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

Лабораторная работа №1

по дисциплине «Организация графических систем и систем мультимедиа»

Исследование методов обработки изображений

Пархомов А.С.

Студент

Группа М-АС-21

Кургасов В.В.

Руководитель

Доцент

Липецк 2022 г.

Цель работы

Изучить различные методы обработки изображений с использованием CPU.

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

[1 Теоретические сведения 5](#_Toc103533492)

[2 Ход работы 5](#_Toc103533493)

[2.1 Чёрно-белый фильтр 5](#_Toc103533494)

[2.2 Негатив 7](#_Toc103533495)

[2.3 Шум 8](#_Toc103533496)

[2.4 Сепия 9](#_Toc103533497)

[2.5 Преобразование форматов изображения 10](#_Toc103533498)

[2.6 Создание установщика 11](#_Toc103533499)

[Вывод 13](#_Toc103533500)

[Приложение А 14](#_Toc103533501)

1. Теоретические сведения

В качестве языка программирования для реализации проекта была выбрана платформа .NET Framework и язык программирования C#. Графический интерфейс пользователя реализован с помощью фреймворка WPF. Данные инструменты были выбраны для достижения высокой производительности с использованием современных технологий. При разработке приложения были использованы только компоненты платформы без дополнительных библиотек и зависимостей. Разработанный установщик позволяет использовать приложения на операционных системах семейства Windows начиная с версии Windows XP.

Исходный код основной части программы представлен в приложении А.

1. Ход работы

Разработанное приложение позволяет пользователю применить к загруженному изображению четыре фильтра: чёрно-белый фильтр, негатив, шум и сепию. Кроме того, приложение позволяет сохранить обработанное изображение в разных форматах: png, jpeg и bmp.

* 1. Чёрно-белый фильтр

Преобразование цветного изображения (рисунок 1) в модели ARGB (3 байта цвета и 1 байт прозрачности) заключается в нахождении среднего значения цветовых байтов с его последующим присвоением компонентам RGB цветовой модели (рисунок 2).



Рисунок 1 – Исходное изображение



Рисунок 2 – Изображение после применения чёрно-белого фильтра

* 1. Негатив

Данный фильтр инвертирует цвета исходного изображения (рисунок 3) в формате ARGB (3 байта цвета и 1 байт прозрачности) по следующему правилу (рисунок 4):

где , , , – компоненты цветовой модели.



Рисунок 3 – Исходное изображение



Рисунок 4 – Изображение после применения фильтра негатива

* 1. Шум

Применение фильтра шума основано на генераторе случайных чисел, который с заданной вероятностью искажает пиксели исходного изображения (рисунок 5) для получения требуемого эффекта (рисунок 6). Искажение заключается в присвоении значениям пикселей цветовой модели ARGB значений цветовых компонент серого цвета:

.



Рисунок 5 – Исходное изображение



Рисунок 6 – Изображение после применения фильтра шума

* 1. Сепия

Фильтр сепия позволяет изменить цветовую палитру исходного изображения (рисунок 7) с целью получения эффекта съёмки на старый фотоаппарат (рисунок 8). Фильтр работает с цветовой моделью ARGB по следующим правилам:



Рисунок 7 – Исходное изображение

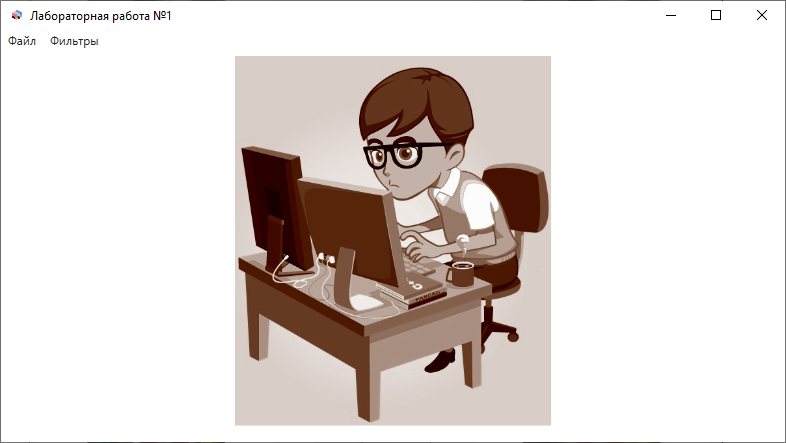


Рисунок 8 – Изображение после применения фильтра сепия

* 1. Преобразование форматов изображения

Разработанное приложение поддерживает открытие и сохранение файлов в следующих форматах: png, jpg и bmp. Для работы с изображением используется класс Bitmap и его методы. Открытие и сохранение изображение продемонстрированы на рисунках 9 и 10.

