



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота № 4
з дисципліни “Основи програмування”
тема “Бібліотеки і обробка зображень”

Виконав
студент I курсу
групи КП-01

Беліцький Олександр Сергійович
(прізвище, ім'я, по батькові)

варіант № 3

Перевірів
“ ____ ” “ ____ ” 20__ р.
викладач

Гадиняк Руслан Анатолійович
(прізвище, ім'я, по батькові)

Київ 2021

Мета роботи

Реалізувати різні алгоритми редагування зображень.

Розбити проект програми на декілька проектів у одному рішенні з використанням бібліотек класів.

Постановка завдання

Створити консольну програму, що дозволяє виконувати редагування зображень.

Аргументи командного рядка

Приклад аргументів:

``dotnet run {module} ./file.jpg ./out.jpg`` - аргументи обов'язкові і зберігають такий порядок, тільки цих аргументів недостатньо, після них задавати команду редагування і її параметри

- ``{module}`` - ``pixel`` або ``fast``, визначає яким саме модулем редагування змінити зображення.
- ``./file.jpg`` - перший аргумент після ``{module}`` - приклад шляху вхідного зображення
- ``./out.jpg`` - другий аргумент після ``{module}`` - приклад шляху вихідного зображення

Команди:

- Команда отримання частини зображення за координатами:
``crop {width}x{height}+{left}+{top}`` - всі аргументи обов'язкові і зберігають такий порядок.
Приклад: ``dotnet run pixel ./file.jpg ./out.jpg crop 100x100+30+90``
- Команда зеркального відображення по вертикалі:
``FlipVertical``
Приклад: ``dotnet run pixel ./file.jpg ./out.jpg FlipVertical``
- Команда для обнулення червого каналу кольору:
``RemoveRed``
Приклад: ``dotnet run pixel ./file.jpg ./out.jpg RemoveRed``

- Команда для перетворення зображення у відтінки сірого:
`GrayScale`
Приклад: `dotnet run pixel ./file.jpg ./out.jpg GrayScale``
- Команда для застосування розмиття за інтенсивністю:
`Blur {sigma}` - {sigma} від 0 до 20.
Приклад: `dotnet run pixel ./file.jpg ./out.jpg Blur 3``

Вимоги до структури коду

Розбити програму на модулі:

- **модуль обробки аргументів командного рядка** - модуль аналізує задані користувачем аргументи командного рядка і використовує інші модулі.
- **модулі редагування зображень** - містять функції, що на основі вхідного зображення створюють змінене зображення.
 - реалізація за допомогою стандартних функцій, матриць кольору, матриць трансформації або будь-якої графічної бібліотеки
 - реалізація за допомогою піксельних змін

Створити бібліотеки:

- **ProgbaseLab.ImageEditor.Common** - містить контракт модулів редагування
- **ProgbaseLab.ImageEditor.Pixel** - містить модуль редагування зображень пікселями
- **ProgbaseLab.ImageEditor.Fast** - містить модуль редагування зображень стандартними функціями або з використанням інших графічних бібліотек

Підключити і використати бібліотеки у проекті консольної програми.

Аналіз вимог і проектування

Модуль обробки аргументів командного рядка

На основі аргументів командного рядка виконати команду із відповідного модуля.

Обробку кожної команди виконувати у окремій функції.

Після виконання команди вивести у консоль час її виконання і окремо час виконання тільки виклику функції, що виконує редагування зображення (використати для заміру часу **Diagnostics.Stopwatch**) і завершити програму з кодом 0.

Якщо команду виконати неможливо, вивести у потік помилок (**Console.Error**) повідомлення і завершити програму з кодом 1.

Для зв'язки наших бібліотек і консольної програми створимо Рішення:

```
dotnet new sln -o Lab4
```

Створимо Консольний додаток:

```
dotnet new console -o ConsoleApp
```

Для створення класової бібліотеки використаємо команди:

```
dotnet new classlib -o ProgbaseLab.ImageEditor.Common
```

```
dotnet new classlib -o ProgbaseLab.ImageEditor.Fast
```

```
dotnet new classlib -o ProgbaseLab.ImageEditor.Pixel
```

Щоби зв'язати дані проекти додамо їх у рішення:

```
dotnet sln ./Lab4.sln add ./ConsoleApp/ConsoleApp.csproj
```

```
dotnet sln ./Lab4.sln add  
./ProgbaseLab.ImageEditor.Common/ProgbaseLab.ImageEditor.Common.csproj
```

```
dotnet sln ./Lab4.sln add  
./ProgbaseLab.ImageEditor.Fast/ProgbaseLab.ImageEditor.Fast.csproj
```

```
dotnet sln ./Lab4.sln add  
./ProgbaseLab.ImageEditor.Pixel/ProgbaseLab.ImageEditor.Pixel.csproj
```

