**Липецкий государственный технический университет**  
Факультет автоматизации и информатики  
Кафедра автоматизированных систем управления

Лабораторная работа №1

по предмету «Организация графических систем и систем мультимедиа»

Исследование методов обработки изображений

Куранов И.В.

Студент

Группа М-АС-21

Кургасов В.В.

Руководитель

Доцент

Липецк 2022 г.

Задание кафедры

Разработать программу, позволяющую применить не менее 3-х фильтров к загруженному изображению и экспортировать результат в иной графический формат. Необходимо наличие графического меню пользователя. Обработка должна проводиться на CPU.

Содержание

[Теоретические сведения 4](#_Toc97641299)

[Ход работы 4](#_Toc97641300)

[Отображение в оттенках серого 4](#_Toc97641301)

[Размытие 5](#_Toc97641302)

[Оператор Собеля 5](#_Toc97641303)

[Вывод 9](#_Toc97641304)

[Приложение А 10](#_Toc97641305)

# Теоретические сведения

В качестве языка реализации был выбран C# с использованием технологии Windows Forms. Это объектно-ориентированный язык компилируемого типа. К преимуществам можно отнести простоту создания пользовательского интерфейса, а также наличие стандартных инструментов для работы с графикой.

# Ход работы

Приложение позволяет пользователю наложить на загруженное изображение один из трех фильтров: отображение в оттенках серого, размытие, а также получение границ при помощи оператора Собеля.

Исходный код программы представлен в приложении А.

## Отображение в оттенках серого

Преобразование цветного изображения в модели RGB к серому заключается в замене разных значений красного, зеленого, и синего цвета для данной точки на единое значение, характеризующее яркость точки. На цвет в стандартной RGB модели отводится 1 байт, соответственно, при таком преобразовании точки могут принимать значения из диапазона [0,0,0], [1,1,1], [2,2,2]...[255,255,255]. Где [0,0,0] - черный цвет, а [255,255,255] - белый цвет. Цветное изображение таким образом представляется 256 оттенками серого.

Так как цвета с разной длиной волны по-разному воспринимаются человеческим глазом, то для вычисления яркости точки разным цветам назначаются разные веса. Наиболее чувствителен человек к зеленому цвету, поэтому для него используется самый большой вес - 0.7152, наименее чувствителен к синему, для которого используется вес 0.0722, и для красного используется вес 0.2126. Эти веса прописаны в стандарте sRGB – именно так вычисляется яркость точки при переходе от sRGB в цветовое пространство CIE XYZ. На рисунке 1 представлен результат наложения фильтра.

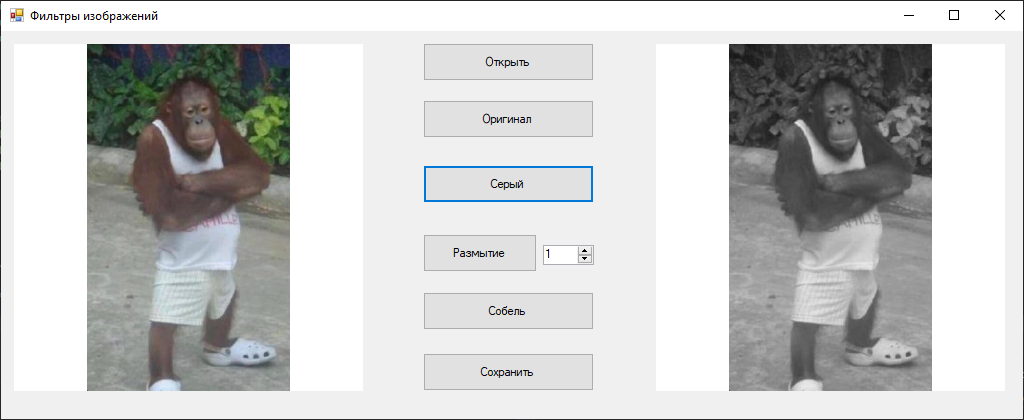


Рисунок 1 – Отображение в оттенках серого

## Размытие

Одним из способов реализации размытия является изменения значений каналов цвета пикселя на среднее значение окружающих его пикселей. На рисунке 2 представлен результат размытия изображения.

