Course: EGDF20
Module: EGE202 Application Programming

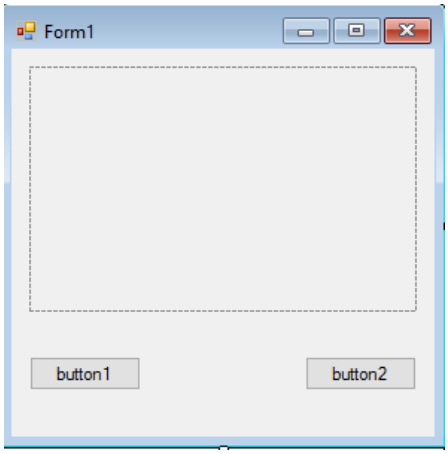
SDL 5: Understanding Colors, Pixels and Imaging Processing Techniques

Objectives: At the end of this lab, the student will learn and understand about color pixels and how to use simple image processing algorithms to process color.

Exercise 1 – Understanding Pixels and Colors

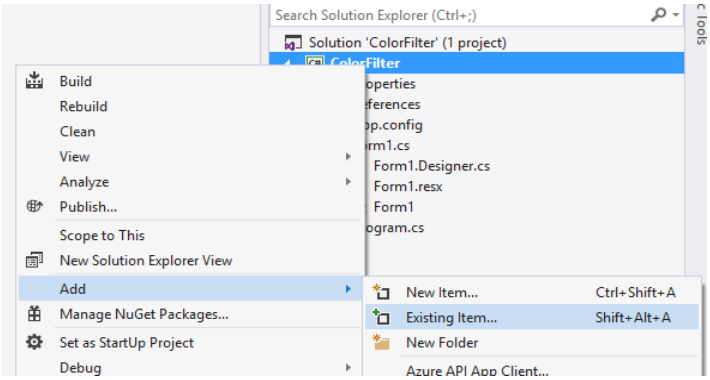
Part 1: Loading an Image from File and Applying Color Filter

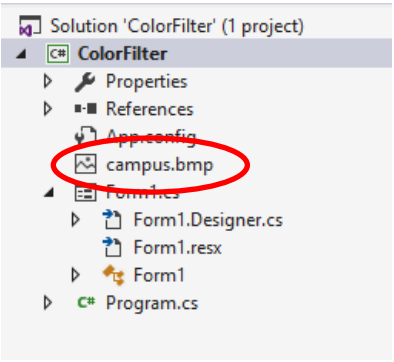
1. Under the ***File*** menu, click ***New Project*** to create a new project. Alternatively you can use the ***New Project*** button or the ***New Project*** link in the ***Start Page*** tab.
2. From the pop-up dialog, select ***Visual C# -> Classic Desktop -> Windows Forms Application***. Type the name of your new project as ***ColorFilter*** and put the project in your own created folder.
3. Double click on "Form1.cs" ***Solution Explorer*** window to launch the ***Form Designer*** tab.
4. From the ***Toolbar***, drag in 2 *Button* and 1 *PictureBox* controls into the *Form1* window. Modify the *(Name)* and *Text* properties based on the table below.



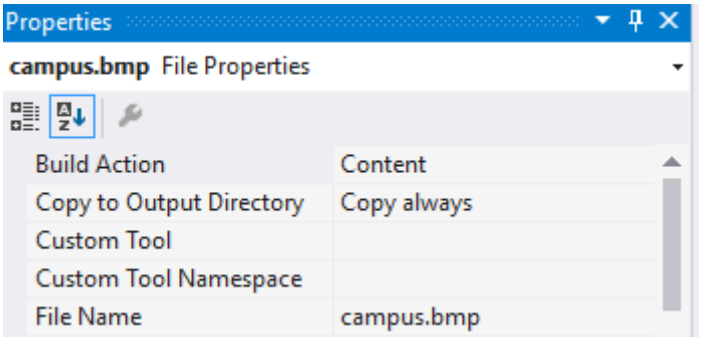
{Name} From	{Name} To	Change the following properties
button1	btnLoad	{Text} = Load
button2	btnTransform	{Text} = Transform
pictureBox1	picImage	{SizeMode} = StretchImage

5. Locate "*campus.bmp*" on your PC (Check with your instructor where to obtain the image file and copy it to your own folder.
6. On the ***Solution Explorer***, right click on the ***ColorFilter*** and choose *Add->Existing Item* to add the file *"campus.bmp".* (Choose ***Image Files (*.gif;*.jpg…)*** for ***File Filter*** type).





