Barba Ilva Cimdiņa

171RDB115

6.praktiskais darbs

Filtru lietošana

Programmas kods

Form1.cs

```
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 openToolStripMenuItem_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"; }
            }
        }
```

```
        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;
```

```
                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.readHistogramHSV(imgData.img,
imgData.imghsv);
                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.readHistogramHSV(imgData.img,
imgData.imghsv);
                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)
            {
                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.readHistogramRGB(imgData.img);
                    imgData.hist2.drawHistogram(chart2, "RGB");
                }
                else if (radioButton2.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("HSV");
                    imgData.hist2.readHistogramHSV(imgData.img,
imgData.imghsv);
                    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.readHistogramRGB(imgData.img);
                    imgData.hist2.drawHistogram(chart2, "R");
                }
                else if (radioButton2.Checked)
```

```
                {
                    pictureBox2.Image = imgData.drawImage("H");
                    imgData.hist2.readHistogramHSV(imgData.img,
imgData.imghsv);
                    imgData.hist2.drawHistogram(chart2, "H");
                }
                else if (radioButton8.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("C");
                }
                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.readHistogramRGB(imgData.img);
                    imgData.hist2.drawHistogram(chart2, "G");
                }
                else if (radioButton2.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("S");
                    imgData.hist2.readHistogramHSV(imgData.img,
imgData.imghsv);
                    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.readHistogramRGB(imgData.img);
                    imgData.hist2.drawHistogram(chart2, "B");
                }
                else if (radioButton2.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("V");
                    imgData.hist2.readHistogramHSV(imgData.img,
imgData.imghsv);
                    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)
            {
                if (radioButton1.Checked)
                {
                    pictureBox2.Image = imgData.drawImage("I");
                    imgData.hist2.readHistogramRGB(imgData.img);
                    imgData.hist2.drawHistogram(chart2, "I");
                }
                else
                {
                    pictureBox2.Image = imgData.drawImage("K");
                }
            }
        }

        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;
        }

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

        private void stretchingToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (imgData.img != null)
            {
                bool isStretch = true;
                int value = 0;
                stretchHistrogram(isStretch, value);
            }
        }

        private void trackBar1_Scroll(object sender, EventArgs e)
        {
```

```
            toolTip1.SetToolTip(trackBar1, trackBar1.Value.ToString() +
"%");
        }

        private void normalizeToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (imgData.img != null)
            {
                bool isStretch = false;
                int value = trackBar1.Value;
                stretchHistrogram(isStretch, value);
            }
        }

        private void stretchHistrogram(bool isStretch, int value)
        {
            if (radioButton1.Checked)
            {
                if (radioButton3.Checked)
                {
                    imgData.hist2.readHistogramRGB(imgData.img);
                    imgData.contrastByHistogram("R", value, isStretch);
                    imgData.contrastByHistogram("G", value, isStretch);
                    imgData.contrastByHistogram("B", value, isStretch);
                    imgData.contrastByHistogram("I", value, isStretch);
                    pictureBox2.Image = imgData.drawImage("StretchRGB");
                    imgData.hist2.readHistogramRGB(imgData.imgnew);
                    imgData.hist2.drawHistogram(chart2, "RGB");
                }
                else if (radioButton4.Checked)
                {
                    imgData.hist2.readHistogramRGB(imgData.img);
                    imgData.contrastByHistogram("R", value, isStretch);
                    pictureBox2.Image = imgData.drawImage("StretchR");
                    imgData.hist2.readHistogramRGB(imgData.imgnew);
                    imgData.hist2.drawHistogram(chart2, "R");
                }
                else if (radioButton5.Checked)
                {
                    imgData.hist2.readHistogramRGB(imgData.img);
                    imgData.contrastByHistogram("G", value, isStretch);
                    pictureBox2.Image = imgData.drawImage("StretchG");
                    imgData.hist2.readHistogramRGB(imgData.imgnew);
                    imgData.hist2.drawHistogram(chart2, "G");
                }
                else if (radioButton6.Checked)
                {
                    imgData.hist2.readHistogramRGB(imgData.img);
                    imgData.contrastByHistogram("B", value, isStretch);
                    pictureBox2.Image = imgData.drawImage("StretchB");
                    imgData.hist2.readHistogramRGB(imgData.imgnew);
                    imgData.hist2.drawHistogram(chart2, "B");
                }
                else if (radioButton7.Checked)
                {
                    imgData.hist2.readHistogramRGB(imgData.img);
                    imgData.contrastByHistogram("I", value, isStretch);
                    pictureBox2.Image = imgData.drawImage("StretchI");
                    imgData.hist2.readHistogramRGB(imgData.imgnew);
                    imgData.hist2.drawHistogram(chart2, "I");
                }
            }
            else if (radioButton2.Checked)
```

```
            {
                if (radioButton3.Checked)
                {
                    imgData.hist2.readHistogramHSV(imgData.img,
imgData.imghsv);
                    imgData.contrastByHistogram("S", value, isStretch);
                    imgData.contrastByHistogram("V", value, isStretch);
                    pictureBox2.Image = imgData.drawImage("StretchHSV");
                    imgData.hist2.readHistogramHSV(imgData.imgnew,
imgData.imghsvnew);
                    imgData.hist2.drawHistogram(chart2, "HSV");
                }
                else if (radioButton5.Checked)
                {
                    imgData.hist2.readHistogramHSV(imgData.img,
imgData.imghsv);
                    imgData.contrastByHistogram("S", value, isStretch);
                    pictureBox2.Image = imgData.drawImage("StretchS");
                    imgData.hist2.readHistogramHSV(imgData.imgnew,
