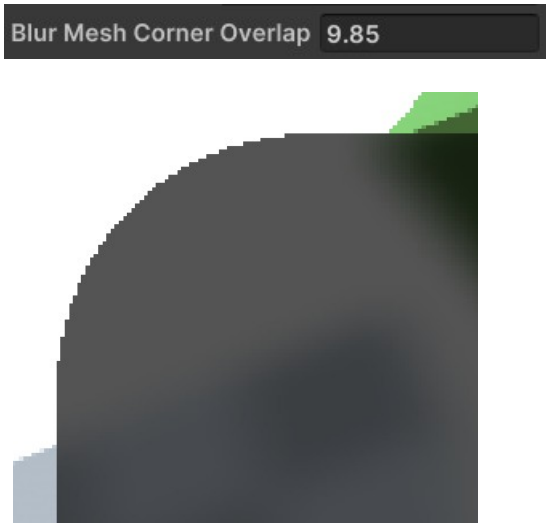
Like this (zoomed in example)





The corner OVERLAP defines how much further out the blurred mesh should be drawn in order to cover up these gaps. Usually a value of about 0.2 or 0.3 is fine.

To illustrate the process here is an image with a ridiculous overlap of about 10. Notice how all the borders are gone.



You may have also noticed that the overlay mesh edges are not anti-aliased. Sadly that's a limitation by the UI Toolkit. Hopefully they will add it in future releases.

Mesh Corner Segments

Since the blurred mesh is a new mesh on top of the default mesh it has to approximate the borders. This setting defines how detailed the rounded corners are approximated.

For illustration purposes here is a rounded corner with various segmentations:

1 2 3 4

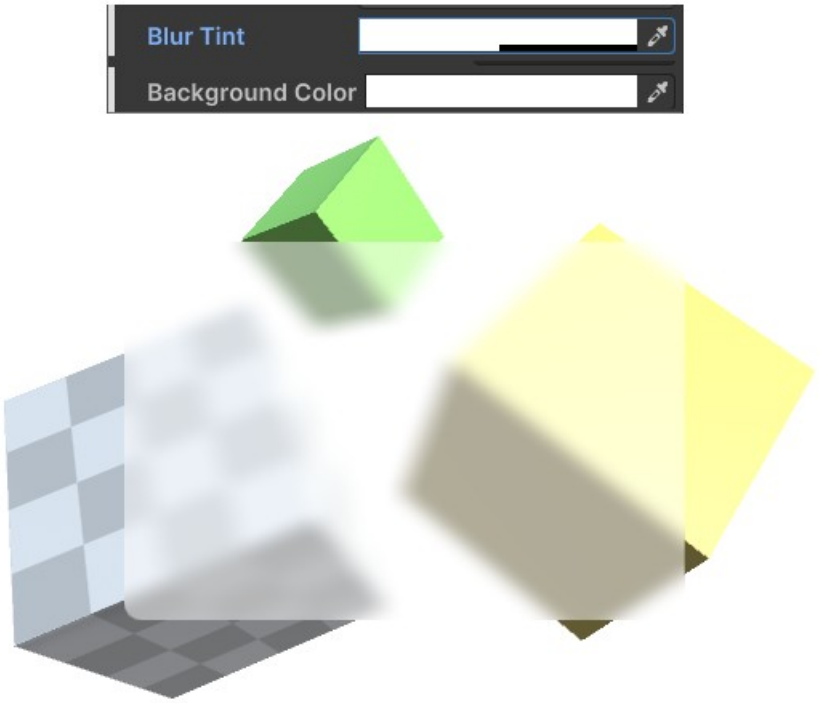
Background Color

This actually is just a wrapper for the normal background color. By default it is set to a fully transparent black color.

If you want to achieve a milky, glass like overlay you will have to reduce the alpha on the tint and use a white none transparent background color.

The blurred background image actually is an opaque image drawn on top of the normal background.