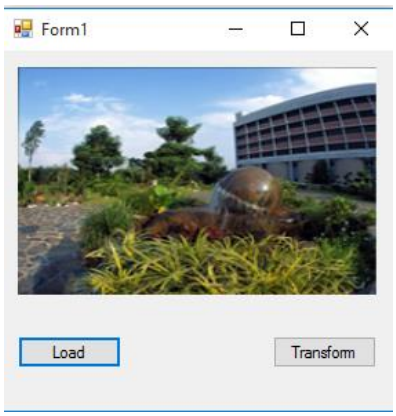
7. Next select "*campus.bmp*" in the ***Solution Explorer*** and right click to view the properties in the ***Properties*** window.
8. In the ***Properties*** window, change the *"Copy to Output Directory"* property to *"Copy Always"*.



9. Next in the ***Form Designer***, double click on the ***Load*** button and modify *btnLoad_Click (…)* to include the following codes. The *Load()* method of **PictureBox** control allows programmers to easily load any images

```
private void btnLoad_Click(object sender, EventArgs e)
{
    picImage.Load("campus.bmp");
}
```

10. Build and run your application. Clicking on the ***Load*** button will load the "*campus.bmp*" image into PictureBox.

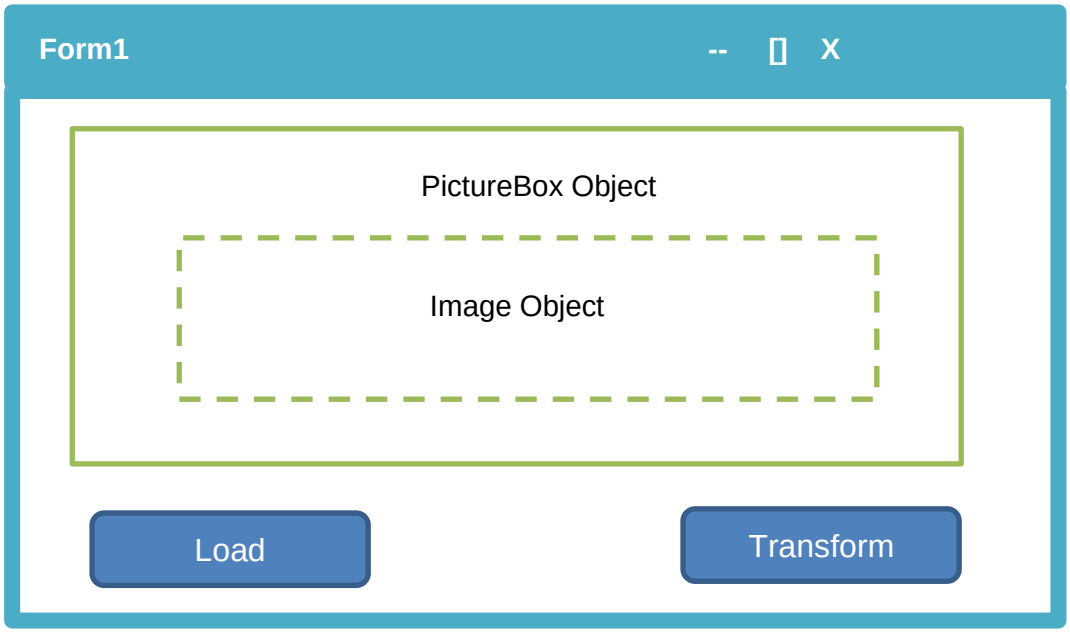


11. Next in the ***Form Designer***, double click on the *Transform* button and *btnTransform_Click (…)* to include the following codes:

```
private void btnTransform_Click(object sender, EventArgs e)
{
    Bitmap oImage = (Bitmap)picImage.Image;
    if (oImage == null)
        return;

    int pixelcol;
    for (int i = 0; i < oImage.Width; i++)
        for (int j = 0; j < oImage.Height; j++)
        {
            pixelcol = oImage.GetPixel(i, j).ToArgb();
            oImage.SetPixel(i, j, Color.FromArgb(pixelcol & 0x00FF0000));

        }
    picImage.Refresh();
}
```



12. Run the application and click on the ***Load*** button followed by the ***Transform*** button. You should now able to transform the image into a red color.

