Barba Ilva Cimdiņa

171RDB115

4.praktiskais darbs

Histogrammu attēlošana

Programmas kods

Form1.cs

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

namespace IKAA_171rdb115_2
{
    public partial class Form1 : Form
    {
        public imgData imgData = new imgData();
        public Form1()
        {
            InitializeComponent();
        }

        private void openButton_Click(object sender, EventArgs e)
        {
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                pictureBox1.Image =
Bitmap.FromFile(openFileDialog1.FileName);
            }
            Bitmap bmp = (Bitmap)pictureBox1.Image.Clone();
            imgData.readImage(bmp);
            pictureBox2.Image = imgData.drawImage("RGB");
            imgData.hist2.drawHistogram(chart1, "RGB");
            imgData.hist2.drawHistogram(chart2, "RGB");
        }


        private void pictureBox1_MouseClick(object sender, MouseEventArgs
e)
        {
            if (pictureBox1.Image != null)
            {
                Bitmap bmpi = pictureBox1.Image as Bitmap;
                double kX = (double)pictureBox1.Image.Width /
pictureBox1.Width;
                double kY = (double)pictureBox1.Image.Height /
pictureBox1.Height;
                double k = Math.Max(kX, kY);
                //centrējam attēlu pēc pictureBox izmēra
                double nobideX = (pictureBox1.Width * k -
pictureBox1.Image.Width) / 2;
                double nobideY = (pictureBox1.Height * k -
pictureBox1.Image.Height) / 2;
                //zīmējam attēlu mērogojot pēc pictureBox
                double kx = Math.Round(e.X * k - nobideX);
                double ky = Math.Round(e.Y * k - nobideY);
                try
                {
                    bmpi.SetPixel(Convert.ToInt32(kx),
Convert.ToInt32(ky), colorDialog1.Color);
                    pictureBox1.Refresh();
                }
                catch (Exception) { label5.Text = "Can't color pixel
outside image"; }
            }
        }
```

```csharp
        private void pictureBox1_MouseMove(object sender, MouseEventArgs
e)
        {
            if (pictureBox1.Image != null)
            {
                Bitmap bmpo = pictureBox1.Image as Bitmap;
                double kX = (double)pictureBox1.Image.Width /
pictureBox1.Width;
                double kY = (double)pictureBox1.Image.Height /
pictureBox1.Height;
                double k = Math.Max(kX, kY);
                //centrējam attēlu pēc pictureBox izmēra
                double nobideX = (pictureBox1.Width * k -
pictureBox1.Image.Width) / 2;
                double nobideY = (pictureBox1.Height * k -
pictureBox1.Image.Height) / 2;
                //zīmējam attēlu mērogojot pēc pictureBox
                double kx = Math.Round(e.X * k - nobideX);
                double ky = Math.Round(e.Y * k - nobideY);
                //izvadam label teksta laukā konvērtētu vērtību no vesela
skaitļa uz tekstu
                try
                {
                    Color colororg = bmpo.GetPixel(Convert.ToInt32(kx),
Convert.ToInt32(ky));
                    PixelClassHSV hsvPixel = new
PixelClassHSV(colororg.R, colororg.G, colororg.B);
                    PixelClassCMYK cmykPixel = new
PixelClassCMYK(colororg.R, colororg.G, colororg.B);
                    PixelClassYUV yuvPixel = new
PixelClassYUV(colororg.R, colororg.G, colororg.B);
                    label1.Text = "RGB \nR = " + colororg.R + ", G = " +
colororg.G + ", B = " + colororg.B;
                    label2.Text = "RGB (inversed) \nR = " + (255 -
colororg.R) + ", G = " + (255 - colororg.G) + ", B = " + (255 -
colororg.B);
                    label3.Text = "HSV \nH = " + hsvPixel.H + ", S = " +
hsvPixel.S + "%, V = " + hsvPixel.V + "%";
                    label4.Text = "CMYK \nC = " +
Convert.ToInt32(cmykPixel.C * 100) + "%, M = " +
Convert.ToInt32(cmykPixel.M * 100)
                        + "%, Y = " + Convert.ToInt32(cmykPixel.Y * 100)
+ "%, K = " + Convert.ToInt32(cmykPixel.K * 100) + "%";
                    label5.Text = "x, y = " + Convert.ToString(kx) + ","
+ Convert.ToString(ky);
                    label6.Text = "YUV \nY = " +
Convert.ToInt32(yuvPixel.Yy) + ", U = " + Convert.ToInt32(yuvPixel.U) +
", V = " + Convert.ToInt32(yuvPixel.Vv);
                }
                catch(Exception) { label5.Text = "Can't read coordinates
outside image"; }


            }
        }

        private void pictureBox2_MouseClick(object sender, MouseEventArgs
e)
        {
            if (pictureBox2.Image != null)
            {
                Bitmap bmpi = pictureBox2.Image as Bitmap;
                double kX = (double)pictureBox2.Image.Width /
pictureBox2.Width;
```