imgData.imghsvnew);
                    imgData.hist2.drawHistogram(chart2, "S");
                }
                else if (radioButton6.Checked)
                {
                    imgData.hist2.readHistogramHSV(imgData.img,
imgData.imghsv);
                    imgData.contrastByHistogram("V", value, isStretch);
                    pictureBox2.Image = imgData.drawImage("StretchV");
                    imgData.hist2.readHistogramHSV(imgData.imgnew,
imgData.imghsvnew);
                    imgData.hist2.drawHistogram(chart2, "V");
                }
            }
        }

        private void filter1blurToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (imgData.img != null)
            {
                imgData.filters = new Filter(); //jauns filtrs
                imgData.filters.filter3x3Blur(); //veidojam filtru
                imgData.filterImage(imgData.filters); //filtrējam attēlu
                imgData.hist2.readHistogramRGB(imgData.img); //noalasam
histogrammu
                imgData.hist2.drawHistogram(chart2, "RGB"); //zīmējam
histogrammu
                radioButton1.Checked = true; //RGB
                radioButton3.Checked = true; //Composite
                pictureBox2.Image = imgData.drawImage("StretchRGB");
//izvadam attēlu
                GC.Collect();
            }
        }

        private void filter2blurToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (imgData.img != null)
            {
                imgData.filters = new Filter(); //jauns filtrs
                imgData.filters.filter3x3Blur2(); //veidojam filtru
                imgData.filterImage(imgData.filters); //filtrējam attēlu
```

```
                imgData.hist2.readHistogramRGB(imgData.img); //noalasam
histogrammu
                imgData.hist2.drawHistogram(chart2, "RGB"); //zīmējam
histogrammu
                radioButton1.Checked = true; //RGB
                radioButton3.Checked = true; //Composite
                pictureBox2.Image = imgData.drawImage("StretchRGB");
//izvadam attēlu
                GC.Collect();
            }
        }

        private void filter3blurToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (imgData.img != null)
            {
                imgData.filters = new Filter(); //jauns filtrs
                imgData.filters.filter3x3Blur3(); //veidojam filtru
                imgData.filterImage(imgData.filters); //filtrējam attēlu
                imgData.hist2.readHistogramRGB(imgData.img); //noalasam
histogrammu
                imgData.hist2.drawHistogram(chart2, "RGB"); //zīmējam
histogrammu
                radioButton1.Checked = true; //RGB
                radioButton3.Checked = true; //Composite
                pictureBox2.Image = imgData.drawImage("StretchRGB");
//izvadam attēlu
                GC.Collect();
            }
        }

        private void filter1sharpenToolStripMenuItem1_Click(object
sender, EventArgs e)
        {
            if (imgData.img != null)
            {
                imgData.filters = new Filter(); //jauns filtrs
                imgData.filters.filter3x3Sharpen(); //veidojam filtru
                imgData.filterImage(imgData.filters); //filtrējam attēlu
                imgData.hist2.readHistogramRGB(imgData.img); //noalasam
histogrammu
                imgData.hist2.drawHistogram(chart2, "RGB"); //zīmējam
histogrammu
                radioButton1.Checked = true; //RGB
                radioButton3.Checked = true; //Composite
                pictureBox2.Image = imgData.drawImage("StretchRGB");
//izvadam attēlu
                GC.Collect();
            }
        }

        private void filter2sharpenToolStripMenuItem1_Click(object
sender, EventArgs e)
        {
            if (imgData.img != null)
            {
                imgData.filters = new Filter(); //jauns filtrs
                imgData.filters.filter3x3Sharpen2(); //veidojam filtru
                imgData.filterImage(imgData.filters); //filtrējam attēlu
                imgData.hist2.readHistogramRGB(imgData.img); //noalasam
histogrammu
                imgData.hist2.drawHistogram(chart2, "RGB"); //zīmējam
histogrammu
```

```
                radioButton1.Checked = true; //RGB
                radioButton3.Checked = true; //Composite
                pictureBox2.Image = imgData.drawImage("StretchRGB");
//izvadam attēlu
                GC.Collect();
            }
        }

        private void filter3sharpenToolStripMenuItem1_Click(object
sender, EventArgs e)
        {
            if (imgData.img != null)
            {
                imgData.filters = new Filter(); //jauns filtrs
                imgData.filters.filter3x3Sharpen3(); //veidojam filtru
                imgData.filterImage(imgData.filters); //filtrējam attēlu
                imgData.hist2.readHistogramRGB(imgData.img); //noalasam
histogrammu
                imgData.hist2.drawHistogram(chart2, "RGB"); //zīmējam
histogrammu
                radioButton1.Checked = true; //RGB
                radioButton3.Checked = true; //Composite
                pictureBox2.Image = imgData.drawImage("StretchRGB");
//izvadam attēlu
                GC.Collect();
            }
        }

        private void median3x3ToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Bitmap sourceBitmap = (Bitmap)pictureBox1.Image.Clone();
            imgData.filters = new Filter(); //jauns filtrs
            imgData.filters.MedianFilter(sourceBitmap, imgData.imgnew, 3,
0, false);
            pictureBox2.Image = imgData.drawImage("StretchRGB");
//izvadam attēlu
            imgData.hist2.readHistogramRGB(imgData.imgnew); //noalasam
histogrammu
            imgData.hist2.drawHistogram(chart2, "I"); //zīmējam
histogrammu
            radioButton1.Checked = true; //RGB
            radioButton3.Checked = true; //Composite

        }

        private void median5x5ToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Bitmap sourceBitmap = (Bitmap)pictureBox1.Image.Clone();
            imgData.filters = new Filter(); //jauns filtrs
            imgData.filters.MedianFilter(sourceBitmap, imgData.imgnew, 5,
0, false);
            pictureBox2.Image = imgData.drawImage("StretchRGB");
//izvadam attēlu
            imgData.hist2.readHistogramRGB(imgData.imgnew); //noalasam
histogrammu
            imgData.hist2.drawHistogram(chart2, "I"); //zīmējam
histogrammu
            radioButton1.Checked = true; //RGB
            radioButton3.Checked = true; //Composite

        }
```

```
        private void median7x7ToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Bitmap sourceBitmap = (Bitmap)pictureBox1.Image.Clone();
            imgData.filters = new Filter(); //jauns filtrs
            imgData.filters.MedianFilter(sourceBitmap, imgData.imgnew, 7,
0, false);
            pictureBox2.Image = imgData.drawImage("StretchRGB");
//izvadam attēlu
            imgData.hist2.readHistogramRGB(imgData.imgnew); //noalasam
histogrammu
            imgData.hist2.drawHistogram(chart2, "I"); //zīmējam
histogrammu
            radioButton1.Checked = true; //RGB
            radioButton3.Checked = true; //Composite

        }
    }
}
```

