```
//Retinex
//(c) 2013 by Tomasz Lubinski
//www.algorytm.org

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
using System.Drawing.Imaging;

namespace retinex
{
    public partial class Form1 : Form
    {

        /// <summary>
        /// Obraz zrodlowy i wynikowy
        /// </summary>
        private Image zrodlo;
        private Obraz wynik;

        /// <summary>
        /// Zainicjuj
        /// </summary>
        public Form1()
        {
            zrodlo = null;
            InitializeComponent();
        }

        /// <summary>
        /// Wczytaj obraz
        /// </summary>
        private void wczytaj_Click(object sender, EventArgs e)
        {
            OpenFileDialog dlg = new OpenFileDialog();
            dlg.Title = "Otwórz obraz";
            dlg.Filter = "pliki jpg (*.jpg)|*.jpg";
            if (dlg.ShowDialog() == DialogResult.OK)
            {
                zrodlo = new Bitmap(dlg.OpenFile());
                wynik = new Obraz();
```

```
            wynik.obrazWynikowy.Image = new
               Bitmap(dlg.OpenFile());
            wynik.obrazWynikowy.Height =
               wynik.obrazWynikowy.Image.Height;
            wynik.obrazWynikowy.Width =
               wynik.obrazWynikowy.Image.Width;
            wynik.obrazWynikowy.Refresh();
            wynik.ClientSize = new
               System.Drawing.Size(wynik.obrazWynikowy.Width +
               24, wynik.obrazWynikowy.Height + 32);
            wynik.Show();
            wynik.Refresh();
        }
        dlg.Dispose();
    }

    /// <summary>
    /// Zainicjuj wartosci w tabeli skal
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void Form1_Load(object sender, EventArgs e)
    {
        wagi.Rows.Add();
        wagi.Rows.Add();
        wagi.Rows.Add();
        wagi.Rows[0].Cells[0].Value = 0.33;
        wagi.Rows[1].Cells[0].Value = 0.33;
        wagi.Rows[2].Cells[0].Value = 0.34;
        wagi.Rows[0].Cells[1].Value = 5;
        wagi.Rows[1].Cells[1].Value = 50;
        wagi.Rows[2].Cells[1].Value = 250;
    }

    /// <summary>
    /// Wieloskalowy retinex z przywracaniem kolorow (MSR with
       Color Restoration)
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void retinex_Click(object sender, EventArgs e)
    {
        //Nie rob nic wiecej jezeli obraz jest jeszcze
           niewczytany
        if (zrodlo == null)
        {
            return;
        }
```

```
        //Pobierz wartosci alfa i beta
        double alfa = Double.Parse(this.alfa.Text);
        double beta = Double.Parse(this.beta.Text);

        //Kopiuj obrazek zrodlowy
        Bitmap bitmap = (Bitmap)zrodlo.Clone();

        //Pobierz wartosc wszystkich punktow obrazu
        BitmapData bmpData = bitmap.LockBits(new Rectangle(0, 0,
           bitmap.Width, bitmap.Height),
           ImageLockMode.ReadWrite, bitmap.PixelFormat);
        byte[] original = new byte[Math.Abs(bmpData.Stride) *
           bitmap.Height];
        System.Runtime.InteropServices.Marshal.Copy(bmpData.Scan
           0, original, 0, original.Length);

        //Utworz tablice z wynikiem
        double[][][] retinexResult = new double[3][][];
        retinexResult[0] = new double[bitmap.Height][];
        retinexResult[1] = new double[bitmap.Height][];
        retinexResult[2] = new double[bitmap.Height][];
        for (var i = 0; i < bitmap.Height; i++)
        {
            retinexResult[0][i] = new double[bitmap.Width];
            retinexResult[1][i] = new double[bitmap.Width];
            retinexResult[2][i] = new double[bitmap.Width];
        }

        //dla wszystkich skal (MSR - multi scale retinex)
        for (var s = 0; s < wagi.Rows.Count-1; s++)
        {
            byte[] blured = (byte[])original.Clone();

            //Pobierz wspolczynniki do rozmywania gaussowskiego
            double[] coefs =
               gaussian_coefs(Double.Parse(wagi.Rows[s].Cells[1].