```csharp
                double kY = (double)pictureBox2.Image.Height /
pictureBox2.Height;
                double k = Math.Max(kX, kY);
                //centrējam attēlu pēc pictureBox izmēra
                double nobideX = (pictureBox2.Width * k -
pictureBox2.Image.Width) / 2;
                double nobideY = (pictureBox2.Height * k -
pictureBox2.Image.Height) / 2;
                //zīmējam attēlu mērogojot pēc pictureBox
                double kx = Math.Round(e.X * k - nobideX);
                double ky = Math.Round(e.Y * k - nobideY);
                try
                {
                    bmpi.SetPixel(Convert.ToInt32(kx),
Convert.ToInt32(ky), colorDialog1.Color);
                    pictureBox2.Refresh();
                }
                catch (Exception) { label5.Text = "Can't color pixel
outside image"; }
            }
        }

        private void radioButton1_CheckedChanged(object sender, EventArgs
e)
        { //RGB
            radioButton3.Checked = true; //Composite
            radioButton4.Text = "Red";
            radioButton5.Text = "Green";
            radioButton6.Text = "Blue";
            radioButton7.Text = "Intensity";
            radioButton7.Visible = true; //Intensity
            if (imgData.img != null)
            {
                pictureBox2.Image = imgData.drawImage("RGB");
                imgData.hist2.drawHistogram(chart2, "RGB");
            }
        }

        private void radioButton2_CheckedChanged(object sender, EventArgs
e)
        { //HSV
            radioButton3.Checked = true; //Composite
            radioButton4.Text = "Hue";
            radioButton5.Text = "Saturation";
            radioButton6.Text = "Value";
            radioButton7.Visible = false; //Intensity
            if (imgData.img != null)
            {
                pictureBox2.Image = imgData.drawImage("HSV");
                imgData.hist2.drawHistogram(chart2, "HSV");
            }
        }

        private void radioButton8_CheckedChanged(object sender, EventArgs
e)
        {    //CMYK
            radioButton3.Checked = true; //Composite
            radioButton4.Text = "Cyan";
            radioButton5.Text = "Magenta";
            radioButton6.Text = "Yellow";
            radioButton7.Text = "Key";
            radioButton7.Visible = true;
            if (imgData.img != null)
            {
```

```csharp
                pictureBox2.Image = imgData.drawImage("CMYK");
            }
        }

        private void radioButton9_CheckedChanged(object sender, EventArgs
e)
        {
            radioButton3.Checked = true; //Composite
            radioButton4.Text = "Luminance (Y)";
            radioButton5.Text = "Blue-luminance (U)";
            radioButton6.Text = "Red-luminance (V)";
            radioButton7.Visible = false;
            if (imgData.img != null)
            {
                pictureBox2.Image = imgData.drawImage("YUV");
            }
        }

        private void radioButton3_CheckedChanged(object sender, EventArgs
e)
        {
            if (imgData.img != null)
            {
                if (radioButton1.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("RGB");
                    imgData.hist2.drawHistogram(chart2, "RGB");
                }
                else if (radioButton2.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("HSV");
                    imgData.hist2.drawHistogram(chart2, "HSV");
                }
                else if (radioButton8.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("CMYK");
                }
                else
                {
                    pictureBox2.Image = imgData.drawImage("YUV");
                }

            }
        }

        private void radioButton4_CheckedChanged(object sender, EventArgs
e)
        {
            if (imgData.img != null)
            {
                if (radioButton1.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("R");
                    imgData.hist2.drawHistogram(chart2, "R");
                }
                else if (radioButton2.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("H");
                    imgData.hist2.drawHistogram(chart2, "H");
                }
                else if (radioButton8.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("C");
                }
```

```csharp
                else
                {
                    pictureBox2.Image = imgData.drawImage("Yy");
                }
            }
        }

        private void radioButton5_CheckedChanged(object sender, EventArgs
e)
        {
            if (imgData.img != null)
            {
                if (radioButton1.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("G");
                    imgData.hist2.drawHistogram(chart2, "G");
                }
                else if (radioButton2.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("S");
                    imgData.hist2.drawHistogram(chart2, "S");
                }
                else if (radioButton8.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("M");
                }
                else
                {
                    pictureBox2.Image = imgData.drawImage("U");
                }
            }
        }

        private void radioButton6_CheckedChanged(object sender, EventArgs
e)
        {
            if (imgData.img != null)
            {
                if (radioButton1.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("B");
                    imgData.hist2.drawHistogram(chart2, "B");
                }
                else if (radioButton2.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("V");
                    imgData.hist2.drawHistogram(chart2, "V");
                }
                else if (radioButton8.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("Y");
                }
                else
                {
                    pictureBox2.Image = imgData.drawImage("Vv");
                }
            }
        }

        private void radioButton7_CheckedChanged(object sender, EventArgs
e)
        {
            if (imgData.img != null)
            {
```