imgData.cs

```
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 PixelClassHSV[,] imghsvnew;
        public Histogram hist1; //original image
        public Histogram hist2; //edited image
        public Filter filters;

        ~imgData()
        {
            img = null;
            imghsv = null;
            imgcmyk = null;
            imgyuv = null;
            imgnew = null;
            imghsvnew = null;
            hist1 = null;
            hist2 = null;
            //filters = 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];
            imghsvnew = 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();
            //filters = new Filter();
            //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);
                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);
                    imghsvnew[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.readHistogramHSV(img, imghsv);
            hist2.readHistogramHSV(imgnew, imghsv);
            watchread.Stop();
            var elapsedMs = watchread.ElapsedMilliseconds;
            Console.WriteLine("Image Read time: " + elapsedMs);
        }

        public void filterImage(Filter f)
        {
            if (img != null)
            {
                for (int x = 1; x < img.GetLength(0) - 1; x++)
                {
                    for (int y = 1; y < img.GetLength(1) - 1; y++)
                    {
                        int r = 0;
                        int g = 0;
                        int b = 0;
                        int i = 0;

                        for (int fi = 0; fi < 3; fi++)
                        {
                            for (int fj = 0; fj < 3; fj++)
                            { //attēla pikseļu reizināšana ar filtra
elementiem
                                r += img[x + fi - 1, y + fj - 1].R *
f.F[fi, fj];
                                g += img[x + fi - 1, y + fj - 1].G *
f.F[fi, fj];
                                b += img[x + fi - 1, y + fj - 1].B *
f.F[fi, fj];
                                i += img[x + fi - 1, y + fj - 1].I *
f.F[fi, fj];
                            }
                        }

                        // izkaitļojam koeficientus katram kanālam
                        r = Math.Max(0, Math.Min(255, r /= f.K));
                        g = Math.Max(0, Math.Min(255, g /= f.K));
```

```
                        b = Math.Max(0, Math.Min(255, b /= f.K));
                        i = Math.Max(0, Math.Min(255, i /= f.K));

                        //piešķiram jaunas vērtības
                        imgnew[x, y].R = (byte)r;
                        imgnew[x, y].G = (byte)g;
                        imgnew[x, y].B = (byte)b;
                        imgnew[x, y].I = (byte)i;
                    }
                }
            }
        }

        public void contrastByHistogram(string mode, int value, bool
isStretch)
        {
            int[] hRGBI = new int[257];
            if (mode == "R") { hRGBI = hist2.hR; }
            else if (mode == "G") { hRGBI = hist2.hG; }
            else if (mode == "B") { hRGBI = hist2.hB; }
            else if (mode == "I") { hRGBI = hist2.hI; }
            else if (mode == "S") { hRGBI = hist2.hS; }
            else if (mode == "V") { hRGBI = hist2.hV; }
            int dBegin, dEnd;
            if (isStretch)
            {
                dBegin = hist2.FindFirst(hRGBI, 0);
                dEnd = hist2.FindLast(hRGBI, 0);
            }
            else
            {
                dBegin = hist2.FindFirst(hRGBI, value);
                dEnd = hist2.FindLast(hRGBI, value);
            }

            int dOriginal = dEnd - dBegin;
            int dDesired = 255;
            double k = dDesired / (double)dOriginal;
            for (int x = 0; x < imgnew.GetLength(0); x++)
            {
                for (int y = 0; y < imgnew.GetLength(1); y++)
                {
                    if (mode == "I")
                    {
                        imgnew[x, y].I = (byte)Math.Min(255, Math.Max(0,
k * (img[x, y].I - dBegin)));
                    }
                    else if (mode == "R")
                    {
                        imgnew[x, y].R = (byte)Math.Min(255, Math.Max(0,
k * (img[x, y].R - dBegin)));
                    }
                    else if (mode == "G")
                    {
                        imgnew[x, y].G = (byte)Math.Min(255, Math.Max(0,
k * (img[x, y].G - dBegin)));
                    }
                    else if (mode == "B")
                    {
                        imgnew[x, y].B = (byte)Math.Min(255, Math.Max(0,
k * (img[x, y].B - dBegin)));
                    }
                    else if (mode == "S")
                    {
```

```
                        imghsvnew[x, y].S = (byte)Math.Min(255,
Math.Max(0, k * (imghsv[x, y].S - dBegin)));
                    }
                    else if (mode == "V")
                    {
                        imghsvnew[x, y].V = (byte)Math.Min(255,
Math.Max(0, k * (imghsv[x, y].V - dBegin)));
                    }
                }
            }

        }

        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];
                                    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 "StretchRGB":
                                {
                                    line[3 * x] = imgnew[x, y].B; //blue
                                    line[3 * x + 1] = imgnew[x, y].G;
//green
                                    line[3 * x + 2] = imgnew[x, y].R;
//red
                                    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 "StretchR":
                                {
                                    line[3 * x] = 0; //blue
                                    line[3 * x + 1] = 0; //green
                                    line[3 * x + 2] = imgnew[x, y].R;
//red
                                    break;
                                }
                            case "StretchG":
                                {
                                    line[3 * x] = 0; //blue
                                    line[3 * x + 1] = imgnew[x, y].G;
//green
                                    line[3 * x + 2] = 0; //red
                                    break;
                                }
                            case "StretchB":
                                {
                                    line[3 * x] = imgnew[x, y].I; //blue
                                    line[3 * x + 1] = 0; //green
                                    line[3 * x + 2] = 0; //red
                                    break;
                                }
                            case "StretchI":
                                {
```

```
                                    line[3 * x] = imgnew[x, y].I; //blue
                                    line[3 * x + 1] = imgnew[x, y].I;
//green
                                    line[3 * x + 2] = imgnew[x, y].I;
//red
                                    break;
                                }
                            case "StretchHSV":
                                {
                                    line[3 * x] = img[x,
y].hsvToRGB(imghsv[x, y].H, imghsvnew[x, y].S, imghsvnew[x, y].V).B;
//blue
                                    line[3 * x + 1] = img[x,
y].hsvToRGB(imghsv[x, y].H, imghsvnew[x, y].S, imghsvnew[x, y].V).G;
//green
                                    line[3 * x + 2] = img[x,
y].hsvToRGB(imghsv[x, y].H, imghsvnew[x, y].S, imghsvnew[x, y].V).R;
//red
                                    break;
                                }
                            case "StretchS":
                                {
                                    line[3 * x] = imghsvnew[x, y].S;
//blue
                                    line[3 * x + 1] = imghsvnew[x, y].S;
//green
                                    line[3 * x + 2] = imghsvnew[x, y].S;
//red
                                    break;
                                }
                            case "StretchV":
                                {
                                    line[3 * x] = imghsvnew[x, y].V;
//blue
                                    line[3 * x + 1] = imghsvnew[x, y].V;
//green
                                    line[3 * x + 2] = imghsvnew[x, y].V;
//red
                                    break;
                                }
                            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
                                    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":
                                {
                                    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.readHistogramHSV(imgnew, imghsv);
                watchdraw.Stop();
                var elapsedMs = watchdraw.ElapsedMilliseconds;
                Console.WriteLine("Image draw time " + elapsedMs);
                return bmp;
            }
            else
```