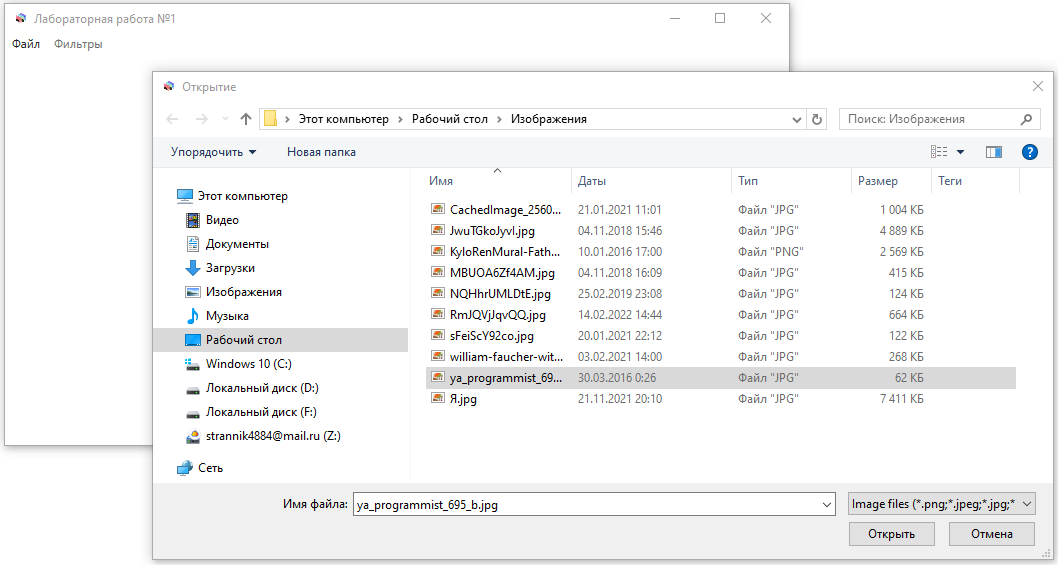


Рисунок 9 – Открытие изображения

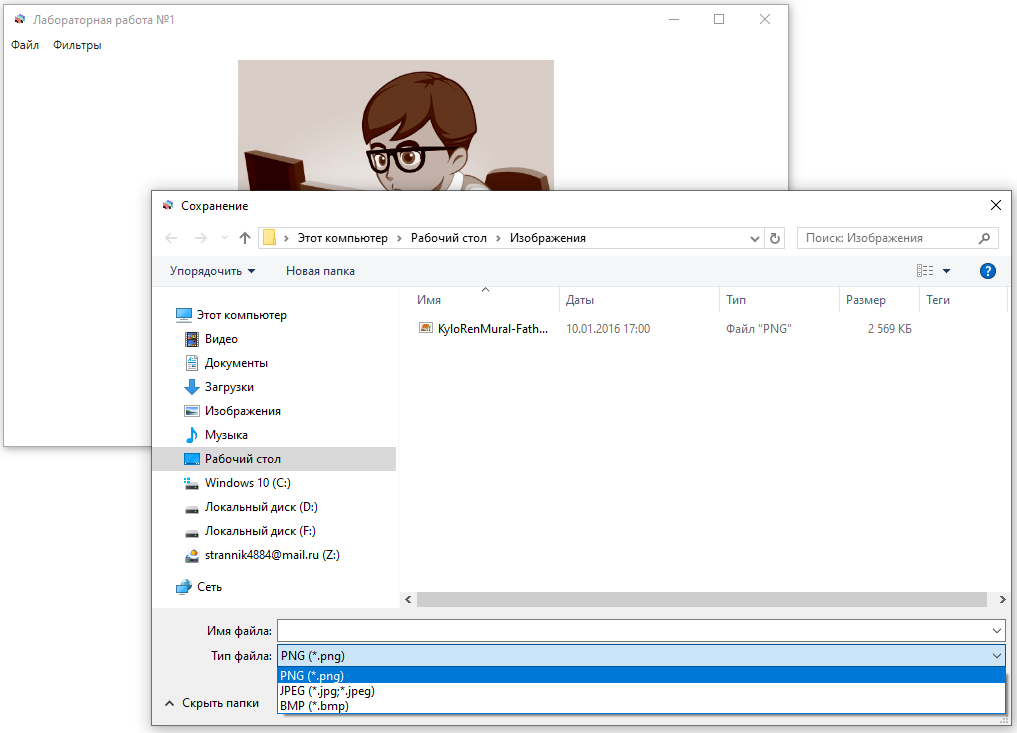


Рисунок 10 – Сохранение изображения

* 1. Создание установщика

Для удобного распространения разработанного приложения был создан установщик с помощью средств Visual Studio (рисунок 11). Данный установщик позволяет распространять как саму программу (рисунок 12), так и платформу .NET Framework требуемой версии (рисунок 13).

