INTERACTIVE & CROSS-PLATFORM DEVELOPMENT STUDIO

UNITY 2D FILTERS

UNITY 2D FILTERS

You're trying to create fancy effects for your **Sprite 2D** but don't know where to start?

You heard about **shaders** but that's some dark magic to you?

Why is it so hard to make a blur **effect** or have simple **color effects** at runtime?

The ***UNITY 2D FILTERS*** package is here to solve these problems for you!

UNITY 2D FILTERS

Multiple filters are supported using shaders :

You can combine several effects at runtime. They are all **parametrable**!

Outline

Hue

Tint

Solid color

Contrast

User defined color matrix

Blur

Invert

Saturation

Pixelate

Brightness

HOW TO USE ?

Using Unity 2D Filters is quite easy. When your sprite is in the scene, select it and click on the **`Component`** menu, select Filters 2D and the filter of your choice :

- **`Blur`** for the … blur effect!
- **`Outline`** for the outline effect
- **`Pixelate`** for a nice retro game style!
- **`ColorMatrix`** to tweak the colors of your sprite

A script to adjust the parameters of the filter will be added and the original material of your Sprite will be replaced by another. Don't worry, removing the script will set your material back! ☻

Note : ColorMatrix & Pixelate & Blur are the only filters compatible with canvas elements yet. You can't combine two different filters listed above at the same time.

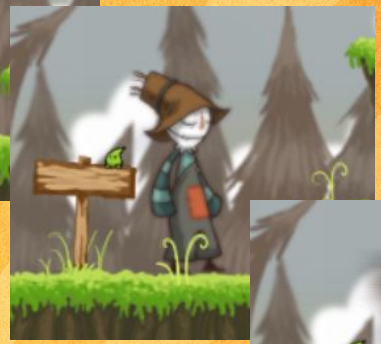
BLUR EFFECT

Nothing difficult for this effect, you only have one value to tweak, the **`Intensity`**!

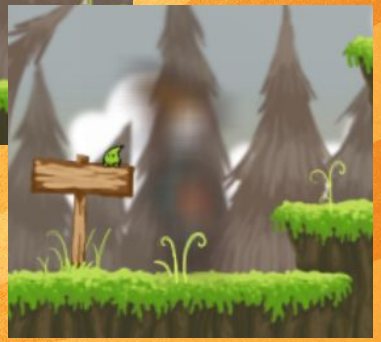
```
// C#
Blur blur = GetComponent<Blur>() ??
     gameObject.AddComponent<Blur>();
// ...
blur.Intensity  = (Mathf.Sin(Time.time)+1)*5;
```

You can also enable mip maps on your Sprite and change the LOD value of the component if you want a smoother blur.

Intensity = 0



Intensity = 2



Intensity = 10

Note : Using large intensity values may not give the expected result.

Caution : Sprite in atlas generated by sprite packer in tight mode will not work or not correctly.

Caution : On small sprite in atlas may appears artefacts. Please read this.

OUTLINE EFFECT

The outline effect adds a thin colored line all around your sprite.

You can change the color of the outline, its thickness and even shift the outline. If you want, you can have only the outline or the outline and the sprite itself.

```
// C#
Outline outline = GetComponent<Outline>() ??
     gameObject.AddComponent<Outline>();
// ...
void OnMouseEnter(){outline.Color=Color.red;}
void OnMouseExit(){outline.Color=new Color(0,0,0,0);}
```

No outline

Red outline on mouse over

No fill

Caution : Sprite in atlas generated by sprite packer in tight mode will not work or not correctly.

Caution : On small sprite in atlas may appears artefacts. Please read this.

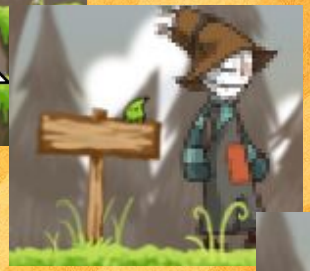
PIXELATE EFFECT

The pixelate effect adds a nice retro pixelated effect to your sprite.

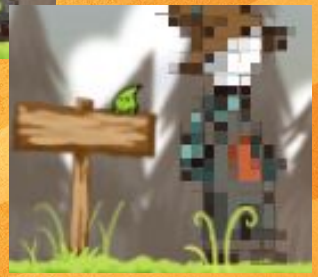
You can change the pixel size of the pixelated effect.

```
// C#
Pixelate pixelate = GetComponent<Pixelate>() ??
     gameObject.AddComponent<Pixelate>();
// ...
void OnMouseEnter(){pixelate.PixelSize=new Vector(3,
3);}
void OnMouseExit(){pixelate.PixelSize=new
Vector(0,0);}
```

No pixelated



PixelSize 2, 2



PixelSize 5, 5

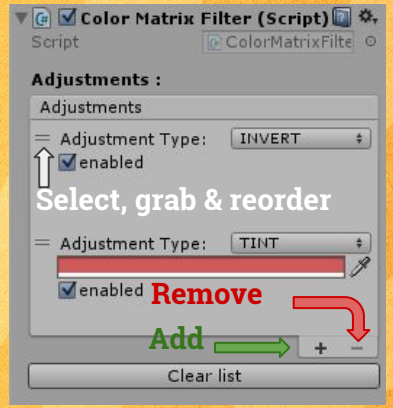
Caution : Sprite in atlas generated by sprite packer in tight mode will not work or not correctly.