Для можливості підключення створених бібліотек і використання об'єкта інтерфейсу у Консольному Додатку потрібно додати посилення на ці проєкти:

dotnet add ./ConsoleApp/ConsoleApp.csproj reference ./ProgbaseLab.ImageEditor.Common/ProgbaseLab.ImageEditor.Common.csproj
dotnet add ./ConsoleApp/ConsoleApp.csproj reference ./ProgbaseLab.ImageEditor.Fast/ProgbaseLab.ImageEditor.Fast.csproj
dotnet add ./ConsoleApp/ConsoleApp.csproj reference ./ProgbaseLab.ImageEditor.Pixel/ProgbaseLab.ImageEditor.Pixel.csproj

Для забезпечення виконання Інтерфейсу редактора зображень потрібно передати посилання Інтерфейсу до створених бібліотек:

dotnet add ./ProgbaseLab.ImageEditor.Fast/ProgbaseLab.ImageEditor.Fast.csproj reference ./ProgbaseLab.ImageEditor.Common/ProgbaseLab.ImageEditor.Common.csproj
dotnet add ./ProgbaseLab.ImageEditor.Pixel/ProgbaseLab.ImageEditor. Pixel.csproj reference ./ProgbaseLab.ImageEditor.Common/ProgbaseLab.ImageEditor.Common.csproj

Тексти коду програм

Program.cs

```
using System;

namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {
            try{
                ArgumentProcessor.Run(args);
            }
            catch(Exception ex)
            {
                Console.Error.WriteLine(ex.Message);
                Environment.Exit(1);
            }
            finally
            {
                Console.WriteLine("Tip: Operation was successful");
                Environment.Exit(0);
            }
        }
    }
}
```

ArgumentProcessor.cs

```
using System;
using System.Diagnostics;
using System.Drawing;
using Progbaselab.ImageEditor.Common;

namespace ConsoleApp
{
    static class ArgumentProcessor
    {
        public struct ProgramArguments
        {
            public string module;
            public string inputFile;
            public string outputFile;
            public string operation;
            public string[] otherArgs;
        }

        public static void Run(string[] args)
        {
            ProgramArguments progArgs = ParseArgs(args);
            IRedactingImage redactor = ChooseRedactor(progArgs.module);
            Bitmap inputBit = new Bitmap(progArgs.inputFile);
            switch (progArgs.operation)
            {
                case "crop":
                {
                    ProcessCrop(redactor, progArgs, inputBit);
                    break;
                }
                case "FlipVertical":
                {

```

```

        ProcessFlipVertical(redactor, inputBit, progArgs.outputFile, progArgs.otherArgs);
        break;
    }
    case "RemoveRed":
    {
        ProcessRemoveRed(redactor, inputBit, progArgs.outputFile, progArgs.otherArgs);
        break;
    }
    case "GrayScale":
    {
        ProcessGrayScale(redactor, inputBit, progArgs.outputFile, progArgs.otherArgs);
        break;
    }
    case "Blur":
    {
        ProcessBlur(redactor, progArgs, inputBit);
        break;
    }
}

}

private static void ProcessBlur(IRedactingImage redactor, ProgramArguments progArgs, Bitmap
inputBit)
{
    if (progArgs.otherArgs.Length != 1)
    {
        throw new ArgumentException($"Blur must have one intensity argument, but have
{progArgs.otherArgs.Length}");
    }
    if (Int32.TryParse(progArgs.otherArgs[0], out int sigma) == false)
    {
        throw new ArgumentException($"Blur must have integer intensity argument, but have
{progArgs.otherArgs[0]}");
    }
    Stopwatch watchProcess = new Stopwatch();
    watchProcess.Start();
    Stopwatch watchImage = new Stopwatch();

    watchImage.Start();
    Bitmap outBit = redactor.Blur(inputBit, sigma);
    watchImage.Stop();

    Console.WriteLine($"Image process finished in {watchImage.ElapsedMilliseconds}");
    outBit.Save(progArgs.outputFile);
    watchProcess.Stop();
    Console.WriteLine($"Whole process finished in {watchProcess.ElapsedMilliseconds}");
}

private static void ProcessGrayScale(IRedactingImage redactor, Bitmap bitmap, string outputFile,
string[] otherArgs)
{
    if (otherArgs.Length != 0)
    {
        throw new FormatException($"Incorrect RemoveRed format. Expected other arguments `0` but
have {otherArgs.Length}");
    }
    Stopwatch watchProcess = new Stopwatch();
    watchProcess.Start();
    Stopwatch watchImage = new Stopwatch();
    watchImage.Start();

    Bitmap outBit = redactor.GrayScale(bitmap);

    watchImage.Stop();

    Console.WriteLine($"Image process finished in {watchImage.ElapsedMilliseconds}");
    outBit.Save(outputFile);
    watchProcess.Stop();
    Console.WriteLine($"Whole process finished in {watchProcess.ElapsedMilliseconds}");
}