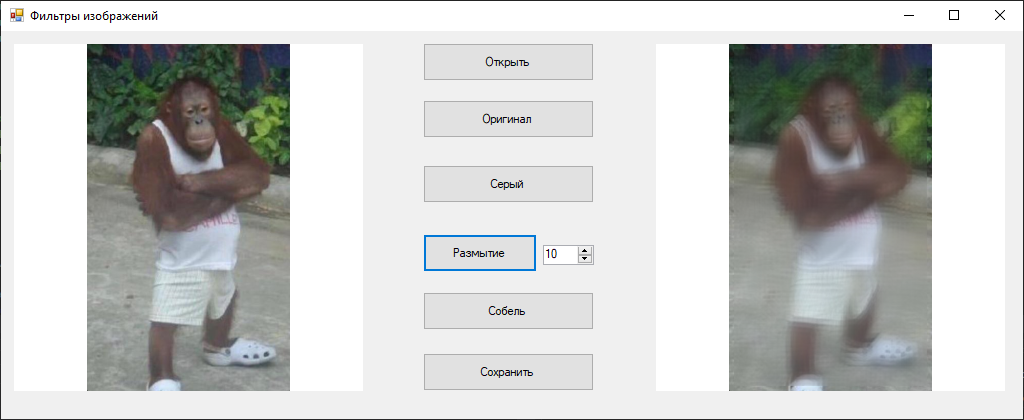


Рисунок 2 – Размытие

## Оператор Собеля

Оператор Собеля – дискретный дифференциальный оператор, вычисляющий приближённое значение градиента яркости изображения. Результатом применения оператора Собеля в каждой точке изображения является либо вектор градиента яркости в этой точке, либо его норма. Оператор вычисляет градиент яркости изображения в каждой точке. Так находится направление наибольшего увеличения яркости и величина её изменения в этом направлении. Результат показывает, насколько «резко» или «плавно» меняется яркость изображения в каждой точке, а значит, вероятность нахождения точки на грани, а также ориентацию границы. На рисунке 3 показаны найденные границы.

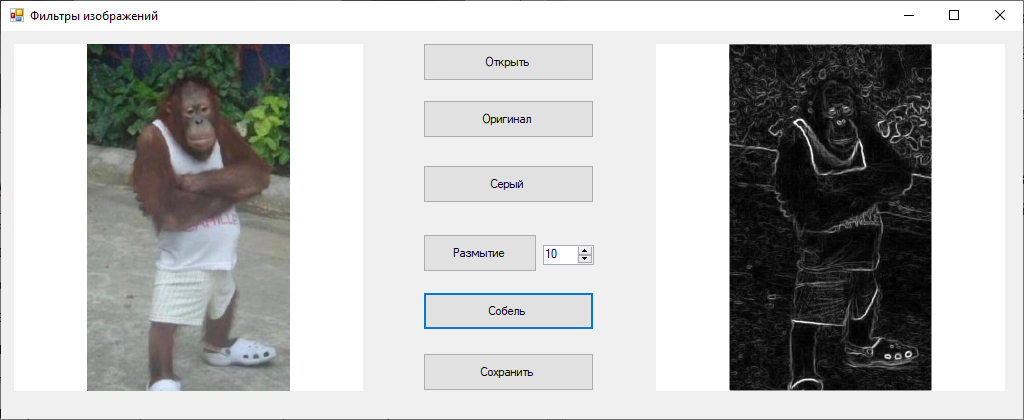


Рисунок 3 – Оператор Собеля

Для сохранения полученного изображения в выбранный формат можно воспользоваться функцией Save класса Image, передав туда нужный формат при помощи класса ImageFormat (например ImageFormat.Jpeg).

Из-за того, что разработанное приложение использует нативные средства для построения графического пользовательского интерфейса (WPF/Windows Forms), то данная программа не может быть упакована в контейнер. Создадим инсталлятор для разработанного приложения. Он позволят устанавливать вместе с приложением все необходимые компоненты зависимостей.

На рисунке 4 показана структура проекта после добавления инсталлятора.

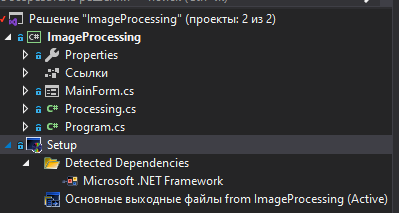


Рисунок 4 – Структура проекта

На рисунке 5 показаны компоненты, которые будут устанавливаться вместе с программой.