13. Let's analyze the codes for step 11.

Explanation of the codes

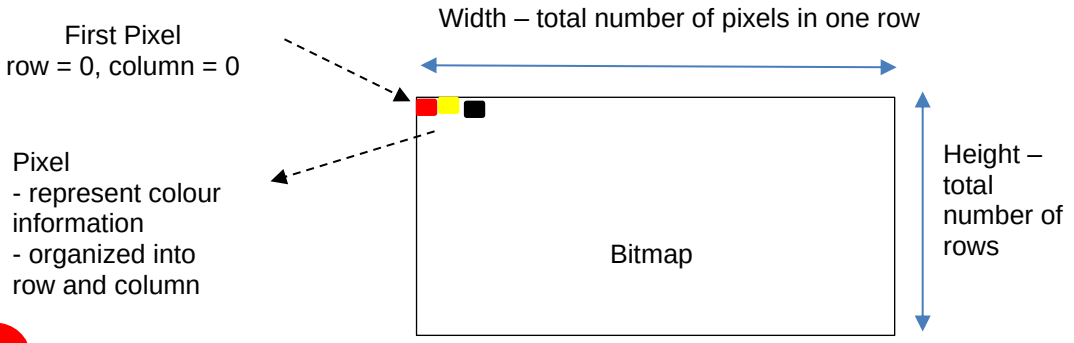
```
Bitmap oImage;
oImage = (Bitmap)picImage.Image;
```

As illustrated in the diagram above, every **PictureBox** control has an **Image** property (object of type **Image** class). This **Image** property contains the "image information" that is displayed on the **PictureBox** control.

```
picImage.Image          \\ access the image object within PictureBox
```

Bitmap class deals with "Images" readily to be displayed on the screen while **Image** class deals with different type of image sources (file {bmp, gif, jpg} or stream {memory, Internet}). Therefore in order to manipulate the images and display to screen we need to convert the **Image** object into **Bitmap** object.

```
oImage = (Bitmap)picImage.Image; \\ convert Image to Bitmap
```



1

```
oImage.Width  \\ retrieve the Bitmap width information
oImage.Height  \\ retrieve the Bitmap height information
```

2

```
\\ double for loop to go through every pixels (every row and column)
for (int i = 0; i < oImage.Width; i++)
    for (int j = 0; j < oImage.Height; j++)
```

3

```
pixelcol = oImage.GetPixel(i, j).ToArgb();
```

int variable
(32 bit or 4 bytes)

GetPixel(. . .)

Retrieve pixel color information based on row = i, column = j

ToArgb(. . .)

Convert from Color information to 32 bit Integer

0xAARRGGBB (Hexadecimal)

Color Values in 32-bit Integer

0xAARRGGBB

AA, RR, GG, BB is hexadecimal digits range from 00 to FF (decimal 0 to 255)

RR for red color intensity

GG for green color intensity

BB for blue color intensity

AA for color transparency (opaque, transparent or translucent)

For simplicity we will set AA to be 00 for transparent color

Example:

0x00FF0000 – Pure RED color (lighter)

0x005A0000 – Pure RED color (darker)

0x000000FF – Pure BLUE color (lighter)

0x00000072 – Pure BLUE color (darker)

0x00560056 – BLUE & RED mix = MAGENTA color

0x00FFFFFF – REG & GREEN & BLUE mix = WHITE color

Full red, green and blue mixed to be white

0x00000000 –BLACK color

No color mixed, hence black

0x00121212- GREY color (darker)

same composition of red, green and blue

0x00A2A2A2- GREY color (lighter)

same composition of red, green and blue

Color Filtering Example

0x00A2A2A2	**& 0x00FF0000**	**=**	**0x00A20000**
Grey color	RED Filter: Logic AND (&) with RED pattern 0x00FF0000		Red color (only has RED component)

4

```
oImage.SetPixel(i, j, Color.FromArgb(pixelcol & 0x00FF0000));
```

SetPixel(. . .)