```

```

        private static void ProcessRemoveRed(IRedatctingImage redactor, Bitmap bitmap, string outputFile,
string[] otherArgs)
        {
            if (otherArgs.Length != 0)
            {
                throw new FormatException($"Incorrect RemoveRed format. Expected other arguments `0` but
have {otherArgs.Length}");
            }
            Stopwatch watchProcess = new Stopwatch();
            watchProcess.Start();
            Stopwatch watchImage = new Stopwatch();
            watchImage.Start();

            Bitmap outBit = redactor.RemoveRed(bitmap);

            watchImage.Stop();

            Console.WriteLine($"Image process finished in {watchImage.ElapsedMilliseconds}");
            outBit.Save(outputFile);
            watchProcess.Stop();
            Console.WriteLine($"Whole process finished in {watchProcess.ElapsedMilliseconds}");
        }

        private static void ProcessCrop(IRedatctingImage redactor, ProgramArguments progArgs, Bitmap
inputBit)
        {
            if (progArgs.otherArgs.Length != 1)
            {
                throw new ArgumentException($"Crop must have one dimensions argument, but have
{progArgs.otherArgs.Length}");
            }
            Stopwatch watchProcess = new Stopwatch();
            watchProcess.Start();
            Stopwatch watchImage = new Stopwatch();

            string cropArguments = progArgs.otherArgs[0];
            Rectangle cropRect = ParseRectangle(cropArguments);

            watchImage.Start();
            Bitmap outBit = redactor.Crop(inputBit, cropRect);
            watchImage.Stop();

            Console.WriteLine($"Image process finished in {watchImage.ElapsedMilliseconds}");
            outBit.Save(progArgs.outputFile);
            watchProcess.Stop();
            Console.WriteLine($"Whole process finished in {watchProcess.ElapsedMilliseconds}");
        }

        private static void ProcessFlipVertical(IRedatctingImage redactor, Bitmap bitmap, string
outputFile, string[] otherArgs)
        {
            if (otherArgs.Length != 0)
            {
                throw new FormatException($"Incorrect FlipVertical format. Expected other arguments `0` but
have {otherArgs.Length}");
            }
            Stopwatch watchProcess = new Stopwatch();
            watchProcess.Start();
            Stopwatch watchImage = new Stopwatch();
            watchImage.Start();

            Bitmap outBit = redactor.FlipVertical(bitmap);

            watchImage.Stop();

            Console.WriteLine($"Image process finished in {watchImage.ElapsedMilliseconds}");
            outBit.Save(outputFile);
            watchProcess.Stop();
            Console.WriteLine($"Whole process finished in {watchProcess.ElapsedMilliseconds}");
        }

```



```

private static IRedatctingImage ChooseRedactor(string module)
{
    if (module == "pixel")
    {
        return new ProgbaseLab.ImageEditor.Pixel.Class1();
    }
    else
    {
        return new ProgbaseLab.ImageEditor.Fast.Class1();
    }
}
private static void ValidateArgumentsLength(int length)
{
    if (length < 4)
    {
        throw new ArgumentException($"Not enough command line arguments. Expected more than 3, got
{length}");
    }
}

private static void ValidateModule(string module)
{
    string[] supportedModules = new string[] { "pixel", "fast" };
    for (int i = 0; i < supportedModules.Length; i++)
    {
        if (supportedModules[i] == module)
        {
            return;
        }
    }
    throw new ArgumentException($"Not supported module {module}");
}

private static void ValidateInputFile(string file)
{
    if (System.IO.File.Exists(file) == false)
    {
        throw new ArgumentException($"File does not exist {file}");
    }
}

private static void ValidateOperation(string operation)
{
    string[] supportedOperations = new string[] { "crop", "FlipVertical", "RemoveRed", "GrayScale",
"Blur"};
    for (int i = 0; i < supportedOperations.Length; i++)
    {
        if (supportedOperations[i] == operation)
        {
            return;
        }
    }
    throw new ArgumentException($"Not supported operation {operation}");
}
public static ProgramArguments ParseArgs(string[] args)
{
    ValidateArgumentsLength(args.Length);
    string module = args[0];
    ValidateModule(module);

    string inputFile = args[1];
    ValidateInputFile(inputFile);

    string outputFile = args[2];
    string operation = args[3];
    ValidateOperation(operation);

    ProgramArguments programArgs = new ProgramArguments();
    programArgs.module = module;
    programArgs.inputFile = inputFile;

