

Unsafe code in Image processing

My notes

Introduction:

As the Image processing package which we develop in Image processing course gets bigger its performance gets slower, and the need to use unsafe code increases.

So I took the risk :D and started to modify all my functions in that package to convert it from safe to unsafe code, and since I've faced some problems and have learned some things about bitmaps I've never learned I decided to write them down for those who may face the same problems.

So let's start.

Safe VS Unsafe:

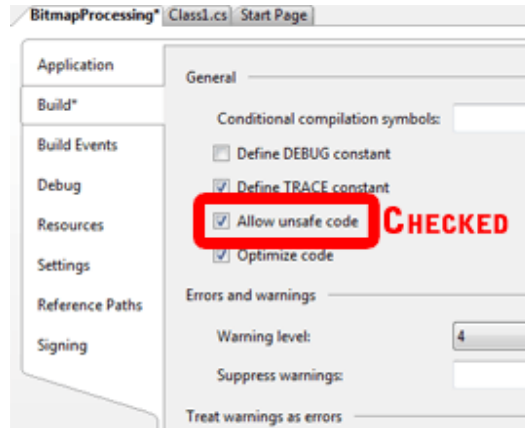
Safe code is the code that runs under CLR. The memory management of safe code is handled by CLR.

Unsafe code is direct memory access or file I/O or database operations. Here you need to manage memory yourself by calling dispose on objects.

For more about this: <http://www.dotnetspider.com/forum/155077-What-Safecode-unsafe-code.aspx>

Before getting started:

We need to tell the C# project to allow unsafe code, R.click on the project and choose properties, then:



Also don't forget to include this library.

```
using System.Drawing.Imaging;
```

What does this code look like?!

Let's take the following function as an example, it's used to convert image to grayscale.

```

private Bitmap GrayScale(Bitmap bmpimg)
{
    BitmapData bmpData = bmpimg.LockBits(new Rectangle(0, 0, bmpimg.Width,
    bmpimg.Height), ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);

    int remain = bmpData.Stride - bmpData.Width * 3;

    unsafe
    {
        byte* ptr = (byte*)bmpData.Scan0;

        for (int i = 0; i < bmpData.Height; i++)
        {
            for (int j = 0; j < bmpData.Width; j++)
            {
                ptr[0] = ptr[1] = ptr[2] = (byte)((ptr[0] + ptr[1] + ptr[2]) /
                3);
                ptr += 3;
            }
            ptr += remain;
        }
    }
    bmpimg.UnlockBits(bmpData);
    return bmpimg;
}

```

To access the image pixels with pointers you've to lock its bits first, using the

```

BitmapData bmpData = bmpimg.LockBits(new Rectangle(0, 0, bmpimg.Width, bmpimg.Height),
ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);

```

LockBits function

This function takes a rectangle as a parameter to determine the rectangle that represents the portion you need to lock in this image, also it takes the ImageLockMode and the PixelFormat and returns an object of BitmapData.

More about BitmapData: <http://msdn.microsoft.com/en-us/library/system.drawing.imaging.bitmapdata.aspx>

Most of time we use this value "Format24bppRgb" as PixelFormat, which means that each pixel contains 24 bits (3 bytes) and colored by RGB system. **RGB color (24-bit) pixel values are stored with bytes as BGR (blue, green, red), byte 1 for Blue, byte 2 for Green and byte 3 for Red.** (http://en.wikipedia.org/wiki/BMP_file_format)

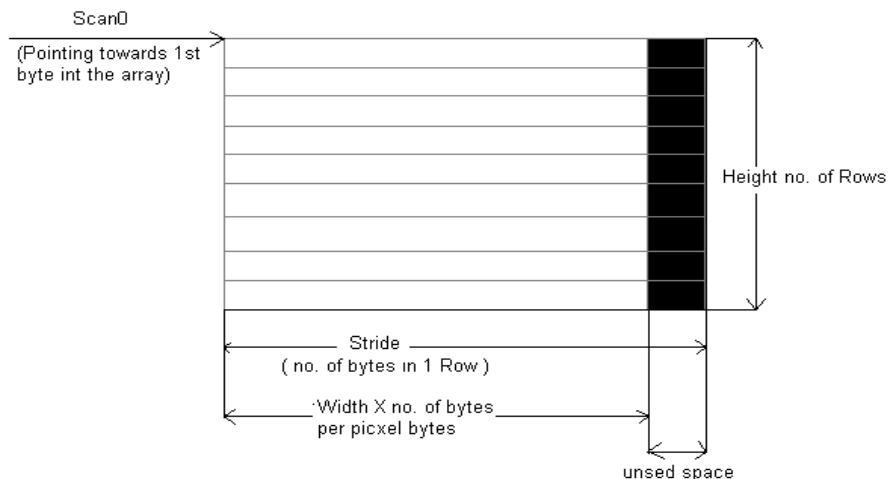
A lot of formats are available in this enumerator, see more about them here

