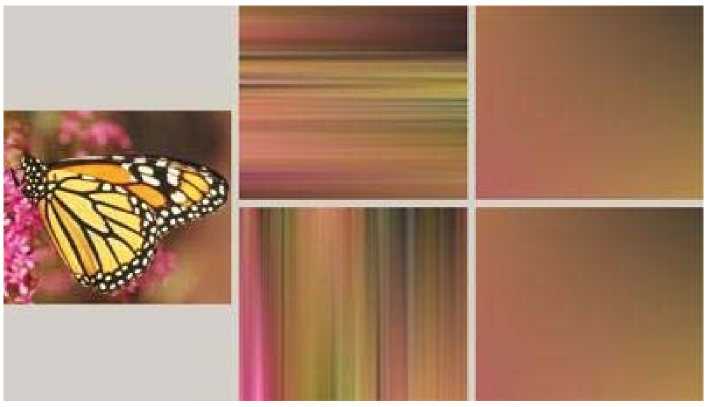
Code Samples: Shading Correction in C#

This code has been developed with Visual C#2.0.
The algorithm computes two orthogonal linear regressions in order to convert any arbitrary image into a flat background image without any pictorial content.

There are two symmetrical possibilities leading to the same background image:

	horizontal + vertical	**vertical + horizontal**
1. regression	compute the regression line of any row	compute the regression line of any column
intermediary image:	the values of the horizontal regression line	the values of the vertical regression line
2. regression	compute the regression line of any column	compute the regression line of any row
background image:	the values of the vertical regression line	the values of the horizontal regression line



left:	original image
upper mid:	intermediary image of 1. horizontal regression
lower mid:	intermediary image of 1. vertical regression
upper right:	vertical regression of upper mid = flat background image
lower right:	horizontal regression of lower mid = flat background image

Both the upper and the lower ways lead to the same background image.
The program below uses the lower way: **vertical + horizontal**.

In a second step the program subtracts the flat background image from the original and rearranges the resulting RGB values between 0 to 255.

The algorithm works with any image format that can be read and written with the `GetPixel(..)`, `SetPixel(..)` methods of the `Bitmap`-class.

Create a new Windows-project "`shading`". Delete the files `Form1.Designer.cs` and `Program.cs` from the project. Clear any prefabricated content from `Form1.cs` and replace it by the following code:

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Windows.Forms;

public class Form1 : Form
{ static void Main() { Application.Run( new Form1() );  }
  Bitmap i0, i1, i2, i3;
  Color [,] a0;
  Int16 [,,] a1;
  float[,,] slope, y0;
```

```
public Form1()
{ MenuItem miRead = new MenuItem( "&Read", new EventHandler( MenuFileRead ) );
  MenuItem miExit = new MenuItem( "&Exit", new EventHandler( MenuFileExit ) );
  MenuItem miFile = new MenuItem( "&File", new MenuItem[] { miRead, miExit } );
  Menu = new System.Windows.Forms.MainMenu( new MenuItem[] { miFile } );
  Text = "Shading Correction by Linear Regression";
  SetStyle( ControlStyles.ResizeRedraw, true );
  try { i0 = new Bitmap( typeof( Form1 ), "shading.shaded_butterfly.jpg" );
        define_images_and_arrays();
        regression1 ( a0 );
        fill_image  ( i1 );
        regression2 ( slope, y0 );
        fill_image  ( i2 );
        shading_correction( slope, y0 );
        fill_image  ( i3 );
        Width  = i0.Width*2 + 10;
  } catch {}
  Height = 600;
}

void MenuFileRead( object obj, EventArgs ea )
{ OpenFileDialog dlg = new OpenFileDialog();
  if ( dlg.ShowDialog() != DialogResult.OK ) return;
  if ( i0 != null ) i0.Dispose();
  i0 = (Bitmap)Image.FromFile( dlg.FileName );
        define_images_and_arrays();
        regression1 ( a0 );
        fill_image  ( i1 );
        regression2 ( slope, y0 );
        fill_image  ( i2 );
        shading_correction( slope, y0 );
        fill_image  ( i3 );
        Width  = i0.Width*2 + 10;
}

void MenuFileExit( object obj, EventArgs ea )
{ Application.Exit(); }

protected override void OnPaint( PaintEventArgs e )
{ Graphics g = e.Graphics;
  g.Clear( BackColor );
  Pen pen = new Pen( Color.Red, 12 );
  pen.EndCap = System.Drawing.Drawing2D.LineCap.ArrowAnchor;
  try { g.DrawImage( i0 ,          0,            0, i0.Width, i0.Height );
        g.DrawImage( i1, i0.Width+10,            0, i0.Width, i0.Height );
        g.DrawImage( i2, i0.Width+10, i0.Height+10, i0.Width, i0.Height );
        g.DrawImage( i3 ,          0, i0.Height+10, i0.Width, i0.Height );
        g.DrawLine( pen, i0.Width-10, i0.Height/2, i0.Width+30, i0.Height/2 );
        g.DrawLine( pen, i0.Width+20, 3*i0.Height/2, i0.Width-20, 3*i0.Height/2 );
        g.DrawLine( pen, 3*i0.Width/2, i0.Height-10, 3*i0.Width/2, i0.Height+30 );
      } catch{}
}

