Brightness Custom Effect (Nokia Imaging SDK)

This article explains how to create a Brightness Effect similar to the BrightnessFilter in the Nokia Imaging SDK. It is provided for educational purposes rather than developer re-use.

Introduction

The *BrightnessEffect* adjusts the brightness of an image up or down very similar to the Nokia Imaging SDK's inbuilt BrightnessFilter. The main differences are that it is implemented as a custom effect and it is less efficient.

Warning: The *BrightnessEffect* should not be used in place of the inbuilt BrightnessFilter. It is provided for educational purposes - to make it easier to understand the techniques used so that more complex filters can be created.

The article provides links to the filter source code and test code, an explanation of how the filter works, and code snippets showing how it is used.

Original

-0.50 Brightness

+0.50 Brightness

Source code

Full source code for the `BrightnessEffect` custom effect is provided below (toggle "Expand") and also in a ready-to-use file here: BrightnessEffect.cs.

BrightnessEffect.cs (03 Feb 2014)

```
// ==========================================================================
// DATE        AUTHOR                    DESCRIPTION
```


(C) Copyright Nokia 2014. All rights reserved.

```
// ----------  -----------------------  ---------------------------------
// 2014.01.15  Rob.Kachmar              Initial creation
// 2014.02.01  Rob.Kachmar              Reducing Math calc for performance
// ==============================================================================
// NOTE: The built-in BrightnessFilter() is more than twice as fast as this.
//       This was simply created to show the logic that can be used to achieve
//       an effect that allows adjusting the brightness of an image.
//       Lumia 920 - 16-17 FPS with BrightnessFilter()
//                   vs
//                   7-8 FPS with BrightnessEffect()
// ==============================================================================

using Nokia.Graphics.Imaging;
using System;

namespace NISDKExtendedEffects.ImageEffects
{
    public class BrightnessEffect : CustomEffectBase
```


(C) Copyright Nokia 2014. All rights reserved.

```
    {
        private double m_BrightnessPercentage = 0;

        public BrightnessEffect(IImageProvider source, double brightness = 0)
            : base(source)
        {
            m_BrightnessPercentage = brightness;
        }

        protected override void OnProcess(PixelRegion sourcePixelRegion, PixelRegion targetPixelRegion)
        {
            var sourcePixels = sourcePixelRegion.ImagePixels;
            var targetPixels = targetPixelRegion.ImagePixels;

            sourcePixelRegion.ForEachRow((index, width, position) =>
            {
                for (int x = 0; x < width; ++x, ++index)
                {
```


(C) Copyright Nokia 2014. All rights reserved.

```
                    uint currentPixel = sourcePixels[index]; // get the current pixel

                    uint red = (currentPixel & 0x00ff0000) >> 16; // red color component

                    uint green = (currentPixel & 0x0000ff00) >> 8; // green color component

                    uint blue = currentPixel & 0x000000ff; // blue color component

                    if (m_BrightnessPercentage < 0)

                    {

                        // Dimming the image - A value of -1 or -100%, will completely reduce all colors to 0

                        // Reduce each color component by the passed in percentage of the amount of color

                        // it is currently displaying.

                        red = (uint)Math.Max(0, (red + (red * m_BrightnessPercentage)));

                        green = (uint)Math.Max(0, (green + (green * m_BrightnessPercentage)));

                        blue = (uint)Math.Max(0, (blue + (blue * m_BrightnessPercentage)));

                    }

                    else

                    {
```


(C) Copyright Nokia 2014. All rights reserved.

```
                        // Brightening the image - A value of 1 or 100%, will
completely increase all colors to 255
```

```
                        // Increase each color component by the passed in
percentage of the amount of color
```

```
                        // is has left before it reaches the max of 255.
```

```
                        red = (uint)Math.Min(255, (red + ((255 - red) *
m_BrightnessPercentage)));
```

```
                        green = (uint)Math.Min(255, (green + ((255 - green) *
m_BrightnessPercentage)));
```

```
                        blue = (uint)Math.Min(255, (blue + ((255 - blue) *
m_BrightnessPercentage)));
```

```
                    }
```

```

```

```
                    // Reassembling each component back into a pixel for the
target pixel location
```

```
                    targetPixels[index] = 0xff000000 | (red << 16) | (green << 8)
| blue;
```

```
                }
```

```
            });
```

```
        }
```

```
    }
```

```
}
```

Testing

(C) Copyright Nokia 2014. All rights reserved.

The Test Apps for Viewing Custom Filters allow you to cycle through a number of different custom effects (including *BrightnessEffect*) and apply them to the real-time camera preview or a static image.