```

```

        programArgs.outputFile = outputFile;
        programArgs.operation = operation;
        programArgs.otherArgs = new string[args.Length - 4];
        for (int i = 0; i < programArgs.otherArgs.Length; i++)
        {
            programArgs.otherArgs[i] = args[i + 4];
        }
        return programArgs;
    }

    public static Rectangle ParseRectangle(string rec)
    {
        string[] stringParameters = rec.Split('x', '+');
        if (stringParameters.Length != 4)
        {
            throw new ArgumentException($"Wrong number of parameters. Need 4 but have {stringParameters.Length}");
        }
        int[] parameters = new int[4];
        for (int i = 0; i < stringParameters.Length; i++)
        {
            if (int.TryParse(stringParameters[i], out parameters[i]) == false)
            {
                throw new ArgumentException("Wrong input rectangle parameters");
            }
        }

        return new Rectangle
        {
            Location = new Point(parameters[2], parameters[3]),
            Width = parameters[0],
            Height = parameters[1],
        };
    }
}

```

IRedactingImage.cs

```

using System;
using System.Drawing;

namespace ProgbaseLab.ImageEditor.Common
{
    public interface IRedatctingImage
    {
        Bitmap Crop(Bitmap bmp, Rectangle rec);
        Bitmap FlipVertical(Bitmap bitmap);
        Bitmap RemoveRed(Bitmap bitmap);
        Bitmap GrayScale(Bitmap bitmap);
        Bitmap Blur (Bitmap bitmap, Int32 sigma);
    }
}

```

ProgbaseLab.ImageEditor.Pixel.Class1.cs

```

using System;
using System.Drawing;

namespace ProgbaseLab.ImageEditor.Pixel
{
    public class Class1 : ProgbaseLab.ImageEditor.Common.IRedatctingImage
    {
        public Bitmap Crop(Bitmap bmp, Rectangle rec)
    }
}

```

```

    {
        if (rec.Left < 0 || rec.Left >= bmp.Width)
        {
            throw new Exception("Invalid left");
        }
        if (rec.Right >= bmp.Width)
        {
            throw new Exception("Invalid right");
        }
        if (rec.Top < 0 || rec.Top >= bmp.Height)
        {
            throw new Exception("Invalid right");
        }
        if (rec.Bottom >= bmp.Height)
        {
            throw new Exception("Invalid right");
        }
        Bitmap cropImage = new Bitmap(rec.Width, rec.Height);
        for (int y = 0; y < cropImage.Height; y++)
        {
            for (int x = 0; x < cropImage.Width; x++)
            {
                Color color = bmp.GetPixel(x + rec.Left, y + rec.Top);
                cropImage.SetPixel(x, y, color);
            }
        }
        return cropImage;
    }

    public static double[,] CreateFilterMatrix(int sigma)
    {
        int radius = (int)(sigma * 2);
        int size = 2 * radius + 1;
        double[,] filter = new double[size, size];
        for (int filterX = -radius; filterX <= radius; filterX++)
        {
            for (int filterY = -radius; filterY <= radius; filterY++)
            {
                filter[radius + filterX, radius + filterY] = Gauss(filterX, filterY, sigma);
            }
        }
        return filter;
    }

    private static double Gauss(int x, int y, int sigma)
    {
        double value = 0.0;
        value = (1 / (2 * Math.PI * (double)sigma * (double)sigma)) * (Math.Exp(-((x * x + y * y) / (2
* (double)sigma * (double)sigma))));
        return value;
    }

    public Bitmap Blur(Bitmap bmp, int sigma)
    {
        if (sigma % 2 != 0)
        {
            Bitmap result = new Bitmap(bmp.Width, bmp.Height);
            double[,] filter = CreateFilterMatrix(sigma);
            for (int x = 0; x < bmp.Width; x++)
            {
                for (int y = 0; y < bmp.Height; y++)
                {
                    Color newColor = ApplyFilter(bmp, x, y, filter);
                    result.SetPixel(x, y, newColor);
                }
            }
            return result;
        }
        else throw new ArgumentException("Sigma must be an odd number");
    }

    public static Color ApplyFilter(Bitmap image, int x, int y, double[,] filter)
    {