               Value.ToString()));

            //Rozmywaj gaussowsko w poziomie
            for (int i = 0; i < bitmap.Height; i++)
            {
                gausssmooth(blured, i * Math.Abs(bmpData.Stride)
                   + 0, 3, Math.Abs(bmpData.Stride) / 3, coefs);
                gausssmooth(blured, i * Math.Abs(bmpData.Stride)
                   + 1, 3, Math.Abs(bmpData.Stride) / 3, coefs);
                gausssmooth(blured, i * Math.Abs(bmpData.Stride)
                   + 2, 3, Math.Abs(bmpData.Stride) / 3, coefs);
```

```
            }

            //Rozmywaj gaussowsko w pionie
            for (int i = 0; i < Math.Abs(bmpData.Stride) / 3;
                i++)
            {
                gausssmooth(blured, i * 3 + 0,
                    Math.Abs(bmpData.Stride), bitmap.Height,
                    coefs);
                gausssmooth(blured, i * 3 + 1,
                    Math.Abs(bmpData.Stride), bitmap.Height,
                    coefs);
                gausssmooth(blured, i * 3 + 2,
                    Math.Abs(bmpData.Stride), bitmap.Height,
                    coefs);
            }

            //Retinex (SSR - single scale retinex)
            double weight =
                Double.Parse(wagi.Rows[s].Cells[0].Value.ToString(
                ));
            for (var i = 0; i < bitmap.Height; i++)
            {
                for (var j = 0; j < bitmap.Width; j++)
                {
                    int index = i * Math.Abs(bmpData.Stride) + j
                        * 3;
                    retinexResult[0][i][j] += weight *
                        Math.Log((original[index + 0] + 1.0) /
                        (blured[index + 0] + 1.0));
                    retinexResult[1][i][j] += weight *
                        Math.Log((original[index + 1] + 1.0) /
                        (blured[index + 1] + 1.0));
                    retinexResult[2][i][j] += weight *
                        Math.Log((original[index + 2] + 1.0) /
                        (blured[index + 2] + 1.0));
                }
            }
        }

        //Wyszukaj wartosci najmniejszej i najwiekszej
        double min = retinexResult[0][0][0];
        double max = min;
        for (var i = 0; i < bitmap.Height; i++)
        {
            for (var j = 0; j < bitmap.Width; j++)
            {
                if (min > retinexResult[0][i][j])
```

```
                    min = retinexResult[0][i][j];
                else if (max < retinexResult[0][i][j])
                    max = retinexResult[0][i][j];
                if (min > retinexResult[1][i][j])
                    min = retinexResult[1][i][j];
                else if (max < retinexResult[1][i][j])
                    max = retinexResult[1][i][j];
                if (min > retinexResult[2][i][j])
                    min = retinexResult[2][i][j];
                else if (max < retinexResult[2][i][j])
                    max = retinexResult[2][i][j];
            }
        }

        //Wyswietl rezultat
        double range = (max - min) / 255.0;
        for (var i = 0; i < bitmap.Height; i++)
        {
            for (var j = 0; j < bitmap.Width; j++)
            {
                //Przywracanie kolorow (MSRCR - multiscale
                  retinex with color restoration)
                int index = i * Math.Abs(bmpData.Stride) + j * 3;
                double sum = original[index + 0] +
                  original[index + 1] + original[index + 2] +
                  3.0;

                double value = retinexResult[0][i][j] * beta *
                  Math.Log(alfa * (original[index + 0] + 1.0) /
                  sum);
                value = (value - min) / range;
                if (value > 255)
                    original[index + 0] = 255;
                else if (value < 0)
                    original[index + 0] = 0;
                else
                    original[index + 0] = (byte)value;

                value = retinexResult[1][i][j] * beta *
                  Math.Log(alfa * (original[index + 1] + 1.0) /
                  sum);
                value = (value - min) / range;
                if (value > 255)
                    original[index + 1] = 255;
                else if (value < 0)
                    original[index + 1] = 0;
                else
                    original[index + 1] = (byte)value;
```

```
                    value = retinexResult[2][i][j] * beta *
                        Math.Log(alfa * (original[index + 2] + 1.0) /
                        sum);
                    value = (value - min) / range;
                    if (value > 255)
                        original[index + 2] = 255;
                    else if (value < 0)
                        original[index + 2] = 0;
                    else
                        original[index + 2] = (byte)value;
                }

            }
            System.Runtime.InteropServices.Marshal.Copy(original, 0,
                bmpData.Scan0, original.Length);
            bitmap.UnlockBits(bmpData);
            wynik.obrazWynikowy.Image = bitmap;
        }

        /// <summary>
        /// Papers:  "Recursive Implementation of the gaussian
            filter.",
        /// Ian T. Young , Lucas J. Van Vliet, Signal Processing
            44, Elsevier 1995.
        /// </summary>
        /// <param name="sigma"></param>
        /// <returns></returns>
        private double[] gaussian_coefs(double sigma)
        {
            double[] result = new double[5];
            double q = 0;
            if (sigma >= 2.5)
                q = 0.98711 * sigma - 0.96330;
            else if ((sigma >= 0.5) && (sigma < 2.5))
                q = 3.97156 - 4.14554 * Math.Sqrt(1.0 - 0.26891 *
                    sigma);
            else
                q = 0.1147705018520355224609375;

            double q2 = q * q;
            double q3 = q * q2;
            result[0] = (1.57825+(2.44413*q)+(1.4281
                *q2)+(0.422205*q3));
            result[1] =          (2.44413*q)+(2.85619*q2)+(1.26661
                *q3);
            result[2] =                      -((1.4281*q2)+(1.26661
                *q3));
```

```
            result[3] =
               (0.422205*q3);
            result[4] =
               (1.0-((result[1]+result[2]+result[3])/result[0]));

            return result;
        }

        private void gausssmooth(byte[] img, int shift, int
           rowstride, int size, double[] coefs)
        {
            // forward pass
            double[] w1 = new double[size+3];
            double[] w2 = new double[size+6];
            w1[0] = img[shift];
            w1[1] = img[shift];
            w1[2] = img[shift];
            for(int i=0, n=3; i<size; i++, n++)
            {
                w1[n] = (coefs[4]*img[shift+i*rowstride] +
                        ((coefs[1]*w1[n-1] +
                        coefs[2]*w1[n-2] +
                        coefs[3]*w1[n-3] ) / coefs[0]));
            }
            w1[size+0] = w1[size-1];
            w1[size+1] = w1[size-2];
            w1[size+2] = w1[size-3];

            // backward pass
            w2[size+0]= w1[size+0];
            w2[size+1]= w1[size+1];
            w2[size+2]= w1[size+2];
            w2[size+3]= w1[size+2];
            w2[size+4]= w1[size+2];
            w2[size+5]= w1[size+2];
            for(int i=size-1, n=i+3; i>=0; i--, n--)
            {
                w2[n] = (coefs[4]*w1[n] +
                    ((coefs[1]*w2[n+1] +
                    coefs[2]*w2[n+2] +
                    coefs[3]*w2[n+3] ) / coefs[0]));
                img[shift + i * rowstride] = (byte)w2[n];
            }
        }
    }
}
```