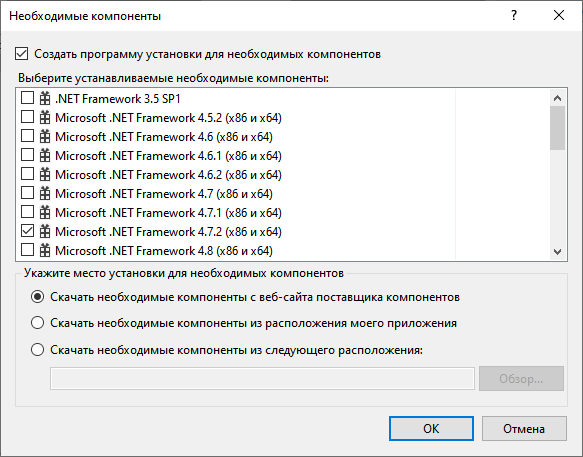


Рисунок 5 – Устанавливаемые компоненты

После сборки проекта появятся 2 файла с расширениями .exe и .msi. На рисунке 6 показано окно установки.

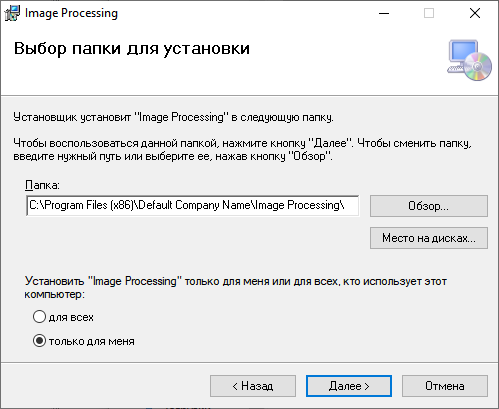


Рисунок 6 – Окно установки

# Вывод

В ходе выполнения лабораторной работы была разработана программа, позволяющая пользователю наложить один из трех фильтров (оттенки серого, размытие и границы объектов) на загруженное изображение, а результат перевести в выбранный формат. Приложение реализовано с использованием стандартных средств языка C# и технологи Windows Forms. Инсталлятор был создан при помощи расширения Microsoft Visual Studio Installer Projects.

# Приложение А

(обязательное)

В данном приложении представлен листинг класса Processing.

using System;

using System.Drawing;

using System.Drawing.Imaging;

using System.Runtime.InteropServices;

namespace ImageProcessing

{

class Processing

{

public Processing()

{

}

public static bool ConvertToGray(Bitmap bitmap)

{

for (int i = 0; i < bitmap.Width; i++)

{

for (int j = 0; j < bitmap.Height; j++)

{

Color c1 = bitmap.GetPixel(i, j);

int r1 = c1.R;

int g1 = c1.G;

int b1 = c1.B;

// Формула для расчета нового цвета

int gray = (byte)(.2126 \* r1 + .7152 \* g1 + .0722 \* b1);

r1 = gray;

g1 = gray;

b1 = gray;

bitmap.SetPixel(i, j, Color.FromArgb(r1, g1, b1));

}

}

return true;

}

public static bool Blur(Bitmap bitmap, int winSize)

{

for (int i = 0; i < bitmap.Width; i++)

{

for (int j = 0; j < bitmap.Height; j++)

{

try

{

int count = 0;

int avgR = 0, avgG = 0, avgB = 0;

// Суммирование значений по каналам для вычисления среднего с проверкой на выход за границы изображения

if (i - winSize >= 0)

{

Color prevX = bitmap.GetPixel(i - winSize, j);

avgR += prevX.R;

avgG += prevX.G;

avgB += prevX.B;

count++;

}

if (i + winSize < bitmap.Width)

{

Color nextX = bitmap.GetPixel(i + winSize, j);

avgR += nextX.R;

avgG += nextX.G;

avgB += nextX.B;

count++;

}

if (j - winSize >= 0)

{

Color prevY = bitmap.GetPixel(i, j - winSize);

avgR += prevY.R;

avgG += prevY.G;

avgB += prevY.B;

count++;

}

if (j + winSize < bitmap.Height)