```

```

double red = 0.0;
double green = 0.0;
double blue = 0.0;

int filterSize = filter.GetLength(0);
int radius = filterSize / 2;

int w = image.Width;
int h = image.Height;

for (int filterX = -radius; filterX <= radius; filterX++)
{
    for (int filterY = -radius; filterY <= radius; filterY++)
    {
        double filterValue = filter[filterX + radius, filterY + radius];

        int imageX = (x + filterX + w) % w;
        int imageY = (y + filterY + h) % h;

        Color imageColor = image.GetPixel(imageX, imageY);

        red += imageColor.R * filterValue;
        green += imageColor.G * filterValue;
        blue += imageColor.B * filterValue;
    }
}
int r = Math.Min(Math.Max((int)(red), 0), 255);
int g = Math.Min(Math.Max((int)(green), 0), 255);
int b = Math.Min(Math.Max((int)(blue), 0), 255);
return Color.FromArgb(r, g, b);
}

public Bitmap FlipVertical(Bitmap bitmap)
{
    Bitmap flippedBMP = new Bitmap(bitmap.Width, bitmap.Height);
    for (int x = bitmap.Width; x > 0; x--)
    {
        for (int y = 0; y < bitmap.Height; y++)
        {
            Color color = bitmap.GetPixel(x - 1, y);
            flippedBMP.SetPixel(bitmap.Width - x, y, color);
        }
    }
    return flippedBMP;
}

public Bitmap GrayScale(Bitmap bitmap)
{
    Bitmap grayBMP = new Bitmap(bitmap.Width, bitmap.Height);
    for (int x = bitmap.Width; x > 0; x--)
    {
        for (int y = 0; y < bitmap.Height; y++)
        {
            Color color = bitmap.GetPixel(x - 1, y);
            int linear = (int)(0.2126 * color.R + 0.7152 * color.G + 0.0722 * color.B);
            Color newColor = Color.FromArgb(255, linear, linear, linear);
            grayBMP.SetPixel(x - 1, y, newColor);
        }
    }
    return grayBMP;
}

public Bitmap RemoveRed(Bitmap bitmap)
{
    Bitmap noRedBMP = new Bitmap(bitmap.Width, bitmap.Height);
    for (int x = bitmap.Width; x > 0; x--)
    {
        for (int y = 0; y < bitmap.Height; y++)
        {
            Color color = bitmap.GetPixel(x - 1, y);
            Color newColor = Color.FromArgb(255, 0, color.G, color.B);

```

```

        noRedBMP.SetPixel(x - 1, y, newColor);
    }
}
return noRedBMP;
}
}
}
}

```

ProgbaseLab.ImageEditor.Fast.Class1.cs

```

using System;
using System.Drawing;
using System.Drawing.Imaging;

namespace ProgbaseLab.ImageEditor.Fast
{
    public class Class1 : ProgbaseLab.ImageEditor.Common.IRedatctingImage
    {
        public Bitmap Blur(Bitmap image, Int32 blurSize) // код взятий із відкритих джерел, бо не знайшов
        // спосіб через System.Drawing
        {
            return Blur(image, new Rectangle(0, 0, image.Width, image.Height), blurSize);
        }

        private unsafe static Bitmap Blur(Bitmap image, Rectangle rectangle, Int32 blurSize)
        {
            Bitmap blurred = new Bitmap(image.Width, image.Height);

            // make an exact copy of the bitmap provided
            using (Graphics graphics = Graphics.FromImage(blurred))
            {
                graphics.DrawImage(image, new Rectangle(0, 0, image.Width, image.Height),
                    new Rectangle(0, 0, image.Width, image.Height), GraphicsUnit.Pixel);

                // Lock the bitmap's bits
                BitmapData blurredData = blurred.LockBits(new Rectangle(0, 0, image.Width, image.Height),
                    ImageLockMode.ReadWrite, blurred.PixelFormat);

                // Get bits per pixel for current PixelFormat
                int bitsPerPixel = Image.GetPixelFormatSize(blurred.PixelFormat);

                // Get pointer to first line
                byte* scan0 = (byte*)blurredData.Scan0.ToPointer();

                // look at every pixel in the blur rectangle
                for (int xx = rectangle.X; xx < rectangle.X + rectangle.Width; xx++)
                {
                    for (int yy = rectangle.Y; yy < rectangle.Y + rectangle.Height; yy++)
                    {
                        int avgR = 0, avgG = 0, avgB = 0;
                        int blurPixelCount = 0;

                        // average the color of the red, green and blue for each pixel in the
                        // blur size while making sure you don't go outside the image bounds
                        for (int x = xx; (x < xx + blurSize && x < image.Width); x++)
                        {
                            for (int y = yy; (y < yy + blurSize && y < image.Height); y++)
                            {
                                // Get pointer to RGB
                                byte* data = scan0 + y * blurredData.Stride + x * bitsPerPixel / 8;

                                avgB += data[0]; // Blue
                                avgG += data[1]; // Green
                                avgR += data[2]; // Red

                                blurPixelCount++;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        avgR = avgR / blurPixelCount;
        avgG = avgG / blurPixelCount;
        avgB = avgB / blurPixelCount;

        // now that we know the average for the blur size, set each pixel to that color
        for (int x = xx; x < xx + blurSize && x < image.Width && x < rectangle.Width; x++)
        {
            for (int y = yy; y < yy + blurSize && y < image.Height && y < rectangle.Height;
y++)
            {
                // Get pointer to RGB
                byte* data = scan0 + y * blurredData.Stride + x * bitsPerPixel / 8;

                // Change values
                data[0] = (byte)avgB;
                data[1] = (byte)avgG;
                data[2] = (byte)avgR;
            }
        }

        // Unlock the bits
        blurred.UnlockBits(blurredData);

        return blurred;
    }

    public Bitmap Crop(Bitmap bmp, Rectangle rec)
    {
        if (rec.Left < 0 || rec.Left >= bmp.Width)
        {
            throw new Exception("Invalid left");
        }
        if (rec.Right >= bmp.Width)
        {
            throw new Exception("Invalid right");
        }
        if (rec.Top < 0 || rec.Top >= bmp.Height)
        {
            throw new Exception("Invalid right");
        }
        if (rec.Bottom >= bmp.Height)
        {
            throw new Exception("Invalid right");
        }
        bmp = bmp.Clone(rec, System.Drawing.Imaging.PixelFormat.DontCare);
        return bmp;
    }

    public Bitmap FlipVertical(Bitmap bitmap)
    {
        bitmap.RotateFlip(RotateFlipType.Rotate180FlipY);
        return bitmap;
    }

    public Bitmap GrayScale(Bitmap bitmap)
    {
        Bitmap newBMP = new Bitmap(bitmap.Width, bitmap.Height);
        Graphics g = Graphics.FromImage(newBMP);
        ColorMatrix TempMatrix = new ColorMatrix(
            new float[][]
            {
                new float[] { .3f, .3f, .3f, 0, 0 },
                new float[] { .59f, .59f, .59f, 0, 0 },
                new float[] { .11f, .11f, .11f, 0, 0 },
                new float[] { 0, 0, 0, 1, 0 },
                new float[] { 0, 0, 0, 0, 1 }
            });
        ImageAttributes attributes = new ImageAttributes();
        attributes.SetColorMatrix(TempMatrix);
        g.DrawImage(bitmap, new Rectangle(0, 0, bitmap.Width, bitmap.Height), 0, 0, bitmap.Width,