In addition, the custom effect file can be dropped into your own project, and used as described in the section Using the filter.

Pre-requisites

- Read the Core concepts (Nokia Developer Library)
- Read the article Custom Filter QuickStart for Nokia Imaging SDK to understand the general requirements on Custom Effects.

Performance

The *BrightnessEffect* is less efficient than the Nokia Imaging SDK's inbuilt BrightnessFilter.

Device	BrightnessEffect	BrightnessFilter
Lumia 920	7-8 FPS	16-17 FPS

The *BrightnessEffect* class runs at about 7-8 FPS (frames per second). The inbuilt *BrightnessFilter* runs at around 16-17 FPS.

The Real Time Filter Demo test project is configured so that the inbuilt and custom effect run side-by-side for easy comparison.

Code walkthrough

The technique for changing the brightness of an image involves adjusting each of the RGB color components up or down uniformly. Moving the colors down gets them closer to zero (black), and moving them up gets them closer to 255 (white).

If you are dimming an image, you will use a negative percentage value to indicate how dim you would like to make the image. If you choose -1.0 (-100%), then each color component will go to zero making the image completely black. Any other percentage value will be that portion of a particular color component subtracted from itself.

If you are brightening an image, you will use a positive percentage value to indicate how bright you would like to make the image. If you choose 1.0 (100%), then each color component will go to 255 making the image completely white. Any other percentage value will be that portion of a particular color component's distance to 255 added to itself.

```
1. if (m_BrightnessPercentage < 0)
2. {
3.     // Dimming the image - A value of -1 or -100%, will completely reduce all colors to 0
4.     // Reduce each color component by the passed in percentage of the amount of color
5.     // it is currently displaying.
6.     red = (uint)Math.Max(0, (red + (red * m_BrightnessPercentage)));
7.     green = (uint)Math.Max(0, (green + (green * m_BrightnessPercentage)));
8.     blue = (uint)Math.Max(0, (blue + (blue * m_BrightnessPercentage)));
```


(C) Copyright Nokia 2014. All rights reserved.

```
}
else
{
    // Brightening the image - A value of 1 or 100%, will completely increase all colors to 255
    // Increase each color component by the passed in percentage of the amount of color
    // is has left before it reaches the max of 255.
    red = (uint)Math.Min(255, (red + ((255 - red) * m_BrightnessPercentage)));
    green = (uint)Math.Min(255, (green + ((255 - green) * m_BrightnessPercentage)));
    blue = (uint)Math.Min(255, (blue + ((255 - blue) * m_BrightnessPercentage)));
}
```

Using the filter

Drop an image control into your XAML.

```
<Image x:Name="FilterEffectImage" Width="800" Height="480" Stretch="Fill" Grid.RowSpan="2" />
```

Use this code to apply the filter effect to your chosen image and assign it to the XAML image control.

```
// Initialize a WriteableBitmap with the dimensions of the XAML image control
WriteableBitmap writeableBitmap = new WriteableBitmap((int)FilterEffectImage.Width, (int)FilterEffectImage.Height);

```


(C) Copyright Nokia 2014. All rights reserved.

```
// Example: Accessing an image stream within a standard photo chooser task callback
```

```
// http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh394019(v=vs.105).aspx
```

```
//using (var imageStream = new StreamImageSource(e.ChosenPhoto))
```

```

```

```
// Example: Accessing an image stream from a sample picture loaded with the project in a folder called "Pictures"
```

```
var resource = App.GetResourceStream(new Uri(string.Concat("Pictures/", "sample_photo_08.jpg"), UriKind.Relative));
```

```
using (var imageStream = new StreamImageSource(resource.Stream))
```

```
{
```

```
    // Applying the custom filter effect to the image stream
```

```
    using (var customEffect = new BrightnessEffect(imageStream, 0.50))
```

```
    {
```

```
        // Rendering the resulting image to a WriteableBitmap
```

```
        using (var renderer = new WriteableBitmapRenderer(customEffect, writeableBitmap))
```

```
        {
```

```
            // Applying the WriteableBitmap to our xaml image control
```

```
            FilterEffectImage.Source = await renderer.RenderAsync();
```


(C) Copyright Nokia 2014. All rights reserved.

```
            }

        }

    }
```

License

The code has been released with the standard MIT License, and can be viewed in the Github project here.

Summary

Hopefully you've enjoyed seeing how this custom effect implementation unlocks some of the mystery behind the amazing Nokia Imaging SDK.

As with all articles on the wiki, you are welcome to contribute any changes to this code that would improve the quality, whether it be additional features or improving its efficiency.

(C) Copyright Nokia 2014. All rights reserved.