Caution : On small sprite in atlas may appears artefacts. Please read this.

COLOR MATRIX - MANAGE ADJUSTMENTS

With this filter, you can change many color parameters of your Sprite, and even of the whole scene by adding the component to the Camera!

You can add as many adjustment as you want ! But keep in mind that the order of the adjusments is important.



Invert after tint

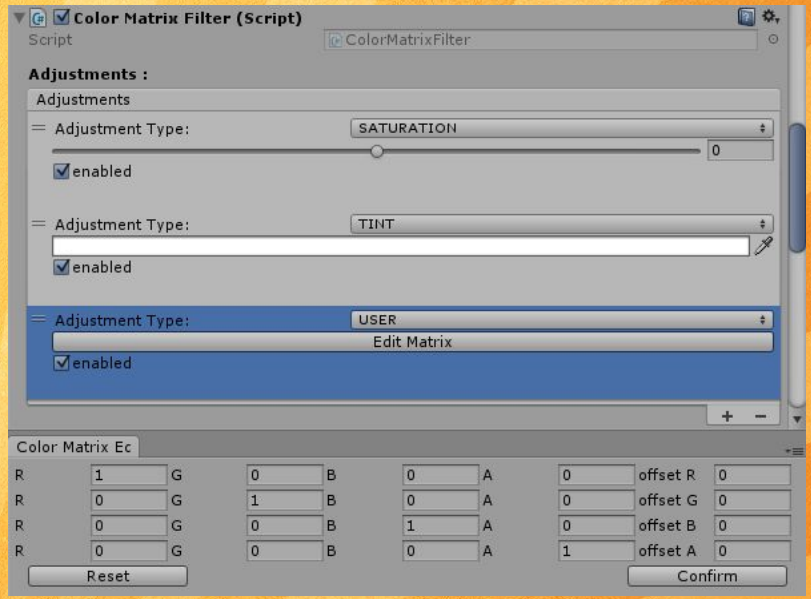
Invert before tint

COLOR MATRIX - TWEAK THE COLORS

Basic parameters (brightness, contrast, saturation, ...) can be changed using the slider of the adjustment, or using the floating value.

Other parameters (tint, solid color) use a Color picker field.

If you are accustomed to color matrices, you can even define your own matrix. Select **`User`** in the adjustment type, and click on the **`Edit Matrix`** button.



COLOR MATRIX - TWEAK THE COLORS USING CODE

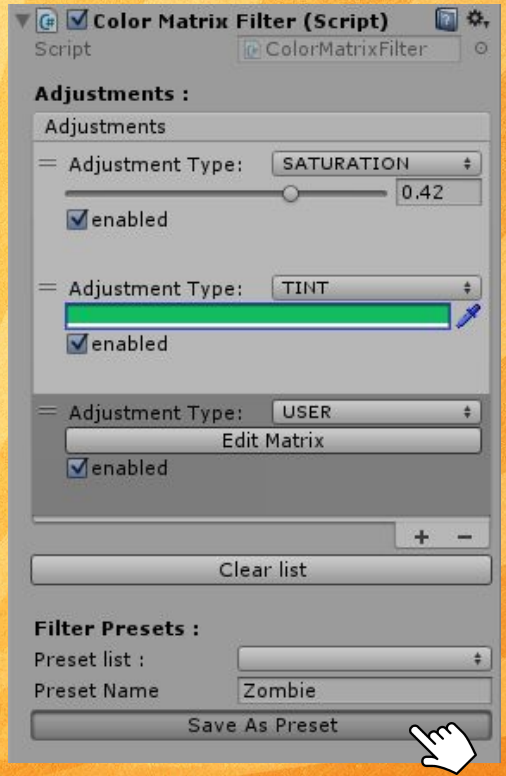
```
// C#
ColorMatrix colorMatrix = gameObject.GetComponent<ColorMatrix>() ??
  gameObject.AddComponent<ColorMatrix>() ;
ColorMatrixAdjustment hue = new ColorMatrixAdjustment();
hue.type       = ColorMatrixAdjustmentType.HUE;
hue.floatValue = UnityEngine.Random.value * 2 - 1;
colorMatrix.adjustments.Add(hue);

ColorMatrixAdjustment negative = new ColorMatrixAdjustment();
ColorMatrix4x5 matrix = new ColorMatrix4x5();
matrix[0, 0]  = matrix[1, 1] = matrix[2, 2] = -1;
matrix[0, 4]  = matrix[1, 4] = matrix[2, 4] = 1;
negative.type = ColorMatrixAdjustmentType.USER;
negative.colorMatrix = matrix;
colorMatrix.adjustments.Add( negative );
```

COLOR MATRIX - PRESETS

You love the new effect you gave to your Sprite? Save it using presets!