```csharp
                if (radioButton1.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("I");
                    imgData.hist2.drawHistogram(chart2, "I");
                }
                else
                {
                    pictureBox2.Image = imgData.drawImage("K");
                }
            }
        }

        private void saveButton_Click(object sender, EventArgs e)
        {
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                pictureBox2.Image.Save(saveFileDialog1.FileName);
        }

        private void invertButton_Click(object sender, EventArgs e)
        {

            Bitmap bmp = (Bitmap)pictureBox1.Image.Clone();
            imgData.readImage(bmp);
            pictureBox2.Image = imgData.drawImage("Invert");
        }

        private void colorButton_Click(object sender, EventArgs e)
        {
            colorDialog1.ShowDialog();
            colorButton.BackColor = colorDialog1.Color;
        }
    }
}
```

imgData.cs

```csharp
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;

namespace IKAA_171rdb115_2
{
    public class imgData
    {
        public PixelClassRGB[,] img;
        public PixelClassHSV[,] imghsv;
        public PixelClassCMYK[,] imgcmyk;
        public PixelClassYUV[,] imgyuv;
        public PixelClassRGB[,] imgnew;
        public Histogram hist1; //original image
        public Histogram hist2; //edited image

        ~imgData()
        {
            img = null;
            imghsv = null;
            imgcmyk = null;
            imgyuv = null;
            imgnew = null;
            hist1 = null;
            hist2 = null;
        }
        public void readImage(Bitmap bmp)
        {
            var watchread = System.Diagnostics.Stopwatch.StartNew();
            img = new PixelClassRGB[bmp.Width, bmp.Height];
            imgnew = new PixelClassRGB[bmp.Width, bmp.Height];
            imghsv = new PixelClassHSV[bmp.Width, bmp.Height];
            imgcmyk = new PixelClassCMYK[bmp.Width, bmp.Height];
            imgyuv = new PixelClassYUV[bmp.Width, bmp.Height];
            hist1 = new Histogram();
            hist2 = new Histogram();
            //nolasām datus no attēla
            var bmpData = bmp.LockBits(new Rectangle(0, 0, bmp.Width,
bmp.Height), ImageLockMode.ReadOnly, bmp.PixelFormat);
            //nolasām atmiņā datus par attēlu
            IntPtr ptr = IntPtr.Zero; //mēģinām nolasīt rindu
            int pixelComponents; //kanālu skaits
            if (bmpData.PixelFormat == PixelFormat.Format24bppRgb) //ja
ir 24 bitu formāts
            {
                pixelComponents = 3; //kanālu skaits
            }
            else if (bmpData.PixelFormat == PixelFormat.Format32bppRgb)
//ja ir 32 bitu formāts
            {
                pixelComponents = 4;
            }
            else pixelComponents = 0;
            var line = new byte[bmp.Width * pixelComponents]; //the
length of row array we scan from image
            for (int y = 0; y < bmpData.Height; y++)
            {
                ptr = bmpData.Scan0 + y * bmpData.Stride;
                //nolasām no pirmā pixeļa un stride-pixeļu rinas platums
                Marshal.Copy(ptr, line, 0, line.Length);
```

```csharp
                    for (int x = 0; x < bmpData.Width; x++)
                    {
                        img[x, y] = new PixelClassRGB(line[pixelComponents *
x + 2], line[pixelComponents * x + 1], line[pixelComponents * x]); //BGR
                        imgnew[x, y] = new PixelClassRGB(line[pixelComponents
* x + 2], line[pixelComponents * x + 1], line[pixelComponents * x]);
//BGR
                        imghsv[x, y] = new PixelClassHSV(img[x, y].R, img[x,
y].G, img[x, y].B);
                        imgcmyk[x, y] = new PixelClassCMYK(img[x, y].R,
img[x, y].G, img[x, y].B);
                        imgyuv[x, y] = new PixelClassYUV(img[x, y].R, img[x,
y].G, img[x, y].B);
                    }
                }
            bmp.UnlockBits(bmpData); //nolasīšanas rezultāts
            hist1.readHistogram(img, imghsv);
            hist2.readHistogram(imgnew, imghsv);
            watchread.Stop();
            var elapsedMs = watchread.ElapsedMilliseconds;
            Console.WriteLine("Image Read time: " + elapsedMs);
        }