{

Color nextY = bitmap.GetPixel(i, j + winSize);

avgR += nextY.R;

avgG += nextY.G;

avgB += nextY.B;

count++;

}

avgR = (avgR / count);

avgG = (avgG / count);

avgB = (avgB / count);

bitmap.SetPixel(i, j, Color.FromArgb(avgR, avgG, avgB));

}

catch

{

}

}

}

return true;

}

public static bool Sobel(Bitmap Image, Bitmap Image2)

{

BitmapData ImageData, ImageData2;

byte[] buffer, buffer2;

IntPtr pointer, pointer2;

int b, g, r, r\_x, g\_x, b\_x, r\_y, g\_y, b\_y, grayscale, location, location2;

sbyte weight\_x, weight\_y;

sbyte[,] weights\_x;

sbyte[,] weights\_y;

// Матрицы Собеля

weights\_x = new sbyte[,] { { 1, 0, -1 }, { 2, 0, -2 }, { 1, 0, -1 } };

weights\_y = new sbyte[,] { { 1, 2, 1 }, { 0, 0, 0 }, { -1, -2, -1 } };

// Блокируем объекты Bitmap в системной области

ImageData = Image.LockBits(new Rectangle(0, 0, Image.Width, Image.Height), ImageLockMode.ReadOnly, PixelFormat.Format24bppRgb);

ImageData2 = Image2.LockBits(new Rectangle(0, 0, Image.Width, Image.Height), ImageLockMode.WriteOnly, PixelFormat.Format24bppRgb);

buffer = new byte[ImageData.Stride \* Image.Height];

buffer2 = new byte[ImageData.Stride \* Image.Height];

pointer = ImageData.Scan0;

pointer2 = ImageData2.Scan0;

Marshal.Copy(pointer, buffer, 0, buffer.Length);

for (int y = 0; y < Image.Height; y++)

{

for (int x = 0; x < Image.Width \* 3; x += 3)

{

r\_x = g\_x = b\_x = 0; // Обнуление градиентов в x-направленных значениях

r\_y = g\_y = b\_y = 0; // Обнуление градиентов в y-направленных значениях

location = x + y \* ImageData.Stride; // Получение местоположения каждого пикселя

for (int yy = -(int)Math.Floor(weights\_y.GetLength(0) / 2.0d), yyy = 0; yy <= (int)Math.Floor(weights\_y.GetLength(0) / 2.0d); yy++, yyy++)

{

if (y + yy >= 0 && y + yy < Image.Height) // Чтобы не выйти за границы массива

{

for (int xx = -(int)Math.Floor(weights\_x.GetLength(1) / 2.0d) \* 3, xxx = 0; xx <= (int)Math.Floor(weights\_x.GetLength(1) / 2.0d) \* 3; xx += 3, xxx++)

{

if (x + xx >= 0 && x + xx <= Image.Width \* 3 - 3) // Чтобы не выйти за границы массива

{

location2 = x + xx + (yy + y) \* ImageData.Stride;

weight\_x = weights\_x[yyy, xxx];

weight\_y = weights\_y[yyy, xxx];

// Прмиенение веса ко всем каналам

b\_x += buffer[location2] \* weight\_x;

g\_x += buffer[location2 + 1] \* weight\_x;

r\_x += buffer[location2 + 2] \* weight\_x;

b\_y += buffer[location2] \* weight\_y;

g\_y += buffer[location2 + 1] \* weight\_y;

r\_y += buffer[location2 + 2] \* weight\_y;

}

}

}

}

// Получение значения каждого канала

b = (int)Math.Sqrt(Math.Pow(b\_x, 2) + Math.Pow(b\_y, 2));

g = (int)Math.Sqrt(Math.Pow(g\_x, 2) + Math.Pow(g\_y, 2));

r = (int)Math.Sqrt(Math.Pow(r\_x, 2) + Math.Pow(r\_y, 2));

if (b > 255) b = 255;

if (g > 255) g = 255;

if (r > 255) r = 255;

grayscale = (b + g + r) / 3;

buffer2[location] = (byte)grayscale;

buffer2[location + 1] = (byte)grayscale;

buffer2[location + 2] = (byte)grayscale;

}

}

Marshal.Copy(buffer2, 0, pointer2, buffer.Length);

Image.UnlockBits(ImageData);

Image2.UnlockBits(ImageData2);

return true;

}

}

}