Milky glass:



USS Styles

Many properties can be controlled by USS styles too:

--blur-strength (float)

--blur-iterations (int)

--blur-quality (string, takes values „low“, „medium“ or „high“)

NOTICE: The quality values are like keywords not strings:

```
.test-styles {
    --blur-iterations: 1;
    --blur-strength: 100;
    --blur-quality: low; /* <- this works */
    --blur-quality: "high";  /* <- this does not. Please do not add any ". */
}
```

--blur-resolution (int, powers of 2 from 32 up to 2048)

--blur-tint (color)

--blur-mesh-corner-overlap (float)

--blur-mesh-corner-segments (int)

--blur-background-color (color)

Frequently Asked Questions

Here are some common issues that have been reported.

If you can, please upgrade to the highest LTS version of Unity. The newer the version the less „glitches" the UI Toolkit has.

Keep in mind, UI Toolkit as a whole it is still a work in progress and not quite ready for prime time. Unity itself still recommends using UGUI instead of UI Toolkit for runtime applications (source).

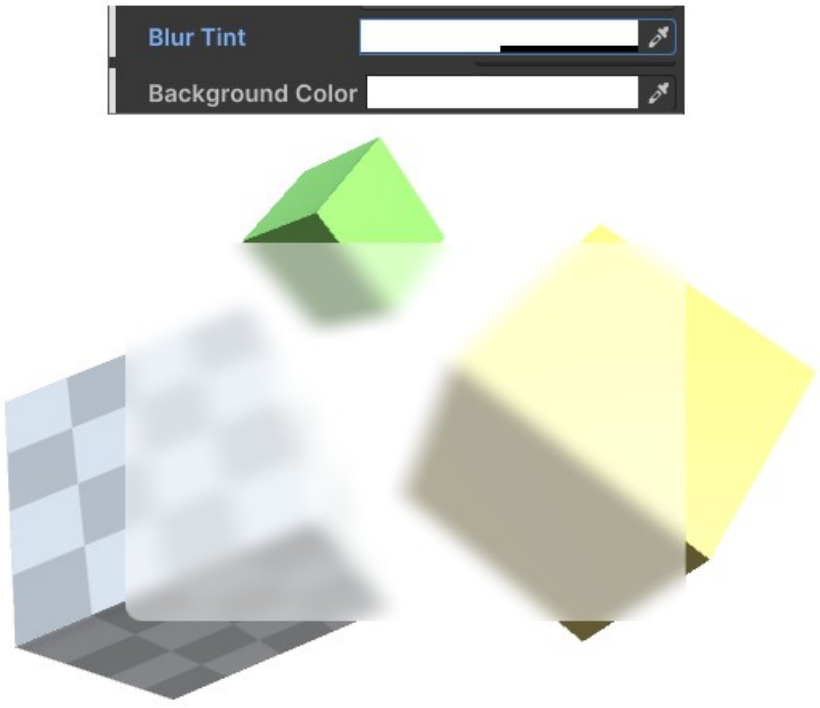
How do I make a white tint?

You will have to use the background color in combination with a tint.

Like this:

If you want to achieve a milky, glass like overlay you will have to reduce the alpha on the tint and use a white none transparent background color. The blurred background image actually is an opaque image drawn on top of the normal background.

Milky glass:



All the blurred UIs have the same blur settings applied.

Yes, that is done on purpose to save performance. The blur effect is generated by extracting the rendered image and blurring it. To support multiple blur factors at once it would have to be done multiple times, which means double or tripple the cost (basically one for each settings combination). This could escalate quickly and reduce the performance quite a bit. Thus it is not (yet) supported.

If you need multiple different blurs within one UI then please contact support. It will be added if demand is high enough.

The blur does not update or align properly in the game view.

It sometimes takes time for Unity to pick up on changes in the game view if not playing. However, rest assured it will work at runtime and in builds. To force an update change any of the blur settings on the element. While that's not a 100% fix it works most of the time.