        public Bitmap drawImage(string mode)
        {
            var watchdraw = System.Diagnostics.Stopwatch.StartNew();
            if (img != null)
            {
                IntPtr ptr = IntPtr.Zero;
                int Height = img.GetLength(1);
                int Width = img.GetLength(0);
                var bmp = new Bitmap(Width, Height,
PixelFormat.Format24bppRgb);
                var bmpData = bmp.LockBits(new Rectangle(0, 0, bmp.Width,
bmp.Height), ImageLockMode.WriteOnly, bmp.PixelFormat);
                var line = new byte[bmp.Width * 3]; //3 kanāli
                for (int y = 0; y < bmpData.Height; y++)
                {
                    for (int x = 0; x < bmpData.Width; x++)
                    {
                        switch (mode)
                        {
                            case "RGB":
                                {
                                    line[3 * x] = img[x, y].B; //blue
                                    line[3 * x + 1] = img[x, y].G;
//green
                                    line[3 * x + 2] = img[x, y].R; //red
                                    imgnew[x, y].R = line[3 * x + 2];
                                    imgnew[x, y].G = line[3 * x + 1];
                                    imgnew[x, y].B = line[3 * x];
                                    imgnew[x, y].I =
Convert.ToByte(0.0722f * imgnew[x, y].B + 0.7152f * imgnew[x, y].G +
0.2126f * imgnew[x, y].R);
                                    break;
                                } //rgb
                            case "R":
                                {
                                    line[3 * x] = 0; //blue
                                    line[3 * x + 1] = 0; //green
                                    line[3 * x + 2] = img[x, y].R; //red
                                    imgnew[x, y].R = line[3 * x + 2];
                                    imgnew[x, y].G = line[3 * x + 1];
                                    imgnew[x, y].B = line[3 * x];
```

```csharp
                                break;
                            } //red
                        case "G":
                            {
                                line[3 * x] = 0; //blue
                                line[3 * x + 1] = img[x, y].G;
//green
                                line[3 * x + 2] = 0; //red
                                imgnew[x, y].R = line[3 * x + 2];
                                imgnew[x, y].G = line[3 * x + 1];
                                imgnew[x, y].B = line[3 * x];
                                break;
                            } //green
                        case "B":
                            {
                                line[3 * x] = img[x, y].B; //blue
                                line[3 * x + 1] = 0; //green
                                line[3 * x + 2] = 0; //red
                                imgnew[x, y].R = line[3 * x + 2];
                                imgnew[x, y].G = line[3 * x + 1];
                                imgnew[x, y].B = line[3 * x];
                                break;
                            } //blue
                        case "I":
                            {
                                line[3 * x] = img[x, y].I; //blue
                                line[3 * x + 1] = img[x, y].I;
//green
                                line[3 * x + 2] = img[x, y].I; //red
                                imgnew[x, y].R = line[3 * x + 2];
                                imgnew[x, y].G = line[3 * x + 1];
                                imgnew[x, y].B = line[3 * x];
                                imgnew[x, y].I =
Convert.ToByte(0.0722f * imgnew[x, y].B + 0.7152f * imgnew[x, y].G +
0.2126f * imgnew[x, y].R);
                                break;
                            } //grayscale
                        case "Invert":
                            {
                                line[3 * x] = Convert.ToByte(255 -
img[x, y].B); //blue
                                line[3 * x + 1] = Convert.ToByte(255
- img[x, y].G); //green
                                line[3 * x + 2] = Convert.ToByte(255
- img[x, y].R); //red
                                break;
                            } //inverted
                        case "HSV":
                            {
                                line[3 * x] = img[x,
y].hsvToRGB(imghsv[x, y].H, imghsv[x, y].S, imghsv[x, y].V).B; //blue
                                line[3 * x + 1] = img[x,
y].hsvToRGB(imghsv[x, y].H, imghsv[x, y].S, imghsv[x, y].V).G; //green
                                line[3 * x + 2] = img[x,
y].hsvToRGB(imghsv[x, y].H, imghsv[x, y].S, imghsv[x, y].V).R; //red
                                break;
                            } //hue saturation value
                        case "H":
                            {
                                line[3 * x] = img[x,
y].hsvToRGB(imghsv[x, y].H, 255, 255).B; //blue
                                line[3 * x + 1] = img[x,
y].hsvToRGB(imghsv[x, y].H, 255, 255).G; //green
```