private void define_images_and_arrays()
{ a0    = new Color[ i0.Height, i0.Width ];
  slope = new float[ 2, 3, Math.Max( i0.Height, i0.Width ) ];
  y0    = new float[ 2, 3, Math.Max( i0.Height, i0.Width ) ];
  a1    = new Int16[ 3, i0.Height, i0.Width ];
  if ( i1 != null ) i1.Dispose(); i1 = new Bitmap( i0 );
  if ( i2 != null ) i2.Dispose(); i2 = new Bitmap( i0 );
  if ( i3 != null ) i3.Dispose(); i3 = new Bitmap( i0 );
  for ( int y=0; y < i0.Height; y++ )
    for ( int x=0; x < i0.Width; x++ )
    { a0[y,x] = i0.GetPixel( x, y );
      i1.SetPixel( x, y, Color.Black );
      i2.SetPixel( x, y, Color.Black );
      i3.SetPixel( x, y, Color.Black );
    }
}
```

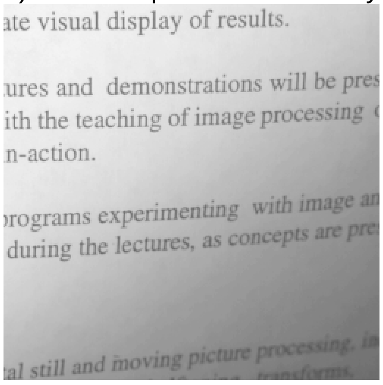
```
private void regression1( Color[,] a0 )
{ Int32 x, y, z, xSize = a0.GetLength(1), ySize = a0.GetLength(0);
  Int32[] sum   = new Int32[3];
  float[] mean  = new float[3];
  float[] sxy   = new float[3];
  float sx2, mid;
  mid = ySize/2f;
  for ( x=0; x < xSize; x++ )
  { for ( z=0; z < 3; z++ ) sum[z] = 0;
    for ( y=0; y < ySize; y++ )
    { sum[0] += a0[y,x].R;
      sum[1] += a0[y,x].G;
      sum[2] += a0[y,x].B;
    }
    for ( z=0; z < 3; z++ )
    { mean[z] = sum[z] / ySize;
      sxy[z] = 0f;
    }
    sx2 = 0f;
    for ( y=0; y < ySize; y++ )
    { sxy[0] += (y-mid)*(a0[y,x].R - mean[0]);
      sxy[1] += (y-mid)*(a0[y,x].G - mean[1]);
      sxy[2] += (y-mid)*(a0[y,x].B - mean[2]);
      sx2    += (y-mid)*(y-mid);
    }
    for ( z=0; z < 3; z++ )
    { slope[1,z,x] = sxy[z] / sx2;
      y0   [1,z,x] = mean[z] - slope[1,z,x]*mid;
    }
  }
}

private void regression2( float[,,] slope, float[,,] y0 )
{ Int32 x, y, z, xSize = i0.Width, ySize = i0.Height;
  float[] sum   = new float[3];
  float[] mean  = new float[3];
  float[] sxy   = new float[3];
  float sx2, mid;
  mid = xSize/2f;
  for ( y=0; y < ySize; y++ )
  { for ( z=0; z < 3; z++ ) sum[z] = 0;
    for ( x=0; x < xSize; x++ )
      for ( z=0; z < 3; z++ ) sum[z] += slope[1,z,x]*y + y0[1,z,x];
    for ( z=0; z < 3; z++ )
    { mean[z] = sum[z] / xSize;
      sxy[z] = 0f;
    }
    sx2 = 0f;
    for ( x=0; x < xSize; x++ )
    { for ( z=0; z < 3; z++ )
        sxy[z] += (x-mid)*(slope[1,z,x]*y + y0[1,z,x] - mean[z]);
      sx2 += (x-mid)*(x-mid);
    }
    for ( z=0; z < 3; z++ )
    { slope[0,z,y] = sxy[z] / sx2;
      y0   [0,z,y] = mean[z] - slope[0,z,y]*mid;
    }
  }
}
```

```
  private void fill_image( Bitmap i )
  { int x, y, z;
    int[] c = new int[3];
    if ( i == i1 )
      for ( x=0; x < i.Width; x++ )
        for ( y=0; y < i.Height; y++ )
        { for ( z=0; z < 3; z++ )
          { c[z] = Convert.ToInt32( slope[1,z,x]*y + y0[1,z,x] );
            c[z] = Math.Max( 0, Math.Min( 255, c[z] ) );
          }
          i.SetPixel( x, y, Color.FromArgb( c[0], c[1], c[2] ) );
        }
    else if ( i == i2 )
      for ( y=0; y < i.Height; y++ )
        for ( x=0; x < i.Width; x++ )
        { for ( z=0; z < 3; z++ )
          { c[z] = Convert.ToInt32( slope[0,z,y]*x + y0[0,z,y] );
            c[z] = Math.Max( 0, Math.Min( 255, c[z] ) );
          }
          i.SetPixel( x, y, Color.FromArgb( c[0], c[1], c[2] ) );
        }
   else if ( i == i3 )
      for ( x=0; x < i0.Width; x++ )
        for ( y=0; y < i0.Height; y++ )
          i.SetPixel( x, y, a0[y,x] );
  }