Under the list of adjustments, you will find some presets already configured for the most common filters : Black and White, Sepia, and even Neon!

You can also save your own list of adjustment into a preset. Give it a name and hit **`Save As Preset`**!



COLOR MATRIX - RESULTS

FILTER2D IN UI

SHADER STENCIL BUFFER

The stencil buffer in shaders used to display or not a texture pixel. It's more or less a mask function apply after the z-buffer.

It's composed of a number assigned to the shader, a number store in a buffer readable by all shaders, and a mathematical compare function.
The numbers are between 0-255 and the buffer store a number for each screen pixel like an image.

The program determine on which screen pixel the current pixel texture will be render, and take the stencil buffer number of this screen pixel.
The shader stencil number is compared with the stencil number taken previously. If the mathematical compare is exact (or true), the pixel is display, else it is not.

After that, the stencil buffer number could be modified and gave to the next shader.

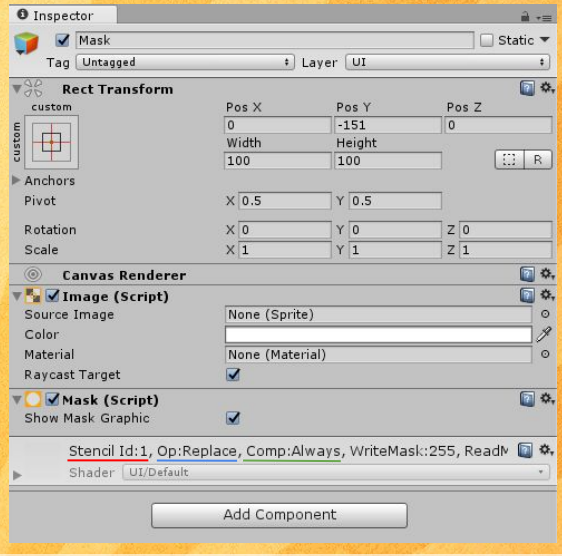
More information about this in the unity documentation :
https://docs.unity3d.com/Manual/SL-Stencil.html

SHADER STENCIL BUFFER AND UI MASK

The Unity UI mask use the stencil buffer to disimulate some UI elements.

When you set a mask, Unity will used the image attach in the gameobject to decide, for each screen pixel, if the shader stencil number will be store in the buffer.

The next UI element have just to know the stencil number of the mask, and compare if this number is equal with the number stored in the buffer. And job done.



Keys :

- The stencil shader number
- Which operation is done on the stencil buffer. Here, we replace the current stencil buffer number by the stencil shader number
- The mathematical compare function. Here, the comparison is always exact (true).

SHADER STENCIL BUFFER, UI MASK AND FILTERS 2D

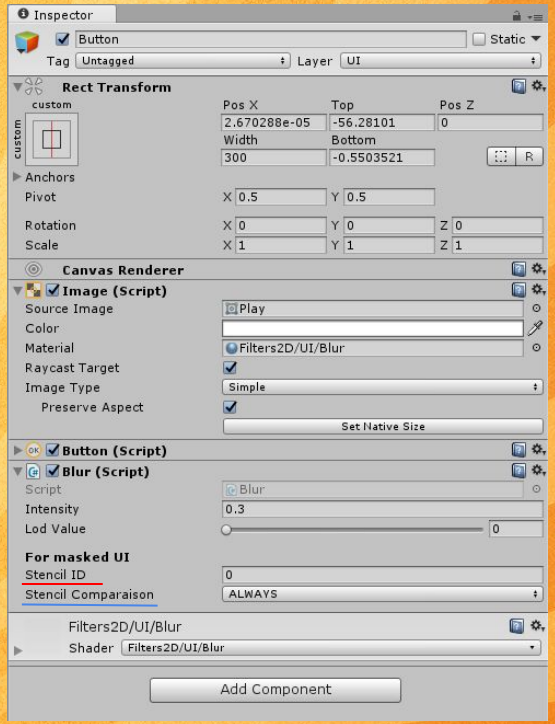
In Filters 2D script, we have access on the 2 main elements of the stencil buffer :

- the stencil shader number
- the mathematical compare function

The mathematical compare function have multiple values, select the function corresponding of the effect what you want.