```csharp
                                        line[3 * x + 2] = img[x,
y].hsvToRGB(imghsv[x, y].H, 255, 255).R; //red
                                        break;
                                    } //hue
                                case "S":
                                    {
                                        line[3 * x] = imghsv[x, y].S; //blue
                                        line[3 * x + 1] = imghsv[x, y].S;
//green
                                        line[3 * x + 2] = imghsv[x, y].S;
//red
                                        break;
                                    } //saturation
                                case "V":
                                    {
                                        line[3 * x] = imghsv[x, y].V;
                                        line[3 * x + 1] = imghsv[x, y].V;
                                        line[3 * x + 2] = imghsv[x, y].V;
                                        break;
                                    } //value
                                case "CMYK":
                                    {
                                        line[3 * x] = img[x,
y].cmykToRGB(imgcmyk[x, y].C, imgcmyk[x, y].M, imgcmyk[x, y].Y,
imgcmyk[x,y].K).B; //blue
                                        line[3 * x + 1] = img[x,
y].cmykToRGB(imgcmyk[x, y].C, imgcmyk[x, y].M, imgcmyk[x, y].Y,
imgcmyk[x, y].K).G; //green
                                        line[3 * x + 2] = img[x,
y].cmykToRGB(imgcmyk[x, y].C, imgcmyk[x, y].M, imgcmyk[x, y].Y,
imgcmyk[x, y].K).R; //red
                                        break;
                                    }//cmyk
                                case "C":
                                    {
                                        line[3 * x] = img[x,
y].cmykToRGB(imgcmyk[x, y].C, 0, 0, 0).B; //blue
                                        line[3 * x + 1] = img[x,
y].cmykToRGB(imgcmyk[x, y].C, 0, 0, 0).G; //green
                                        line[3 * x + 2] = img[x,
y].cmykToRGB(imgcmyk[x, y].C, 0, 0, 0).R; //red
                                        break;
                                    }//cyan
                                case "M":
                                    {
                                        line[3 * x] = img[x, y].cmykToRGB(0,
imgcmyk[x, y].M, 0, 0).B; //blue
                                        line[3 * x + 1] = img[x,
y].cmykToRGB(0, imgcmyk[x, y].M, 0, 0).G; //green
                                        line[3 * x + 2] = img[x,
y].cmykToRGB(0, imgcmyk[x, y].M, 0, 0).R; //red
                                        break;
                                    }//magenta
                                case "Y":
                                    {
                                        line[3 * x] = img[x, y].cmykToRGB(0,
0, imgcmyk[x, y].Y, 0).B; //blue
                                        line[3 * x + 1] = img[x,
y].cmykToRGB(0, 0, imgcmyk[x, y].Y, 0).G; //green
                                        line[3 * x + 2] = img[x,
y].cmykToRGB(0, 0, imgcmyk[x, y].Y, 0).R; //red
                                        break;
                                    }//yellow
                                case "K":
```

```csharp
                                {
                                    line[3 * x] = img[x, y].cmykToRGB(0,
0, 0, imgcmyk[x, y].K).B; //blue
                                    line[3 * x + 1] = img[x,
y].cmykToRGB(0, 0, 0, imgcmyk[x, y].K).G; //green
                                    line[3 * x + 2] = img[x,
y].cmykToRGB(0, 0, 0, imgcmyk[x, y].K).R; //red
                                    break;
                                }//key
                            case "YUV":
                                {
                                    line[3 * x] = img[x,
y].yuvToRGB(imgyuv[x, y].Yy, imgyuv[x, y].U, imgyuv[x, y].Vv).B; //blue
                                    line[3 * x + 1] = img[x,
y].yuvToRGB(imgyuv[x, y].Yy, imgyuv[x, y].U, imgyuv[x, y].Vv).G; //green
                                    line[3 * x + 2] = img[x,
y].yuvToRGB(imgyuv[x, y].Yy, imgyuv[x, y].U, imgyuv[x, y].Vv).R; //red
                                    break;
                                }//yuv
                            case "Yy":
                                {
                                    line[3 * x] = img[x,
y].yuvToRGB(imgyuv[x, y].Yy, 128, 128).B; //blue
                                    line[3 * x + 1] = img[x,
y].yuvToRGB(imgyuv[x, y].Yy, 128, 128).G; //green
                                    line[3 * x + 2] = img[x,
y].yuvToRGB(imgyuv[x, y].Yy, 128, 128).R; //red
                                    break;
                                }
                            case "U":
                                {
                                    line[3 * x] = img[x, y].yuvToRGB(128,
imgyuv[x, y].U, 128).B; //blue
                                    line[3 * x + 1] = img[x,
y].yuvToRGB(128, imgyuv[x, y].U, 128).G; //green
                                    line[3 * x + 2] = img[x,
y].yuvToRGB(128, imgyuv[x, y].U, 128).R; //red
                                    break;
                                }
                            case "Vv":
                                {
                                    line[3 * x] = img[x, y].yuvToRGB(128,
128, imgyuv[x, y].Vv).B; //blue
                                    line[3 * x + 1] = img[x,
y].yuvToRGB(128, 128, imgyuv[x, y].Vv).G; //green
                                    line[3 * x + 2] = img[x,
y].yuvToRGB(128, 128, imgyuv[x, y].Vv).R; //red
                                    break;
                                }
                        } //switch
                    }
                    ptr = bmpData.Scan0 + y * bmpData.Stride;
                    Marshal.Copy(line, 0, ptr, line.Length);
                }
                bmp.UnlockBits(bmpData);
                hist2.readHistogram(imgnew, imghsv);
                watchdraw.Stop();
                var elapsedMs = watchdraw.ElapsedMilliseconds;
                Console.WriteLine("Image draw time " + elapsedMs);
                return bmp;
            }
            else
            {
                watchdraw.Stop();
```

```csharp
                var elapsedMs = watchdraw.ElapsedMilliseconds;
                Console.WriteLine("Image draw time " + elapsedMs);
                return null;
            }
        }
    }
}
```

PixelClassRGB.cs