```


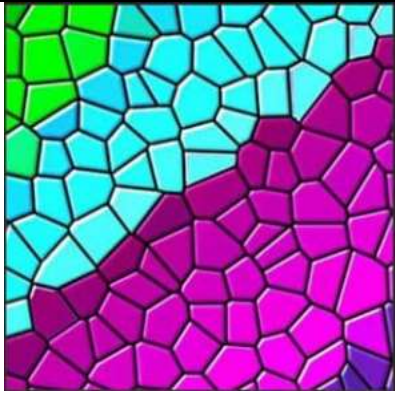
```
bitmap.Height, GraphicsUnit.Pixel, attributes);
    g.Dispose();
    return newBMP;
}

public Bitmap RemoveRed(Bitmap bitmap)    //OpenCvSharp не працює, а інших способів я не знайшов,
тому аналогічна реалізація, як у Pixel
{
    Bitmap noRedBMP = new Bitmap(bitmap.Width, bitmap.Height);
    for (int x = bitmap.Width; x > 0; x--)
    {
        for (int y = 0; y < bitmap.Height; y++)
        {
            Color color = bitmap.GetPixel(x - 1, y);
            Color newColor = Color.FromArgb(255, 0, color.G, color.B);
            noRedBMP.SetPixel(x - 1, y, newColor);
        }
    }
    return noRedBMP;
}
}
```

Приклади результатів

Crop:

Crop 300x300+500+300

Оригінал	Очікуваний результат
	

Fast:

```
dotnet run fast ./lab4.jpg ./out.jpg crop 300x300+500+300
```

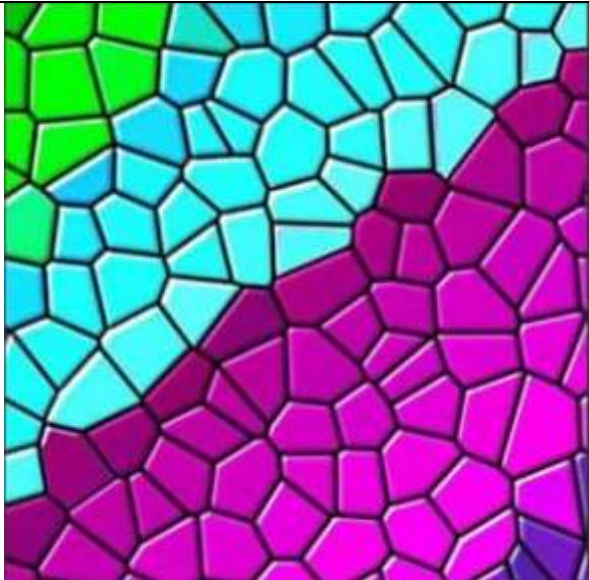
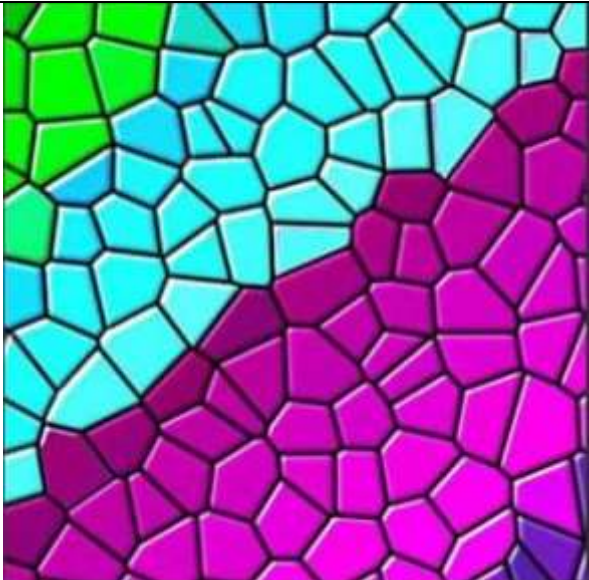
```
Image process finished in 1  
Whole process finished in 49
```