Informations of the values and their use in the “Comparison Function” section.
https://docs.unity3d.com/Manual/SL-Stencil.html

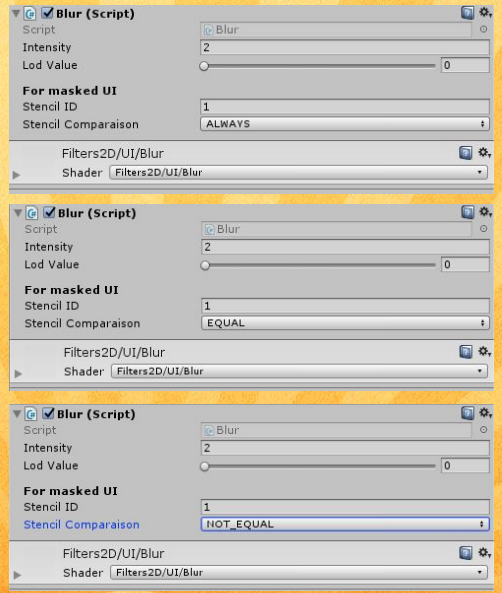
To use with the Unity UI mask, you have to take the stencil shader number (see in the previous slide) of the mask, and put in the “Stencil ID” field. Then select the “Equal” value in the “Stencil Comparaison”. And tadam ! It’s works.



SHADER STENCIL BUFFER, UI MASK AND FILTER 2D - RESULTS

The mask

The UI element + Filter 2D



Results

WARNING AND CAUTIONS

MOBILE GPU FLOAT PRECISION

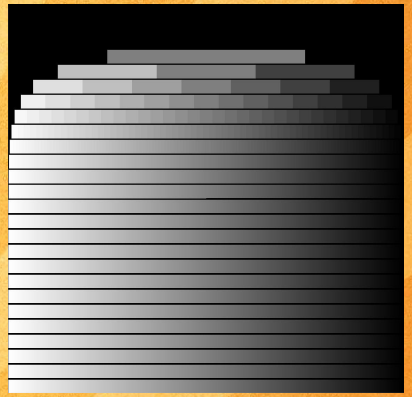
In the current smartphone market, smartphone GPUs may be less accurate than computer GPUs.

Some of ours filters could be affected by the decimal float precision delivered by GPUs.

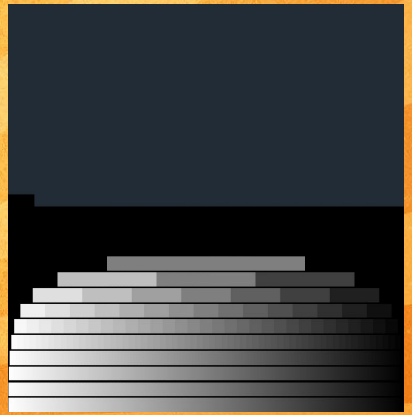
To know the GPU float precision on device, check the unity page on mobile GPU : https://docs.unity3d.com/Manual/SL-DataTypesAndPrecision.html, or use our float precision test scene : /Filters2D/Shaders tests/GPU Float decimal precision based on Tom Olson's works.
https://community.arm.com/graphics/b/blog/posts/benchmarking-floating-point-precision-in-mobile-gpus

The number of grey line is the number of decimal bit precision

Float precision from a Geforce 1070 (23 bits)



Float precision from a Mali-450 (10 bits)



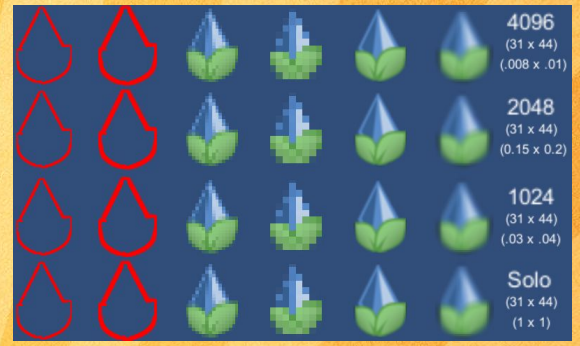
ATLAS AND ARTEFACTS

When a filter is assigned to a sprite in a large atlas, some artefacts may appear.

Cause, in GPU, pixel position are between [0-1], a small sprite in a large atlas may reach or exceed the float precision.
In this case the min step give by the float is larger than the size of a pixel.

The solution is to reduce the size of the atlas or to increase sprite sizes in the atlas and rescale the sprite in Unity.

A test scene is available to provide a test bench on your device. The scene is :
/Filters2D/Shaders tests/Outline - Pixelate - Blur test



Test scene on a Geforce 1070

Test scene on a Mali-450

ANY PROBLEM?

OPEN AN ISSUE ON THE BITBUCKET!

https://bitbucket.org/DaVikingCode/unity-2d-filters/issues/