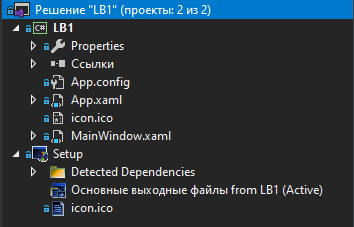


Рисунок 11 – Структура проекта с установщиком

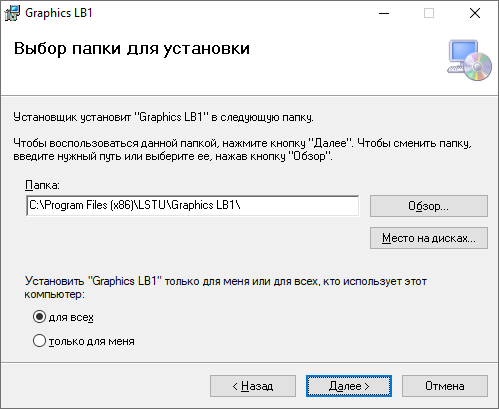


Рисунок 12 – Установка приложения

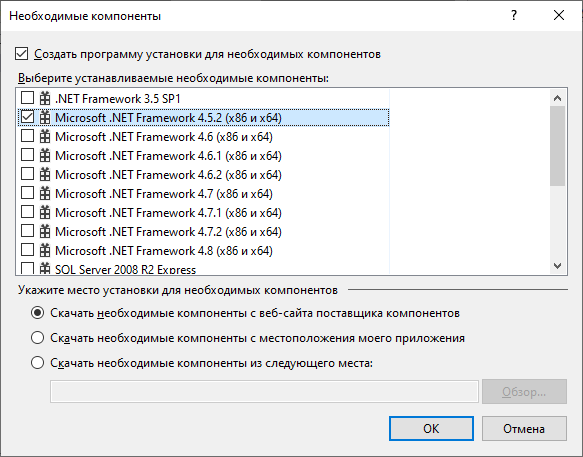


Рисунок 13 – Платформа .NET Framework, распространяемая вместе с установщиком

Вывод

В ходе выполнения данной лабораторной работы были изучены различные методы обработки изображений с использованием CPU и разработано приложение, которое позволяет пользователю применить к загруженному изображению четыре фильтра: чёрно-белый фильтр, негатив, шум и сепию. Кроме того, приложение позволяет сохранить обработанное изображение в разных форматах: png, jpeg и bmp.

Также был создан установщик для удобного распространения приложения вместе с платформой .NET Framework.

Приложение А

Исходный код основного класса приложения

Исходный код класса MainWindow.xaml.cs:

using System;

using System.Windows;

using System.Drawing;

using System.Drawing.Imaging;

using System.Windows.Media.Imaging;

using System.IO;

using Microsoft.Win32;

namespace LB1

{

public partial class MainWindow : Window

{

private bool IsSavedImage = true;

private delegate byte[] Filter(byte[] rgbValues);

public MainWindow()

{

InitializeComponent();

}

// обработчик клика на пункт меню нового файла

private void NewFile\_Click(object sender, RoutedEventArgs e)

{

if (!IsSavedImage)

{

if (MessageBox.Show("Текущее изображение не сохранено! Удалить его и продолжить?", "Несохранённые данные", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

ImageBox.Source = null;

UpdateUI();

}

}

else

{

ImageBox.Source = null;

UpdateUI();

}

}

// обработчик клика на пункт меню открытия файла

private void OpenFile\_Click(object sender, RoutedEventArgs e)

{

try

{

if (!IsSavedImage)

{

if (MessageBox.Show("Текущее изображение не сохранено! Удалить его и продолжить?", "Несохранённые данные", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

OpenFileDialog openFileDialog = new OpenFileDialog

{

Filter = "Image files (\*.png;\*.jpeg;\*.jpg;\*.bmp)|\*.png;\*.jpeg;\*.jpg;\*.bmp"

};

if (openFileDialog.ShowDialog() == true)

{

ImageBox.Source = new BitmapImage(new Uri(openFileDialog.FileName));

UpdateUI();