Pixel


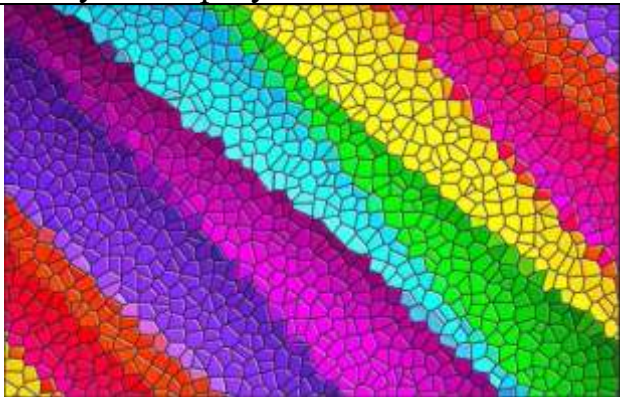
```
dotnet run pixel ./lab4.jpg ./out.jpg crop 300x300+500+300
```

```
Image process finished in 21  
Whole process finished in 83
```

Результат:

Піксельна реалізація	Швидка реалізація
	

FlipVertical:

Оригінал	Очікуваний результат
	

Fast:

```
dotnet run fast ./lab4.jpg ./out.jpg FlipVertical
```



```
Image process finished in 3  
Whole process finished in 38
```

Pixel

```
dotnet run pixel ./lab4.jpg ./out.jpg FlipVertical
```



```
Image process finished in 235  
Whole process finished in 691
```

Результат:

Піксельна реалізація	Швидка реалізація
	

RemoveRed:

Через неможливість виконати дане перетворення існуючими методами, використовуючи System.Drawing, піксельна і швидка реалізація мають однаковий алгоритм.

Оригінал	Очікуваний результат
	

Fast:

```
dotnet run fast ./lab4.jpg ./out.jpg RemoveRed
```


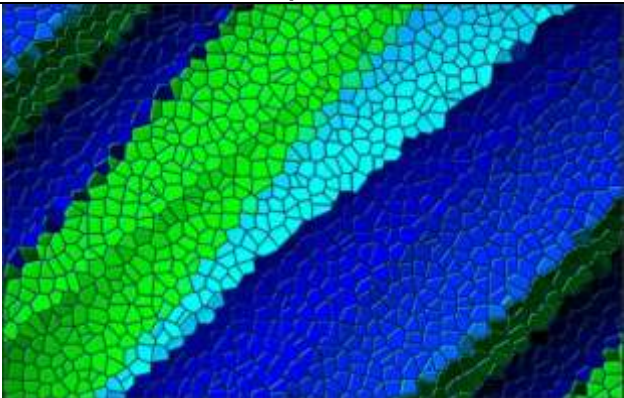
```
Image process finished in 217  
Whole process finished in 819
```

Pixel


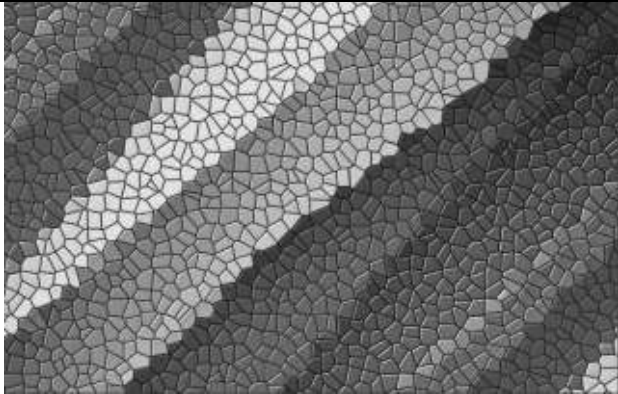
```
dotnet run pixel ./lab4.jpg ./out.jpg RemoveRed
```

```
Image process finished in 232  
Whole process finished in 823
```

