DYNAMIC GRADIENTS IN WPF

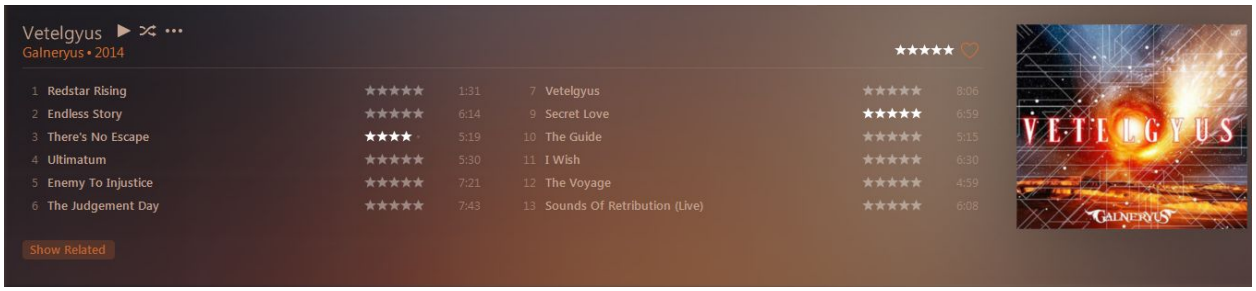
iTunes-esque color detection
Written by Tobe O.

Source Code

Demo Video

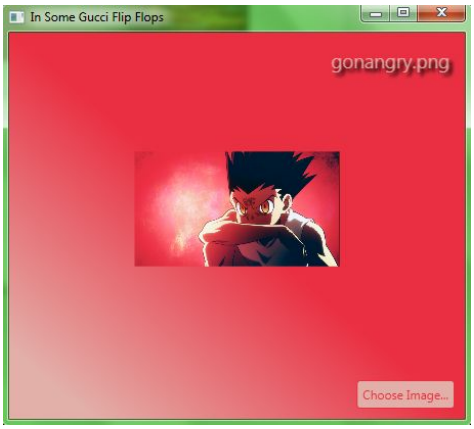
OVERVIEW & PURPOSE

If you have upgraded to iTunes 12, you'll likely appreciate many of its new design features. For me, perhaps the most aesthetically pleasing upgrade is the gradient background for album overviews that adapts itself to the colors of the album in question.



Looks *snazzy*, right? Well, now you can make your own version. The one documented in this tutorial is a less complex gradient, but it still looks nice.

The end product will look something like this:



WHAT THIS TUTORIAL IS NOT

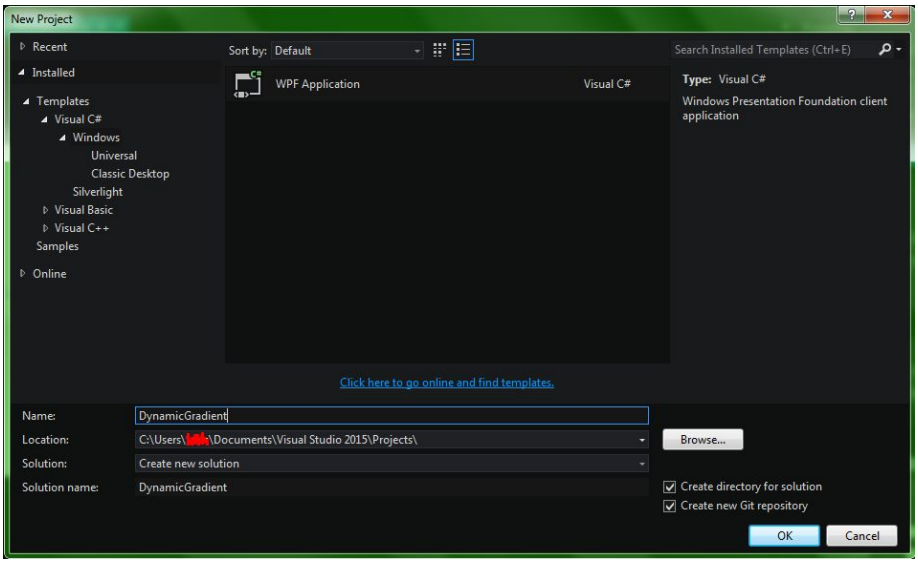
This tutorial is not spoonfed to you. It may help to think of these as guidelines, rather than a lesson. I encourage you to experiment on your own. I uploaded the source code to Github, but within this PDF, it's only included in images, because I want you to type the code out and learn how to do it yourself.

PREREQUISITES

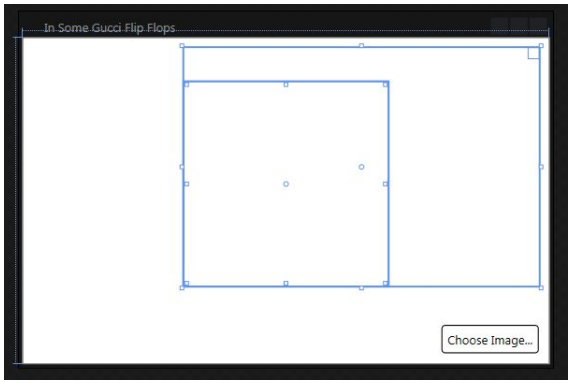
1. All you really need to know is basic C#. You can implement this tutorial in WPF or WinForms, whatever you want.