Set pixel color based on row = i, column = j
and Color structure

Color.FromArgb(. . .)

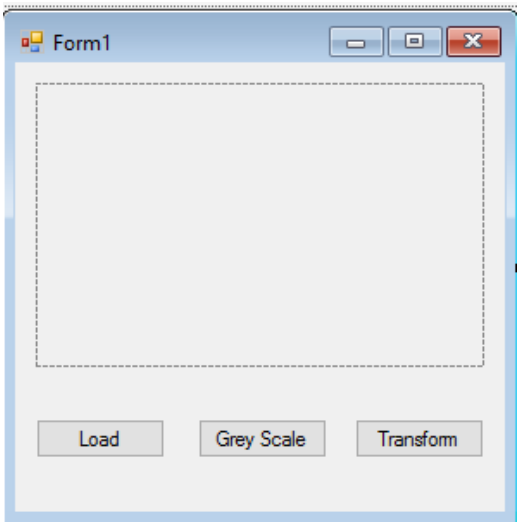
Convert integer to Color information

5

```
picImage.Refresh(); \\ update the PictureBox control
```

Part 2: Working with Color Structure

1. Continuing from Part 1, add one more button called *"Grey Scale"* with name *btnGrey*.



2. Next in the ***Form Designer***, double click on the *Grey Scale* button and modify *btnGrey_Click (…)* to include the following codes:

```
private void btnGrey_Click(object sender, EventArgs e)
{
    Bitmap oImage;
    oImage = (Bitmap)picImage.Image;
    if (oImage == null)
        return;

    Color col;
    int red, green, blue, gray;
    for (int i = 0; i < oImage.Width; i++)
        for (int j = 0; j < oImage.Height; j++)
        {
            col = oImage.GetPixel(i, j);
            red = col.R;
            green = col.G;
            blue = col.B;
            gray = (red + green + blue) / 3;
            oImage.SetPixel(i, j, Color.FromArgb(gray,gray,gray));
        }
    picImage.Refresh();
}
```

3. Let's analyze the codes for step 2

Explanation of the codes

Now we worked with **Color** structure instead of integer.

Previously in *btnTransform_Clicked(…):*

```
oImage.GetPixel(i, j).ToArgb(); \\ return 32 bit integer
```

Now in *btnGrey_Clicked(…)*

```
oImage.GetPixel(i, j); \\ returns a Color structure
```

```
Color col;
col = oImage.GetPixel(i, j);
```

The *GetPixel(…)* will now return a Color structure instead of Integer. The color structure exposes the individual RGB color values through *R* (Red), *G* (Green) and *B* (Blue) properties.

```
red = col.R;
green = col.G;
blue = col.B;
gray = (red + green + blue) / 3;
```

Gray color is formed when all the red, green and blue component has the same value. This value that represents gray color can be estimated easily by taking the average of all the red, green and blue components.

```
gray = (red + green + blue) / 3;
```

or

```
gray = (0.3333)*red + (0.3333)*green + (0.3333)*blue) ;
```

Finally to set a color with red, green and blue all equals to gray value. Noticed that here we are using another overloaded *FromArgb(…)* method

```
oImage.SetPixel(i, j, Color.FromArgb(gray,gray,gray));
```

Color.FromArgb(int red, int green, int blue.)

Self-Assessment Assignment

1	Write down the changes in your codes in order to implement a:
	
	(i) Red Color Filter
	`pixelcol & 0x00FF0000`
	(ii) Green Color Filter
	`pixelcol & _____`
	(iii) Blue Color Filter
	`pixelcol & _____`
	(iv) Magenta Color Filter
	`pixelcol & _____`

2	In the earlier example of creating a grey scale filter we user a simple averaging method `gray = (red + green + blue) / 3;` Luminosity method is a more sophisticated method that takes into account human perception of colors (http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/) Replace the average method with luminosity method. Try it out