The blur does not „work" in the UI Builder

That's because there is no background scene in the UI builder as reference. It will simply show the blurred background of the scene, not the background of the builder.

If „Blur Strength" or „Blur Iterations" is set to 0 then the background image vanishes.

Having either of these set to 0 would result in no blur at all and thus we do not even generate the mesh for the background override. That is done to save performance.

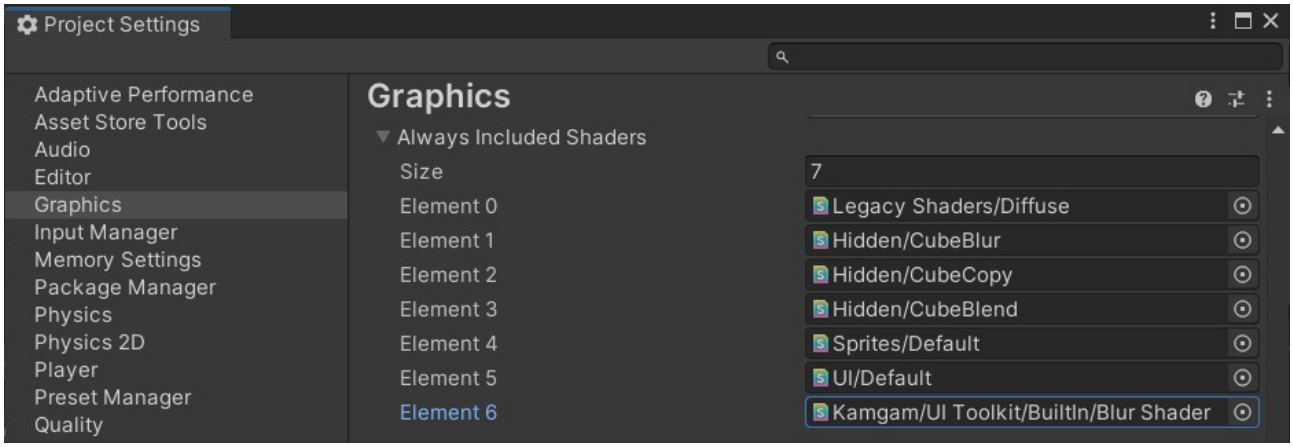
UI over another UI does not show the UI behind.

Yes, that's on purpose (it's how it works). Only scene contents are blurred and shown, sorry.

It works in the editor but not in build

Maybe you have reset the „Always included shaders". Since the blur effect is a graphics effect it is run on the graphics card and requires a shader.

Usually the shaders are added to the „Always included" list at the start. Please check if they are added and if they are using the correct render pipeline (BuiltIn, URP or HDRP).



If I use the „transform“ to set the position then the blur does not move

Yes, sadly moving UI Toolkit elements with **transforms** is not automatically detected by UI Toolkit.

Please call .MarkDirtyRepaint() on the blurred background element after moving it with transforms. This ensures the blur position is properly updated.

HINT: Please consider moving the element with styles (left, top, margins, ..). Style changes are picked up automatically and do not require you to trigger the update manually.

I don’t like the forced square resoltion of the blur. Can I change it to match my screen exactly?

The square resolution was done for performance reasons. I know it's not ideal for all aspect ratios (gives „artefacts“ for low blur values).

You change the resoltuion dynamically via scripts, like this:

```
using UnityEngine;

namespace Kamgam.UIToolkitBlurredBackground
{
    [ExecuteInEditMode]
    public class ChangeBlurResolution : MonoBehaviour
    {
        public bool ChangeResolution = false;
        public Vector2Int Resolution = new Vector2Int(512, 256);

        void Update()
        {
            var mgr = BlurManager.Instance;
            if (mgr != null)
                mgr.Resolution = Resolution;
        }
    }
}
```

The 512 x 256 is just some demo values. You could derive them from your current screen resolution.