```csharp
using System;

namespace IKAA_171rdb115_2
{
    public class PixelClassRGB
    {

        public byte R; //red
        public byte G; //green
        public byte B; //blue
        public byte I; //intensity

        public PixelClassRGB()
        {
            R = 0;
            G = 0;
            B = 0;
            I = 0;
        }
        public PixelClassRGB(byte r, byte g, byte b)
        {
            R = r;
            G = g;
            B = b;
            I = (byte)Math.Round(0.0722f * b + 0.7152f * g + 0.2126f * r);
        }

        public PixelClassRGB hsvToRGB(int h, byte s, byte v)
        {
            byte r = 0;
            byte g = 0;
            byte b = 0;

            int Hi = Convert.ToInt32(h / 60);
            byte Vmin = Convert.ToByte((255 - s) * v / 255);
            int a = Convert.ToInt32((v - Vmin) * (h % 60) / 60);
            byte Vinc = Convert.ToByte(Vmin + a);
            byte Vdec = Convert.ToByte(v - a);

            switch (Hi)
            {
                case 0: { r = v; g = Vinc; b = Vmin; break; }
                case 1: { r = Vdec; g = v; b = Vmin; break; }
                case 2: { r = Vmin; g = v; b = Vinc; break; }
                case 3: { r = Vmin; g = Vdec; b = v; break; }
                case 4: { r = Vinc; g = Vmin; b = v; break; }
                case 5: { r = v; g = Vmin; b = Vdec; break; }
            }
            PixelClassRGB rgbPix = new PixelClassRGB(r, g, b);
            return rgbPix;
        }

        public PixelClassRGB cmykToRGB(float c, float m, float y, float k)
        {
            byte r, g, b;
            r = Convert.ToByte(255 * (1 - c) * (1 - k));
            g = Convert.ToByte(255 * (1 - m) * (1 - k));
            b = Convert.ToByte(255 * (1 - y) * (1 - k));
            PixelClassRGB rgbPix = new PixelClassRGB(r, g, b);
            return rgbPix;
        }

        public PixelClassRGB yuvToRGB(float Yy, float U, float Vv)
```

```csharp
        {
            byte r, g, b;
            r = Convert.ToByte(ClampYUV(Yy + 1.13983f * (Vv - 128f)));
            g = Convert.ToByte(ClampYUV(Yy - 0.39465f * (U - 128f) - 0.58060f * (Vv
- 128f)));
            b = Convert.ToByte(ClampYUV(Yy + 2.03211f * (U - 128f)));/*
            byte R = Convert.ToByte(r);
            byte G = Convert.ToByte(g);
            byte B = Convert.ToByte(b);*/
            PixelClassRGB rgbPix = new PixelClassRGB(r, g, b);
            return rgbPix;
        }

        private static float ClampYUV(float value)
        {
            if (value > 255)
            {
                value = 255;
            }else if (value < 0)
            {
                value = 0;
            }
            return value;
        }
    }
}
```

PixelClassHSV.cs

```csharp
using System;

namespace IKAA_171rdb115_2
{
    public class PixelClassHSV
    {
        public int H; //hue
        public byte S; //saturation
        public byte V; //value

        public PixelClassHSV()
        {
            H = 0;
            S = 0;
            V = 0;
        }

        public PixelClassHSV(byte r, byte g, byte b)
        {
            int MAX = Math.Max(r, Math.Max(g, b));
            int MIN = Math.Min(r, Math.Min(g, b));
            if (MAX == MIN) { H = 0; }
            else if ((MAX == r) && (g >= b)) { H = 60 * (g - b) / (MAX - MIN); }
            else if ((MAX == r) && (g < b)) { H = 60 * (g - b) / (MAX - MIN) + 360;
}

            else if (MAX == g) { H = 60 * (b - r) / (MAX - MIN) + 120; }
            else { H = 60 * (r - g) / (MAX - MIN) + 240; };
            if (H == 360) { H = 0; }
            if (MAX == 0) { S = 0; }
            else { S = Convert.ToByte(255 * (1 - ((float)MIN / MAX))); }
            V = (byte)(MAX);
        }
    }
}
```

PixelClassCMYK.cs

```csharp
using System;

namespace IKAA_171rdb115_2
{
    public class PixelClassCMYK
    {
        public float C;
        public float M;
        public float Y;
        public float K;

        public PixelClassCMYK()
        {
            C = 0;
            M = 0;
            Y = 0;
            K = 0;
        }

        public PixelClassCMYK(byte r, byte g, byte b)
        {
            float Ri = r / 255f;
            float Gi = g / 255f;
            float Bi = b / 255f;
            K = ClampCMYK(1 - Math.Max(Ri, Math.Max(Gi, Bi)));
            C = ClampCMYK((1 - Ri - K) / (1 - K));
            M = ClampCMYK((1 - Gi - K) / (1 - K));
            Y = ClampCMYK((1 - Bi - K) / (1 - K));
        }

        private static float ClampCMYK(float value)
        {
            if(value < 0 || float.IsNaN(value))
            {
                value = 0;
            }
            return value;
        }
    }
}
```

PixelClassYUV.cs