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

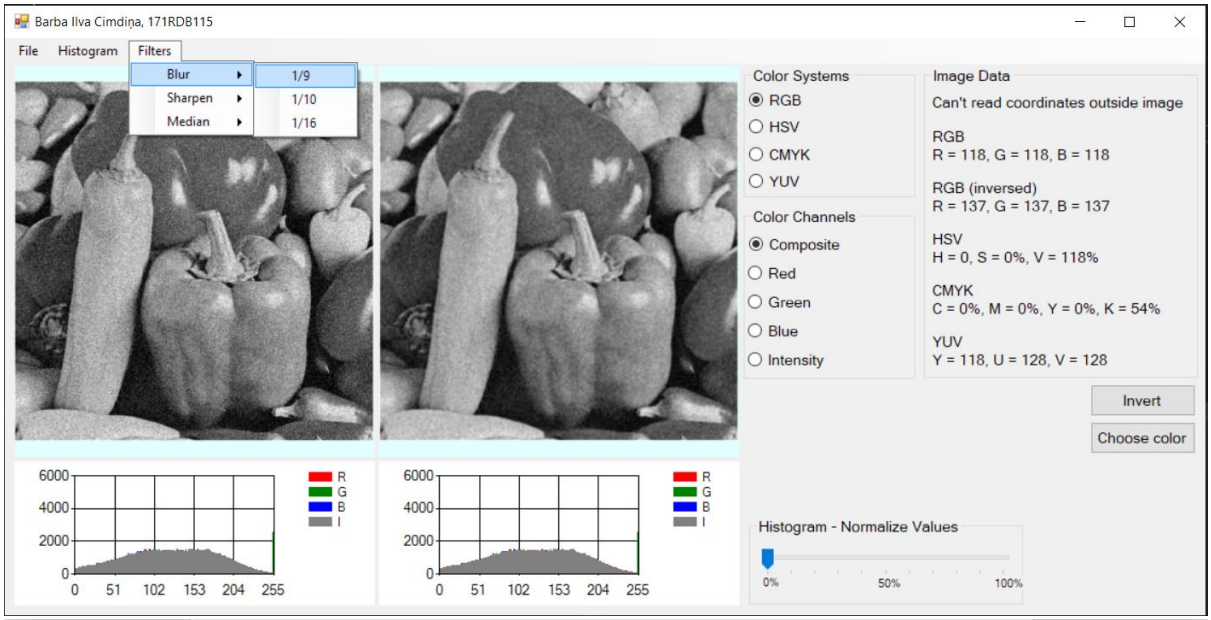
Filter.cs

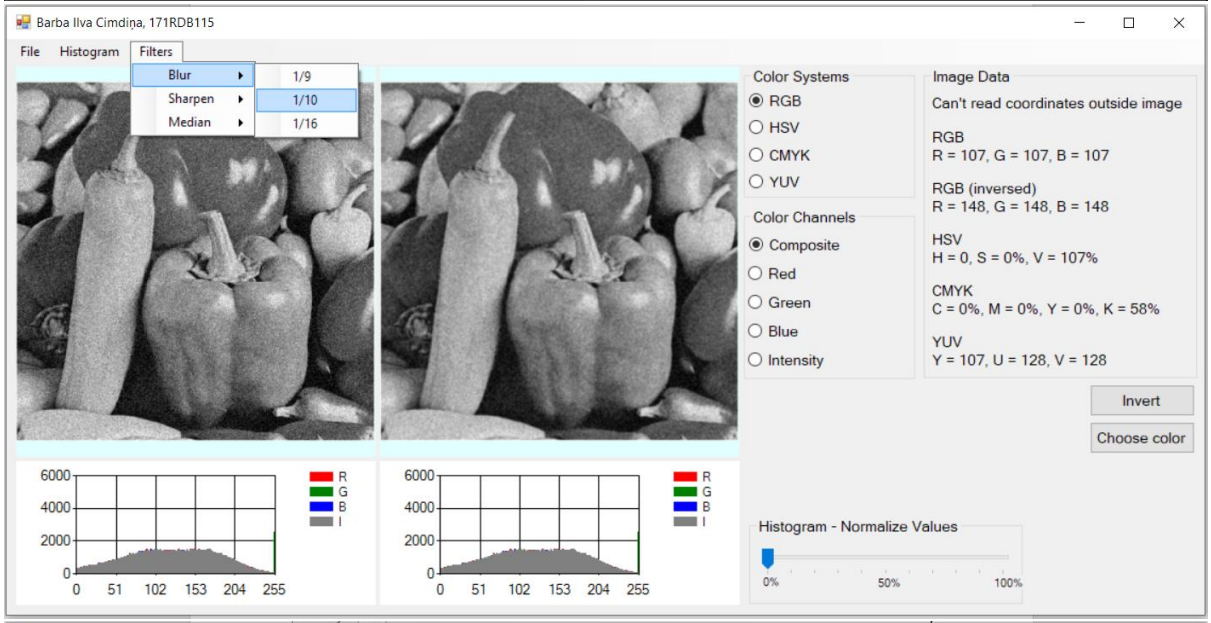
```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;

namespace IKAA_171rdb115_2
{
    public class Filter
    {
        public int[,] F; //vektors ar filtra vērtībam
        public int K; // koefficients
        public Filter()
        {
            F = new int[3, 3]; //veidojam jaunu filtru
        }
        public int calculateCoefficient(int[,] f)
        //izskaitļojam koefficientus
        {
            int k = 0;
            for (int i = 0; i < f.GetLength(0); i++)
            {
                for (int j = 0; j < f.GetLength(1); j++)
                {
                    k += f[i, j]; //saskaitam filtra elementu summu
                }
            }
            return k;
        }
        public void filter3x3Blur() //Pirmais blur
        {
            F = new int[,] { { 1, 1, 1 }, { 1, 1, 1 }, { 1, 1, 1 } };
//filtrs
            K = calculateCoefficient(F); //izskaitļojam koefficientu
        }

        public void filter3x3Blur2() //Otrais blur
        {
            F = new int[,] { { 1, 1, 1 }, { 1, 2, 1 }, { 1, 1, 1 } };
            K = calculateCoefficient(F);
        }

        public void filter3x3Blur3() //Tresais blur
        {
            F = new int[,] { { 1, 2, 1 }, { 2, 4, 2 }, { 1, 2, 1 } };
            K = calculateCoefficient(F);
        }

        public void filter3x3Sharpen() //Pirmais sharpen
        {
            F = new int[,] { { 0, -1, 0 }, { -1, 5, -1 }, { 0, -1, 0 } };
            K = calculateCoefficient(F);
        }

        public void filter3x3Sharpen2() //Otrais sharpen
        {
            F = new int[,] { { -1, -1, -1 }, { -1, 9, -1 }, { -1, -1, -1
} };
            K = calculateCoefficient(F);
        }
```