PROJECT SETUP

1. Create a new WPF Application project. I highly recommend using Visual Studio or Expression Blend 2015.



2. In your *MainWindow.xaml* file, add an image, a label and a button (or anything you can click). I selected the image and label so you could see where I placed them.



3. Add some code to open an image file on button click and set the image's source to it.

```
private void button_Click(object sender, RoutedEventArgs e)
{
    var ofd = new OpenFileDialog
    {
        Title = "Choose an Image...",
        Filter = "Image Files|*.jpg;*.jpeg;*.png;*.gif;*.tiff|All files|*.*"
    };
    if (ofd.ShowDialog() == true)
    {
        var bitmapImage = new BitmapImage(new Uri(ofd.FileName));
        image.Source = bitmapImage;
```

Now, we're ready to do the hard stuff. :)

FINDING PIXELS

So, how does iTunes pick colors to make that fancy gradient with? The process is pretty straightforward: get the colors of pixels in a few spots around the picture, and use those colors for the gradient.

The System.Drawing.Bitmap class provides an API for this called GetPixel(int, int), which returns a System.Drawing.Color. It is far easier to use this method than to implement it using solely WPF. So, if you're using WPF, import System.Drawing.dll into your project.

If you're using WPF, then prior to finding pixels, you'll need to convert your BitmapImage into a Bitmap. One way to do it is as follows:

```
System.Drawing.Bitmap ConvertBitmapImageToBitmap(BitmapImage bitmapImage)
{
    using (MemoryStream outStream = new MemoryStream())
    {
        BitmapEncoder enc = new BmpBitmapEncoder();
        enc.Frames.Add(BitmapFrame.Create(bitmapImage));
        enc.Save(outStream);
        System.Drawing.Bitmap bitmap = new System.Drawing.Bitmap(outStream);

        return new System.Drawing.Bitmap(bitmap);
    }
}
```

Next, you just have to find the actual pixels. For our purposes, it will suffice simply to

pick the pixel in the center of the image, as well as a pixel very close to the left edge, and one very close to the right edge. WPF users can then create a SolidColorBrush for each color.

```
//Left
var leftPixelColor = bitmap.GetPixel(2, bitmap.Height / 2);
//Middle pixel
var middlePixelColor = bitmap.GetPixel(bitmap.Width / 2, bitmap.Height / 2);
//Right pixel
var rightPixelColor = bitmap.GetPixel(bitmap.Width - 2, bitmap.Height / 2);

var leftPixelColorBrush = new SolidColorBrush(Color.FromArgb(leftPixelColor.A, leftPixelColor.R, leftPixelColor.G, leftPixelColor.B));
var middlePixelColorBrush = new SolidColorBrush(Color.FromArgb(middlePixelColor.A, middlePixelColor.R, middlePixelColor.G, middlePixelColor.B));
var rightPixelColorBrush = new SolidColorBrush(Color.FromArgb(rightPixelColor.A, rightPixelColor.R, rightPixelColor.G, rightPixelColor.B));
```

FINAL STEPS

Now, let's make the actual gradient. You can change the GradientStop offsets if you'd like. Here's how I did it:

```
var backgroundBrush = new LinearGradientBrush();
backgroundBrush.GradientStops.Add(new GradientStop
{
    Color = leftPixelColorBrush.Color,
    Offset = 0
});
backgroundBrush.GradientStops.Add(new GradientStop
{
    Color = middlePixelColorBrush.Color,
    Offset = 0.4
});

backgroundBrush.GradientStops.Add(new GradientStop
{
    Color = middlePixelColorBrush.Color,
    Offset = 0.6
});
backgroundBrush.GradientStops.Add(new GradientStop
{
    Color = rightPixelColorBrush.Color,
    Offset = 1
});
backgroundBrush.Transform = new RotateTransform(90d);
Background = backgroundBrush;
```

The remaining steps do not need to be followed exactly; they are solely an example. You have already accomplished the goal of this tutorial, which was to dynamically choose colors and create an intelligent gradient.

I changed the colors of the picture select button and filename label to colors we found

earlier:

```
lblFilename.Foreground = middlePixelColorBrush;
lblFilename.Effect = new DropShadowEffect {
    BlurRadius = 5,
    Color =  Colors.Black,
    ShadowDepth = 5
};
label.Foreground = leftPixelColorBrush;
button.Background = middlePixelColorBrush;
```

Anything else is purely aesthetic. Knock yourself out.

FIN

Thanks for reading this tutorial. Feel free to share it with your friends. Check out my blog for more.