  private void  shading_correction( float[,,] slope, float[,,] y0 )
  { int x, y, z;
    int[] c = new int[3], min = new int[3], max = new int[3];
    float[] factor = new float[3];
    for ( z=0; z < 3; z++ ) { min[z] = Int32.MaxValue;
                              max[z] = Int32.MinValue; }
    for ( y=0; y < i0.Height; y++ )
      for ( x=0; x < i0.Width; x++ )
      { a1[0,y,x] = Convert.ToInt16( a0[y,x].R - ( slope[0,0,y]*x + y0[0,0,y] ) );
        a1[1,y,x] = Convert.ToInt16( a0[y,x].G - ( slope[0,1,y]*x + y0[0,1,y] ) );
        a1[2,y,x] = Convert.ToInt16( a0[y,x].B - ( slope[0,2,y]*x + y0[0,2,y] ) );
        for ( z=0; z < 3; z++ )
        { if ( a1[z,y,x] < min[z] ) min[z] = a1[z,y,x];
          if ( a1[z,y,x] > max[z] ) max[z] = a1[z,y,x];
        }
      }
    for ( z=0; z < 3; z++ ) factor[z] = 255f / ( max[z] - min[z] );
    for ( x=0; x < i0.Width; x++ )
      for ( y=0; y < i0.Height; y++ )
      { for ( z=0; z < 3; z++ ) c[z] = Convert.ToInt32( ( a1[z,y,x]-min[z] ) * factor[z] );
        a0[y,x] = Color.FromArgb( c[0], c[1], c[2] );
      }
  }
}
```

Recommended experiments:
1) Load different images by using the `File` menu.
3) Store the pictures below on your hard disk and try them out:


ate visual display of results.

ures and demonstrations will be pres
ith the teaching of image processing
n-action.

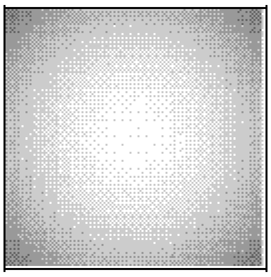
rograms experimenting with image an
during the lectures, as concepts are pre

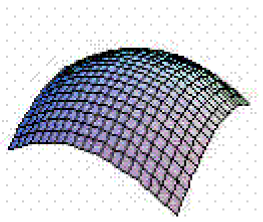
al still and moving picture processing, im
transforms.

FAQ: **What sort of shading can this program remove ?**

A: All sorts of uneven illumination with a brightness ramp, i.e. with gradually change of background color raising or falling from any part of a border to the opposite border.

The program cannot remove circular spotlight like this on the right:





FAQ: **What is the sense of the arrays**:
`slope = new float[ 2, 3, Math.Max( i0.Height, i0.Width ) ];`
`   y0 = new float[ 2, 3, Math.Max( i0.Height, i0.Width ) ];` **?**
A: A linear regression line has an equation of the form `y = slope*x + y0`, where
`x` is the explanatory variable = distance from left/upper image border and
`y` is the dependent variable = brightness.
`slope` is the slope of the regression line, and
`y0` is its intercept (the value of `y` when `x = 0`).
Dimension 0: We need 2 sets of `slope` and `y0` because we compute 2 regressions:
1. regression = vertical regression from the columns of the original image, and
2. regression = horizontal regression from the first regression.
Dimension 1: We need 3 color channels because of the 3 independent red, green and blue values of the pixels.
Dimension 2: We need space for image.`Width` resp. image.`Height` `slopes` and `y0`s: `Math.Max( i0.Height, i0.Width )`.

FAQ: **Why stores the program the `slopes` and `y0`s instead of storing the intermediary images `i1` and `i2` ?**
A: 1. To save memory and 2. to avoid multiple rounding errors.

How to embed an arbitrary sample image into the code:
Copy all the code into an empty `Form1.cs` of a new `Windows Application C#`-project `shading` and (in case of VS 2005) delete `Form1.Designer.cs` and `Program.cs`.
1) In the `Solution Explorer` window right click on `shading`. A context menu opens.
2) Click `Add`. Click `Existing Item`. A file dialog box opens.
3) At the bottom of the box is a drop-down menu: `Files of type:`
4) Select `Image Files (*.gif;*.jpg;...)`
5) Choose an arbitrary image from your computer and leave the file dialog box with button `Add`.
6) The file name should now appear in the `Solution Explorer` tree. Right click this name.
7) A context menu opens. Click `Properties`. A `Properties`-window opens below the `Solution Explorer` window.
8) Click its first property: `Build Action` and set it to `Embedded Resource`.
9) Line 20 of the program below is: `b0 = new Bitmap( typeof( Form1 ), "shading.shaded_butterfly.jpg" );`
Replace `shaded_butterfly.jpg` by the name of your image.