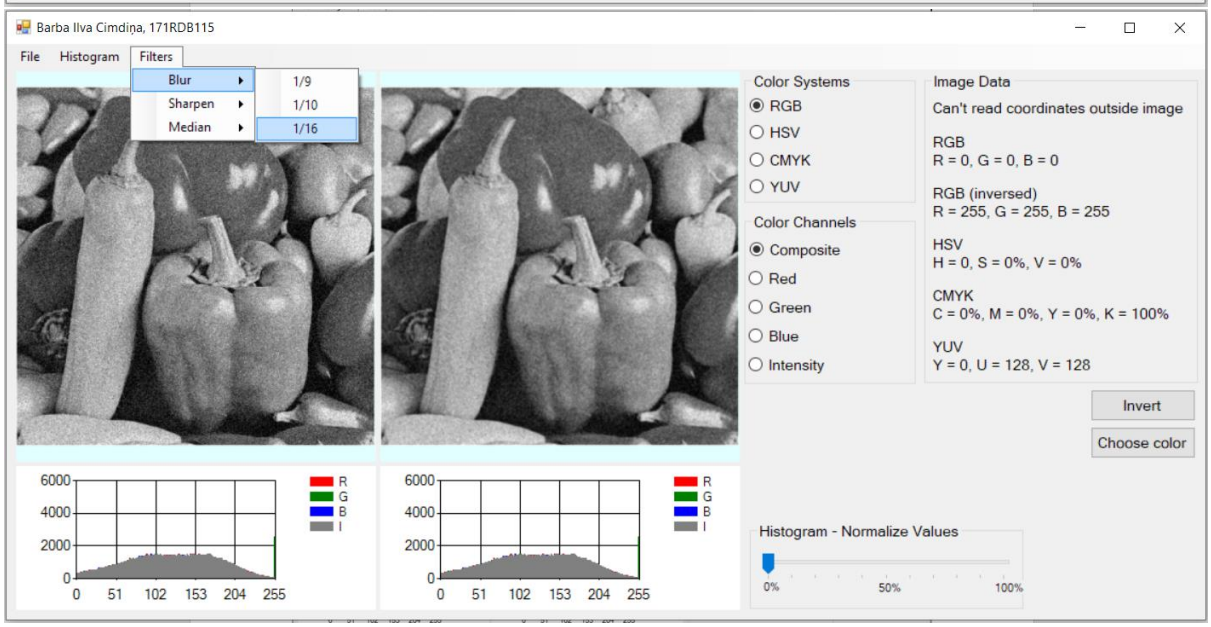
```
        public void filter3x3Sharpen3() //Tresais sharpen
        {
            F = new int[,] { { 0, -2, 0 }, { -2, 9, -2 }, { 0, -2, 0 } };
            K = calculateCoefficient(F);
        }

        public void MedianFilter(
            Bitmap bmp,
            PixelClassRGB[,] img,
            int matrixSize,
            int bias = 0,
            bool grayscale = false)
        {
            BitmapData bmpData = bmp.LockBits(new Rectangle(0, 0,
bmp.Width, bmp.Height),
                ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);
            byte[] pixelBuffer = new byte[bmpData.Stride *
bmpData.Height];
            byte[] resultBuffer = new byte[bmpData.Stride *
bmpData.Height];
            Marshal.Copy(bmpData.Scan0, pixelBuffer, 0,
pixelBuffer.Length);
            bmp.UnlockBits(bmpData);

            if (grayscale == true)
            {
                float rgb = 0;
                for (int k = 0; k < pixelBuffer.Length; k += 4)
                {
                    rgb = pixelBuffer[k] * 0.0722f + pixelBuffer[k + 1] *
0.7152f + pixelBuffer[k + 2] * 0.2126f;

                    pixelBuffer[k] = (byte)rgb;
                    pixelBuffer[k + 1] = pixelBuffer[k];
                    pixelBuffer[k + 2] = pixelBuffer[k];
                    pixelBuffer[k + 3] = 255;
                }
            }

            int filterOffset = (matrixSize - 1) / 2;
            int calcOffset = 0;
            int byteOffset = 0;

            List<int> neighbourPixels = new List<int>();
            byte[] middlePixel;

            for (int offsetY = filterOffset; offsetY < bmp.Height -
filterOffset; offsetY++)
            {
                for (int offsetX = filterOffset; offsetX < bmp.Width -
filterOffset; offsetX++)
                {
                    byteOffset = offsetY * bmpData.Stride + offsetX * 4;
                    neighbourPixels.Clear();

                    for (int filterY = -filterOffset; filterY <=
filterOffset; filterY++)
                    {
                        for (int filterX = -filterOffset; filterX <=
filterOffset; filterX++)
                        {
                            calcOffset = byteOffset + (filterX * 4) +
(filterY * bmpData.Stride);
```