<http://msdn.microsoft.com/en-us/library/system.drawing.imaging.pixelformat.aspx>

In my application I used the “Format24bppRgb” and “Format8bppIndexed” only which I’ll discuss later in this paper.

Now give me your attention in this part,

The basic layout of a locked bitmap array is



Basic Layout of the Locked Image Pixel Array

In brief let me say that the size of pixels row of an image must be a multiple of 4 and since it’s not the case in all images so padding is usually added at the end of each row.

Ex: an image of format 24bpp with width 17 pixels has a stride of 52 bytes, 51 bytes (3*17) and 1 bit as padding.

For more info: <http://www.bobpowell.net/lockingbits.htm>

After understanding this concept let’s see the rest of the code.

```
int remain = bmpData.Stride - bmpData.Width * 3;
```

remain represents the padding magnitude.

Then to start using pointers you have to write your code in an unsafe block.

```
Unsafe
{.....
```

Now we need a pointer to the first pixel of the image

```
byte* ptr = (byte*)bmpData.Scan0;
```

Where scan0 is a property of BitmapData which gets or sets the address of the first **pixel** in the image, note that I said pixel not byte, this means if the image is of PixelFormat "Format24bppRgb" then ptr is a pointer to an array of 3 bytes.

Now we need to loop on the image pixels and change their values to grayscale, we use these nested for loops to do so

```
for (int i = 0; i < bmpData.Height; i++)
{
    for (int j = 0; j < bmpData.Width; j++)
    {
        ptr[0] = ptr[1] = ptr[2] = (byte)((ptr[0] + ptr[1] + ptr[2]) / 3);
        ptr += 3;
    }
    ptr += remain;
}
```

Note this line

```
ptr += remain;
```

Where we add the padding value (remain) to go to the next line and get the next pixel correctly.

And finally

```
bmpimg.UnlockBits(bmpData);
return bmpimg;
}
```

We unlock the bits of the image using UnlockBits method which take the BitmapData object as a parameter and by then your image is converted to grayscale.

You can use this example to apply the unsafe code in many other functions of image processing package the concept is the same.

Take care of the pixel format:

“Image is an array of pixels that carries the color intensities”, yes sometimes, but it’s not always the case.

There’s a format of images called indexed images where the image consists of 2 parts, pixel data and color model data. Each pixel gets its color equivalent by asking the color model for the color corresponding to the pixel’s index.

In Bitmap object the color map is stored in what’s called palette. To get the array of colors in palette, simply:

```
Color[] myPalette = Pic.Palette.Entries;
```

Where Entries is the color array in palette.

More about indexed images: http://www.cylekx.net/help/indexed_images.htm

http://en.wikipedia.org/wiki/Indexed_color

More about palette: <http://msdn.microsoft.com/en-us/library/system.drawing.image.palette.aspx>

So if your image PixelFormat is “Format8bppIndexed”, don’t forget:

1- Lock the image by the same PixelFormat and if you don’t know the format of

```
BitmapData orgImgData = Pic.LockBits(new Rectangle(Point.Empty, Pic.Size),  
ImageLockMode.ReadWrite, Pic.PixelFormat);
```

the input image you can use this

Where **Pic** is the image sent to the function.

2- The padding value will be as follows

```
int remain = bmpData.Stride - bmpData.Width;
```

as the pixel is represented by only one byte.

3- To get the color components of the pixel, construct an array of Color from the Palette.Entries:

```
Color[] myPalette = Pic.Palette.Entries;
```

And to get the red component for example:

```
myPalette[orgImgPtr[0]].R
```

```
|-----|  
|      Hope this was useful : )      |  
|-----|
```

References:

<http://www.vcskicks.com/fast-image-processing.html>
<http://www.devasp.net/net/articles/display/453.html>
<http://www.bobpowell.net/lockingbits.htm>
http://www.codersource.net/csharp_image_Processing.aspx
http://www.cylekx.net/help/indexed_images.htm
<http://msdn.microsoft.com/en-us/library/ms229672.aspx>