```csharp
using System;

namespace IKAA_171rdb115_2
{
    public class PixelClassYUV
    {
        public float Yy;
        public float U;
        public float Vv;

        public PixelClassYUV()
        {
            Yy = 0;
            U = 0;
            Vv = 0;
        }

        public PixelClassYUV(byte r, byte g, byte b)
        {
            Yy = (float) ((0.299 * r) + (0.587 * g) + (0.114 * b));
            U = (float) ((-0.14713 * r) - (0.28886 * g) + (0.436 * b) +128);
            Vv = (float) ((0.615 * r) - (0.51499 * g) - (0.10001 * b) +128);
        }

    }
}
```

Histogram.cs

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
using System.Drawing.Imaging;
using System.Windows.Forms.DataVisualization.Charting;

namespace IKAA_171rdb115_2
{
    public class Histogram
    {
        public int[] hR; //red
        public int[] hG; //green
        public int[] hB; //blue
        public int[] hI; //intensity
        public int[] hHr;
        public int[] hHg;
        public int[] hHb;
        public int[] hS;
        public int[] hV;

        public Histogram()
        {
            hR = new int[257];
            hG = new int[257];
            hB = new int[257];
            hI = new int[257];
            hHr = new int[257];
            hHg = new int[257];
            hHb = new int[257];
            hS = new int[257];
            hV = new int[257];
        }

        public void eraseHistogram()
        {
            for(int i = 0; i<256; i++)
            {
                hR[i] = 0;
                hG[i] = 0;
                hB[i] = 0;
                hI[i] = 0;
                hHr[i] = 0;
                hHg[i] = 0;
                hHb[i] = 0;
                hS[i] = 0;
                hV[i] = 0;
            }
        }

        public void readHistogram(PixelClassRGB[,] imgArray, PixelClassHSV[,] imgArrayHSV)
        {
            eraseHistogram();
            for(int x=0; x<imgArray.GetLength(0); x++)
            {
                for(int y =0; y<imgArray.GetLength(1); y++)
                {
                    hR[imgArray[x, y].R]++;
                    hG[imgArray[x, y].G]++;
                    hB[imgArray[x, y].B]++;
```

```csharp
                        hI[imgArray[x, y].I]++;
                        hHr[imgArray[x, y].hsvToRGB(imgArrayHSV[x, y].H, 255, 255).R]++;
                        hHg[imgArray[x, y].hsvToRGB(imgArrayHSV[x, y].H, 255, 255).G]++;
                        hHb[imgArray[x, y].hsvToRGB(imgArrayHSV[x, y].H, 255, 255).B]++;
                        hS[imgArrayHSV[x, y].S]++;
                        hV[imgArrayHSV[x, y].V]++;
                    }
                }

            for(int i=0; i<256; i++)
            {
                hR[256] = Math.Max(hR[i], hR[256]);
                hG[256] = Math.Max(hG[i], hG[256]);
                hB[256] = Math.Max(hB[i], hB[256]);
                hI[256] = Math.Max(hI[i], hI[256]);
                hHr[256] = Math.Max(hHr[i], hHr[256]);
                hHg[256] = Math.Max(hHg[i], hHg[256]);
                hHb[256] = Math.Max(hHb[i], hHb[256]);
            }
        }

        public void drawHistogram(Chart chart, string Channels)
        {
            chart.Series.Clear();
            chart.ChartAreas.Clear();
            chart.ChartAreas.Add("ChartArea");
            chart.ChartAreas["ChartArea"].BackColor = Color.Transparent;

            switch (Channels)
            {
                case "RGB":
                    {
                        chart.Series.Add("R");
                        chart.Series["R"].Color = Color.Red;
                        chart.Series.Add("G");
                        chart.Series["G"].Color = Color.Green;
                        chart.Series.Add("B");
                        chart.Series["B"].Color = Color.Blue;
                        chart.Series.Add("I");
                        chart.Series["I"].Color = Color.Gray;
                        chart.ChartAreas["ChartArea"].AxisX.Maximum = 255;
                        chart.ChartAreas["ChartArea"].AxisX.Minimum = 0;
                        for (int i = 0; i < 256; i++)
                        {
                            chart.Series["R"].Points.AddXY(i, hR[i]);
                            chart.Series["G"].Points.AddXY(i, hG[i]);
                            chart.Series["B"].Points.AddXY(i, hB[i]);
                            chart.Series["I"].Points.AddXY(i, hI[i]);
                        }
                        break;
                    }
                case "R":
                    {
                        chart.Series.Add("R");
                        chart.Series["R"].Color = Color.Red;
                        chart.ChartAreas["ChartArea"].AxisX.Maximum = 255;
                        chart.ChartAreas["ChartArea"].AxisX.Minimum = 0;
                        for (int i = 0; i < 256; i++)
                        {
                            chart.Series["R"].Points.AddXY(i, hR[i]);
                        }
                        break;
                    }
                case "G":
                    {
```

```csharp
                    chart.Series.Add("G");
                    chart.Series["G"].Color = Color.Green;
                    chart.ChartAreas["ChartArea"].AxisX.Maximum = 255;
                    chart.ChartAreas["ChartArea"].AxisX.Minimum = 0;
                    for (int i = 0; i < 256; i++)
                    {
                        chart.Series["G"].Points.AddXY(i, hG[i]);
                    }
                    break;
                }
            case "B":
                {
                    chart.Series.Add("B");
                    chart.Series["B"].Color = Color.Blue;
                    chart.ChartAreas["ChartArea"].AxisX.Maximum = 255;
                    chart.ChartAreas["ChartArea"].AxisX.Minimum = 0;
                    for (int i = 0; i < 256; i++)
                    {
                        chart.Series["B"].Points.AddXY(i, hB[i]);
                    }
                    break;
                }
            case "I":
                {
                    chart.Series.Add("I");
                    chart.Series["I"].Color = Color.Gray;
                    chart.ChartAreas["ChartArea"].AxisX.Maximum = 255;
                    chart.ChartAreas["ChartArea"].AxisX.Minimum = 0;
                    for (int i = 0; i < 256; i++)
                    {
                        chart.Series["I"].Points.AddXY(i, hI[i]);
                    }
                    break;
                }
            case "HSV":
                {
                    chart.Series.Add("H");
                    chart.Series["H"].Color = Color.Red;
                    chart.Series.Add("S");
                    chart.Series["S"].Color = Color.Orange;
                    chart.Series.Add("V");
                    chart.Series["V"].Color = Color.Gray;
                    chart.ChartAreas["ChartArea"].AxisX.Maximum = 255;
                    chart.ChartAreas["ChartArea"].AxisX.Minimum = 0;
                    for(int i = 0; i<256; i++)
                    {
                        chart.Series["H"].Points.AddXY(i, hHr[i]);
                        chart.Series["H"].Points.AddXY(i, hHg[i]);
                        chart.Series["H"].Points.AddXY(i, hHb[i]);
                        chart.Series["H"].Points.AddXY(i, hS[i]);
                        chart.Series["H"].Points.AddXY(i, hV[i]);
                    }
                    break;
                }
            case "H":
                {
                    chart.Series.Add("H");
                    chart.Series["H"].Color = Color.Red;
                    chart.ChartAreas["ChartArea"].AxisX.Maximum = 255;
                    chart.ChartAreas["ChartArea"].AxisX.Minimum = 0;
                    for (int i = 0; i < 256; i++)
                    {
                        chart.Series["H"].Points.AddXY(i, hHr[i]);
                        chart.Series["H"].Points.AddXY(i, hHg[i]);
                        chart.Series["H"].Points.AddXY(i, hHb[i]);
```

```
                }
                break;
            }
        case "S":
            {
                chart.Series.Add("S");
                chart.Series["S"].Color = Color.Orange;
                chart.ChartAreas["ChartArea"].AxisX.Maximum = 255;
                chart.ChartAreas["ChartArea"].AxisX.Minimum = 0;
                for (int i = 0; i < 256; i++)
                {
                    chart.Series["S"].Points.AddXY(i, hS[i]);
                }
                break;
            }
        case "V":
            {
                chart.Series.Add("V");
                chart.Series["V"].Color = Color.Gray;
                chart.ChartAreas["ChartArea"].AxisX.Maximum = 255;
                chart.ChartAreas["ChartArea"].AxisX.Minimum = 0;
                for (int i = 0; i < 256; i++)
                {
                    chart.Series["V"].Points.AddXY(i, hV[i]);
                }
                break;
            }
        }

    }
  }
}
```