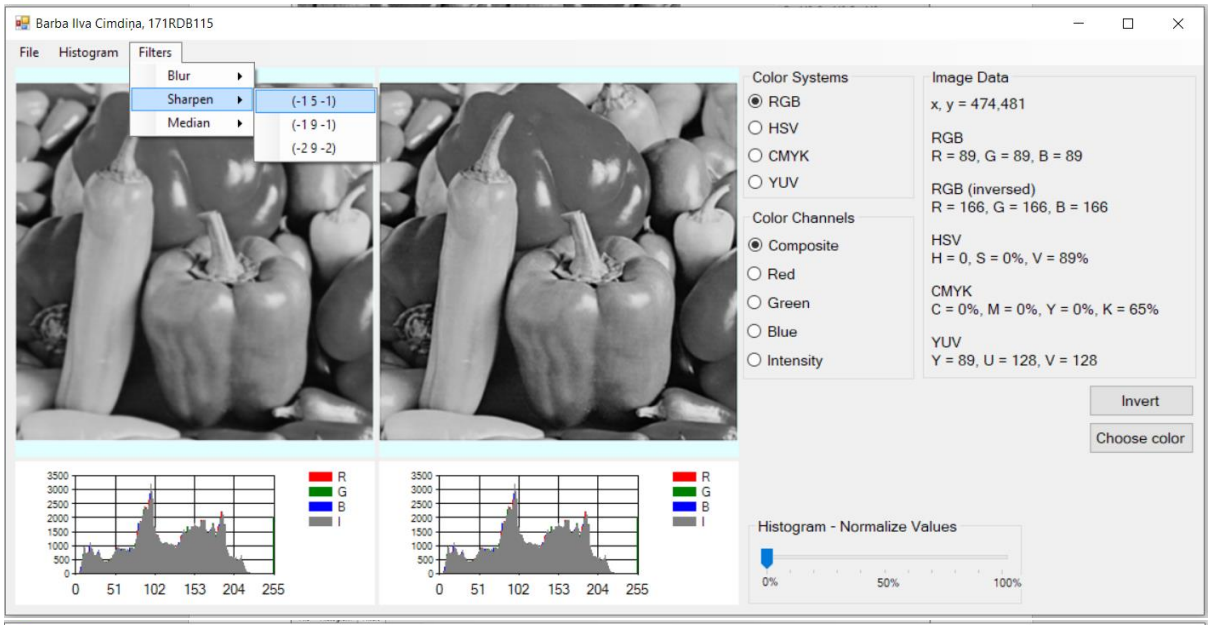
```
neighbourPixels.Add(BitConverter.ToInt32(pixelBuffer, calcOffset));
                        }
                    }

                    neighbourPixels.Sort();
                    middlePixel =
BitConverter.GetBytes(neighbourPixels[filterOffset]);

                    resultBuffer[byteOffset] = middlePixel[0];
                    resultBuffer[byteOffset + 1] = middlePixel[1];
                    resultBuffer[byteOffset + 2] = middlePixel[2];
                    resultBuffer[byteOffset + 3] = middlePixel[3];

                    img[offsetX, offsetY].R = resultBuffer[byteOffset +
2];
                    img[offsetX, offsetY].G = resultBuffer[byteOffset +
1];
                    img[offsetX, offsetY].B = resultBuffer[byteOffset];
                    img[offsetX, offsetY].I = resultBuffer[byteOffset +
3];
                }
            }
        }
    }
}
```