Результат:

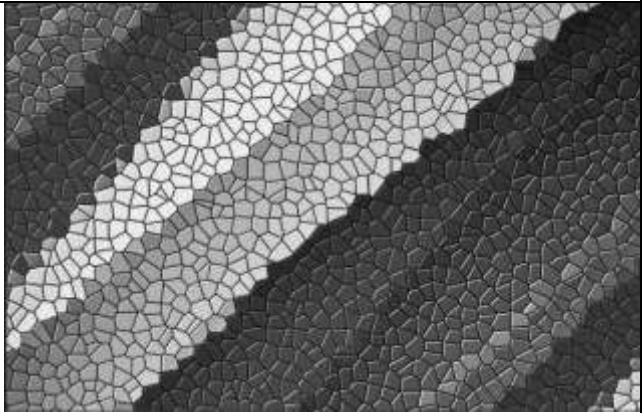
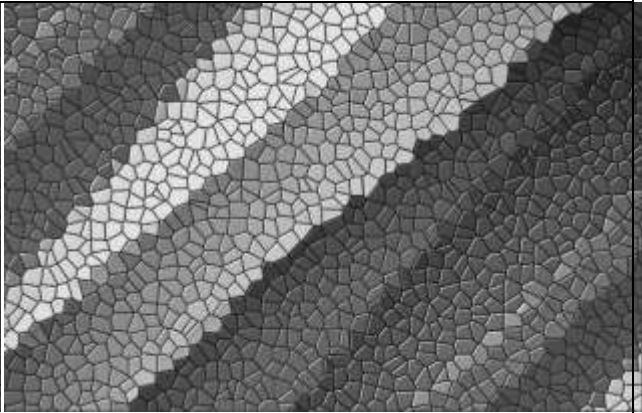
Піксельна реалізація	Швидка реалізація
	

GrayScale:

Оригінал	Очікуваний результат
	


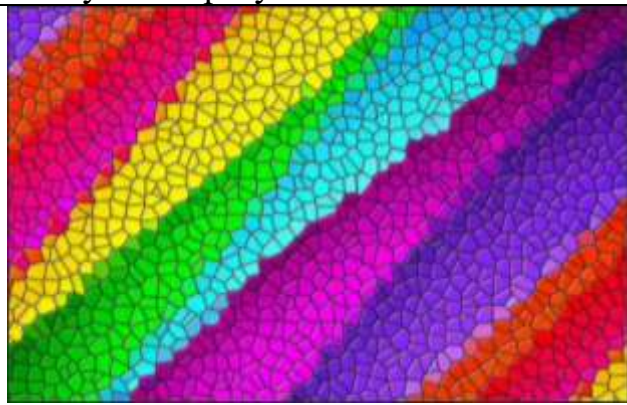
Fast:	
<code>dotnet run fast ./lab4.jpg ./out.jpg GrayScale</code>	
Image process finished in 23	
Whole process finished in 377	
Pixel	
<code>dotnet run pixel ./lab4.jpg ./out.jpg GrayScale</code>	
Image process finished in 237	
Whole process finished in 591	

Результат:

Піксельна реалізація	Швидка реалізація
	

Blur:

Для розмиття було використано алгоритм Гаусового розмиття. Матриця-фільтр формується на основі введеного значення $\{\sigma\}$, тому від його числового значення час виконання алгоритму залежить напрому.

Оригінал	Очікуваний результат
	

Fast:

```
dotnet run fast ./lab4.jpg ./out.jpg Blur 3
```



```
Image process finished in 565  
Whole process finished in 1095
```

Pixel

```
dotnet run pixel ./lab4.jpg ./out.jpg Blur 3
```

```
Image process finished in 12969  
Whole process finished in 13501
```

Результат:

Піксельна реалізація	Швидка реалізація
	

Висновки

Виконавши дану лабораторну роботу було використано розбиття коду на бібліотеки класів. Різні класи і структури можна оформити у вигляді окремих бібліотек, які компілюються в файли dll і потім можуть підключатись в інші проекти. Завдяки цьому можна визначити один і той же функціонал у вигляді бібліотеки класів і підключати в різні проекти.

Також було застосовано System.Drawing, що надає доступ до графічних функцій. Було розібрано кольорові моделі та канали, за допомогою яких можна перетворити зображення у відтінки сірого (Grayscale) або у відтінки світло-коричневого (Sepia). Познайомився з новою кольоровою моделлю HSL (hue, saturation and lightness).

Важливе значення при обробці зображень мають фільтри. Використовуючи їх можна виконувати різноманітні перетворення: зробити "загострити" (Sharpen), змити (Blur) та багато іншого. Brightness, Saturation - приклади модифікацій зображень за допомогою кольорових матриць.

Компіляція всього коду відбувалася за допомогою утиліти dotnet.