Ekrānuzņēmumi

Barba Ilva Cimdiņa, 171RDB115

Color Systems
- ◉ RGB
- ○ HSV
- ○ CMYK
- ○ YUV

Color Channels
- ○ Composite
- ○ Red
- ◉ Green
- ○ Blue
- ○ Intensity

Image Data
Can't read coordinates outside image

RGB
Color

RGB (inversed)
Color

HSV
Values

CMYK
Values

YUV
Values

Open | Invert | Choose color | Save

Barba Ilva Cimdiņa, 171RDB115

Color Systems
- ◉ RGB
- ○ HSV
- ○ CMYK
- ○ YUV

Color Channels
- ○ Composite
- ○ Red
- ○ Green
- ◉ Blue
- ○ Intensity

Image Data
Can't read coordinates outside image

RGB
Color

RGB (inversed)
Color

HSV
Values

CMYK
Values

YUV
Values

Open | Invert | Choose color | Save

Barba Ilva Cimdiņa, 171RDB115

Color Systems
- ◉ RGB
- ○ HSV
- ○ CMYK
- ○ YUV

Color Channels
- ○ Composite
- ○ Red
- ○ Green
- ○ Blue
- ◉ Intensity

Image Data
Can't read coordinates outside image

RGB
Color

RGB (inversed)
Color

HSV
Values

CMYK
Values

YUV
Values

Open | Invert | Choose color | Save