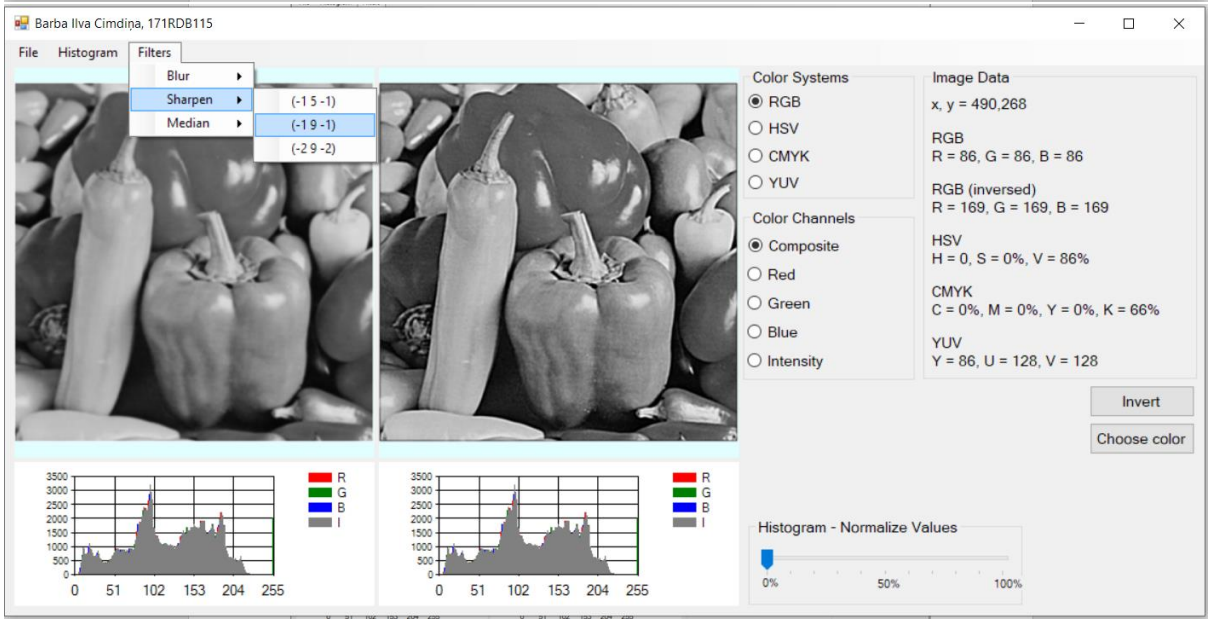
Ekrānuzņēmumi

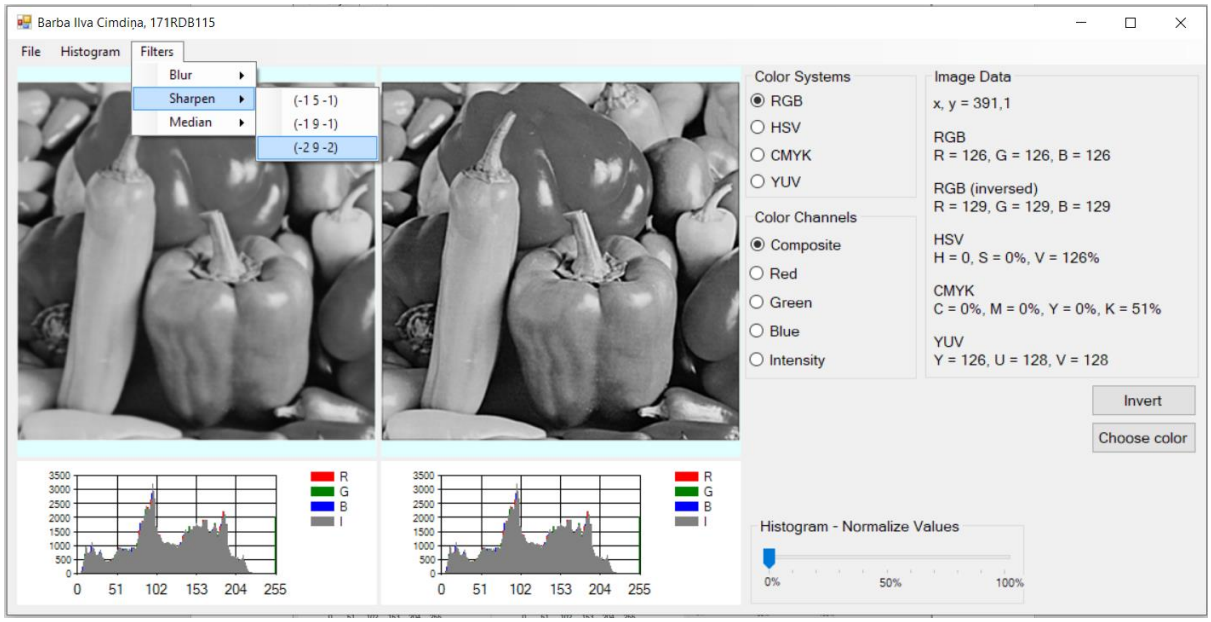


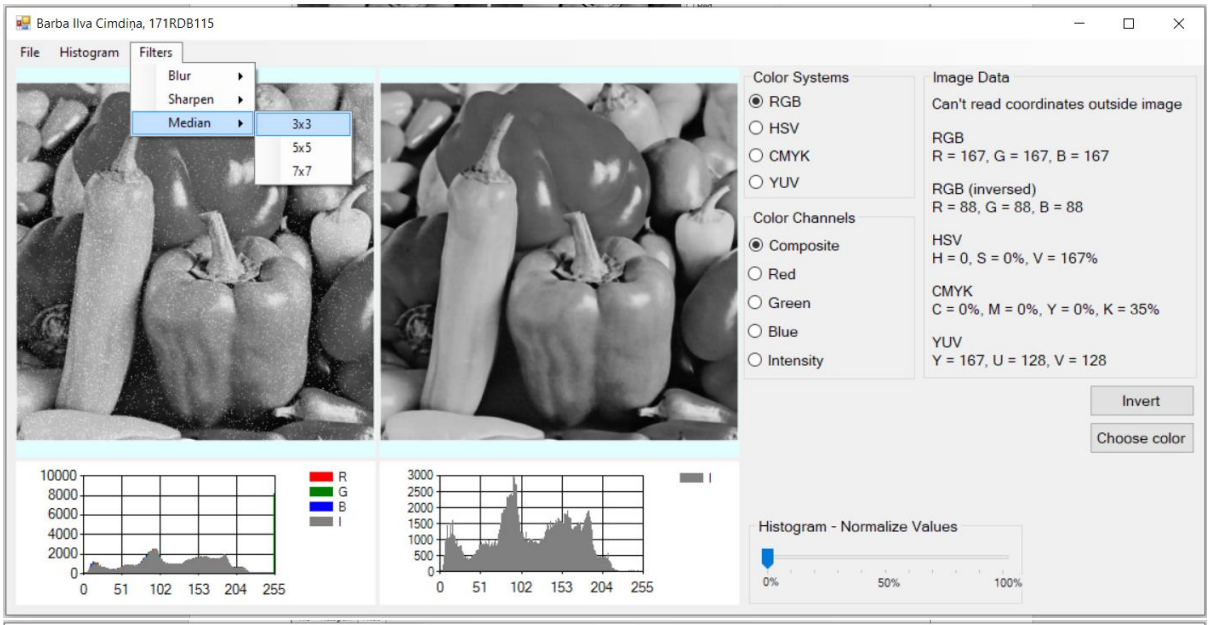


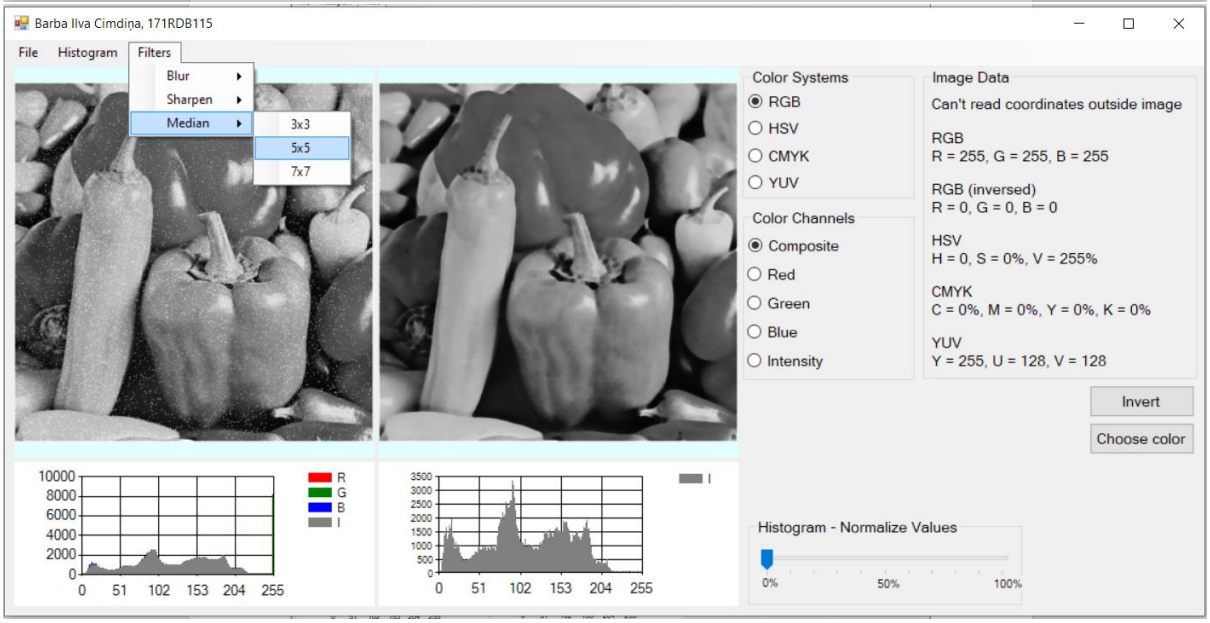


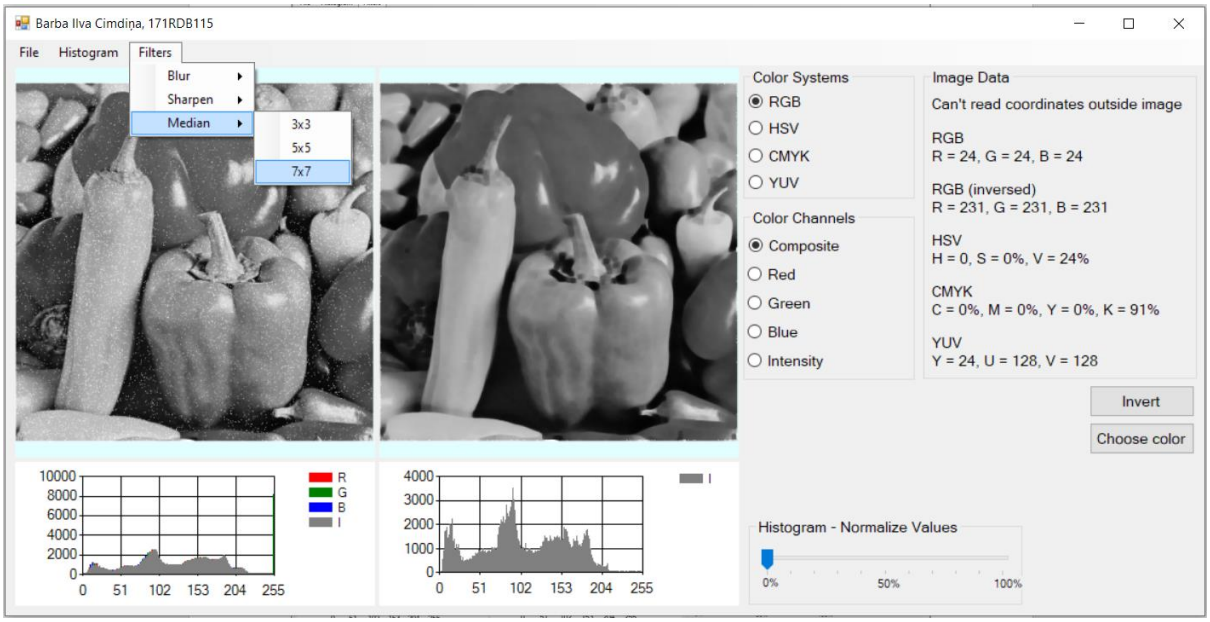

Barba Ilva Cimdiņa, 171RDB115
File
Histogram
Filters
Blur
Sharpen
Median
(-1 5 -1)
(-1 9 -1)
(-2 9 -2)
Color Systems
RGB
HSV
CMYK
YUV
Color Channels
Composite
Red
Green
Blue
Intensity
Image Data
x, y = 474,481
RGB
R = 89, G = 89, B = 89
RGB (inversed)
R = 166, G = 166, B = 166
HSV
H = 0, S = 0%, V = 89%
CMYK
C = 0%, M = 0%, Y = 0%, K = 65%
YUV
Y = 89, U = 128, V = 128
Invert
Choose color
R
G
B
I
0 51 102 153 204 255
Histogram - Normalize Values
0%
50%
100%


Barba Ilva Cimdiņa, 171RDB115
File
Histogram
Filters
Blur
Sharpen
Median
(-1 5 -1)
(-1 9 -1)
(-2 9 -2)
Color Systems
RGB
HSV
CMYK
YUV
Color Channels
Composite
Red
Green
Blue
Intensity
Image Data
x, y = 490,268
RGB
R = 86, G = 86, B = 86
RGB (inversed)
R = 169, G = 169, B = 169
HSV
H = 0, S = 0%, V = 86%
CMYK
C = 0%, M = 0%, Y = 0%, K = 66%
YUV
Y = 86, U = 128, V = 128
Invert
Choose color
R
G
B
I
0 51 102 153 204 255
Histogram - Normalize Values
0%
50%
100%


Barba Ilva Cimdiņa, 171RDB115
File
Histogram
Filters
Blur
Sharpen
Median
(-1 5 -1)
(-1 9 -1)
(-2 9 -2)
Color Systems
RGB
HSV
CMYK
YUV
Color Channels
Composite
Red
Green
Blue
Intensity
Image Data
x, y = 391,1
RGB
R = 126, G = 126, B = 126
RGB (inversed)
R = 129, G = 129, B = 129
HSV
H = 0, S = 0%, V = 126%
CMYK
