More Post-Processing Effects

User Guide

This user guide will help you to quickly add effects to cameras and learn how to manage them. These effects are only operating with **Unity 5 or higher**.

1 Introduction

Basically, the process is simple to understand. The *post* term refers to the fact that the image processing is performed after the game drawn the scene. We can easily see the process in this order :

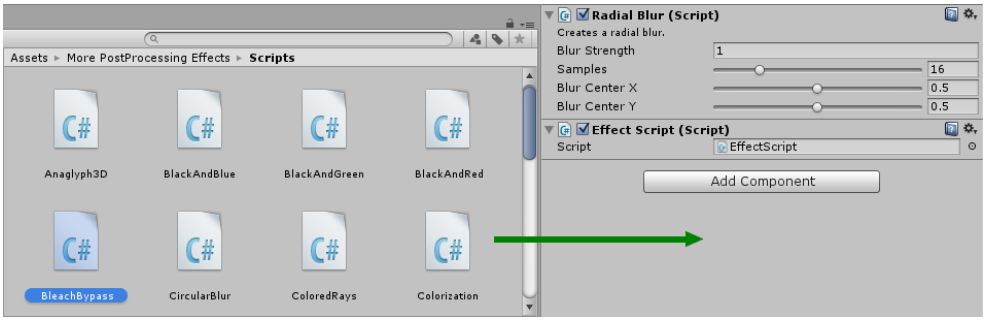
1. The engine will draw the scene and render it to the camera.
2. Then, post-processing effects will be applied on the rendered image.
3. Finally, UI objects are displayed over the final rendering.

The step that interest us is the second. Note that you can add as much effects as you want to your camera (but you must keep in mind that the more effects the camera has, the more resources the game will need). Before starting, I advise you to take a look on the *Manual* to have a brief overview of the effects and their parameters.

2 Scripting example

2.1 Add an effect to a camera

We will begin with a simple static camera. All the effects are located in the *Scripts* folder (or you can find them in the menu *Image Effects* → *More Effects*). Then you simply just have to drag and drop the effect you want to your object.



2.2 First example : Set radial blur center with mouse cursor

In this example, we will see how to use scripting to interact with the effects. We will create a **Radial Blur** (see the Manual for more details) and our goal is to define the blur center at the position of the mouse cursor. Let us create a script that we will call *EffectScript* that inherits from *MonoBehaviour* class.

```
public class EffectScript : MonoBehaviour {
        // Use this for initialization
        void Start () {
        }

        // Update is called once per frame
        void Update () {
        }
}
```

Listing 1: First example

First, we need the object to have a *RadialBlur* component, so we need to call the *RequireComponent* function just before the class definition :

```
[RequireComponent (typeof(RadialBlur))]
```

Listing 2: Don't forget to import the namespace *MorePPEffects*

Now, we will create a variable named *effect* which contains our **Radial Blur** effect. Because we required the effect, it will be automatically added to the object and it is possible to get the effect with the *GetComponent* method without incurring an exception :

```
private RadialBlur effect;

void Start () {
        effect = GetComponent<RadialBlur>();
}
```

Listing 3: The initialization

The final step is to define the center of the radial blur. On the manual, we can see that the center is defined by two variables : *centerX* and *centerY*. Because the values have to be between 0 and 1, we must divide the mouse position by the screen width or height to get values lower or equal to 1 :

```
void Update () {
        effect.centerX = Input.mousePosition.x / Screen.width;
        effect.centerY = Input.mousePosition.y / Screen.height;
}
```

Listing 4: The update loop

And it is done ! The center of the **Radial Blur** effect is now the position of the mouse cursor. This example can be adapted to any effect.