C = 0%, M = 0%, Y = 0%, K = 51%
YUV
Y = 126, U = 128, V = 128
Invert
Choose color
R
G
B
I
0 51 102 153 204 255
Histogram - Normalize Values
0%
50%
100%


Barba Ilva Cimdiņa, 171RDB115
File
Histogram
Filters
Blur
Sharpen
Median
3x3
5x5
7x7
Color Systems
RGB
HSV
CMYK
YUV
Color Channels
Composite
Red
Green
Blue
Intensity
Image Data
Can't read coordinates outside image
RGB
R = 167, G = 167, B = 167
RGB (inversed)
R = 88, G = 88, B = 88
HSV
H = 0, S = 0%, V = 167%
CMYK
C = 0%, M = 0%, Y = 0%, K = 35%
YUV
Y = 167, U = 128, V = 128
Invert
Choose color
Histogram - Normalize Values
0%
50%
100%


Barba Ilva Cimdiņa, 171RDB115
File
Histogram
Filters
Blur
Sharpen
Median
3x3
5x5
7x7
Color Systems
RGB
HSV
CMYK
YUV
Color Channels
Composite
Red
Green
Blue
Intensity
Image Data
Can't read coordinates outside image
RGB
R = 255, G = 255, B = 255
RGB (inversed)
R = 0, G = 0, B = 0
HSV
H = 0, S = 0%, V = 255%
CMYK
C = 0%, M = 0%, Y = 0%, K = 0%
YUV
Y = 255, U = 128, V = 128
Invert
Choose color
Histogram - Normalize Values
0%
50%
100%


Barba Ilva Cimdiņa, 171RDB115
File
Histogram
Filters
Blur
Sharpen
Median
3x3
5x5
7x7
Color Systems
RGB
HSV
CMYK
YUV
Color Channels
Composite
Red
Green
Blue
Intensity
Image Data
Can't read coordinates outside image
RGB
R = 24, G = 24, B = 24
RGB (inversed)
R = 231, G = 231, B = 231
HSV
H = 0, S = 0%, V = 24%
CMYK
C = 0%, M = 0%, Y = 0%, K = 91%
YUV
Y = 24, U = 128, V = 128
Invert
Choose color
Histogram - Normalize Values
0%
50%
100%