}

}

}

else

{

OpenFileDialog openFileDialog = new OpenFileDialog

{

Filter = "Image files (\*.png;\*.jpeg;\*.jpg;\*.bmp)|\*.png;\*.jpeg;\*.jpg;\*.bmp"

};

if (openFileDialog.ShowDialog() == true)

{

ImageBox.Source = new BitmapImage(new Uri(openFileDialog.FileName));

IsSavedImage = true;

UpdateUI();

}

}

}

catch (Exception)

{

MessageBox.Show("Ошибка при открытии изображения: неверный формат файла", "Ошибка файла", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

// обработчик клика на пункт меню сохранения файла

private void SaveFileAs\_Click(object sender, RoutedEventArgs e)

{

try

{

SaveFileDialog saveFileDialog = new SaveFileDialog

{

Filter = "PNG (\*.png)|\*.png|JPEG (\*.jpg;\*.jpeg)|\*.jpg;\*.jpeg|BMP (\*.bmp)|\*.bmp"

};

if (saveFileDialog.ShowDialog() == true)

{

// получаем Bitmap текущего изображения

Bitmap bmp = GetBitmap((BitmapSource)ImageBox.Source);

bmp.Save(saveFileDialog.FileName, GetImageFormatByFilterIndex(saveFileDialog.FilterIndex));

IsSavedImage = true;

}

}

catch(Exception ex)

{

MessageBox.Show("Ошибка при сохранении изображения: " + ex.Message, "Ошибка файла", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

// обработчик клика на пункт меню закрытия программы

private void ExitMenu\_Click(object sender, RoutedEventArgs e)

{

Close();

}

// обработчик события закрытия приложения

private void Window\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

if (!IsSavedImage)

{

if (MessageBox.Show("Текущее изображение не сохранено! Удалить его и продолжить?", "Несохранённые данные", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.No)

{

e.Cancel = true;

}

}

}

// обработчик перетаскивания файла изображения на окно приложения

private void ImageBox\_Drop(object sender, DragEventArgs e)

{

try

{

if (!IsSavedImage)

{

if (MessageBox.Show("Текущее изображение не сохранено! Удалить его и продолжить?", "Несохранённые данные", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

if (e.Data.GetDataPresent(DataFormats.FileDrop))

{

string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);

// проверяем расширение файла

if (!IsCorrectImageExtension(files[0]))

{

throw new InvalidDataException("Неверный формат файла");

}

ImageBox.Source = new BitmapImage(new Uri(files[0]));

UpdateUI();

}

}

}

else

{

if (e.Data.GetDataPresent(DataFormats.FileDrop))

{

string[] files = (string[])e.Data.GetData(DataFormats.FileDrop);

// проверяем расширение файла

if (!IsCorrectImageExtension(files[0]))

{

throw new InvalidDataException("Неверный формат файла");

}

ImageBox.Source = new BitmapImage(new Uri(files[0]));

UpdateUI();

}

}

}

catch(Exception)

{

MessageBox.Show("Ошибка при открытии изображения: неверный формат файла", "Ошибка файла", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

// функция для обновления элементов пользовательского интерфейса

private void UpdateUI()

{

IsSavedImage = true;

if (ImageBox.Source != null)

{

SaveFileAs.IsEnabled = true;

Filters.IsEnabled = true;

EmptyImageBox.Visibility = Visibility.Collapsed;

}

else

{

SaveFileAs.IsEnabled = false;

Filters.IsEnabled = false;

EmptyImageBox.Visibility = Visibility.Visible;

}

}

// функция для работы с Bitmap'ом напрямую

private void MakeFilter(Filter filter)

{

try

{

// получаем Bitmap текущего изображения

Bitmap bmp = GetBitmap((BitmapSource)ImageBox.Source);

// блокируем данные Bitmap'а

Rectangle rect = new Rectangle(0, 0, bmp.Width, bmp.Height);

BitmapData bmpData = bmp.LockBits(rect, ImageLockMode.ReadWrite, bmp.PixelFormat);

// получаем указатель на начало Bitmap'а

IntPtr ptr = bmpData.Scan0;

// объявляем массив для хранения пикселей

int bytes = Math.Abs(bmpData.Stride) \* bmp.Height;

byte[] rgbValues = new byte[bytes];

// копируем пиксели в массив

System.Runtime.InteropServices.Marshal.Copy(ptr, rgbValues, 0, bytes);

// накладываем фильтр

rgbValues = filter(rgbValues);

// копируем значения пикселей обратно в Bitmap

System.Runtime.InteropServices.Marshal.Copy(rgbValues, 0, ptr, bytes);

// разблокируем данные Bitmap'а

bmp.UnlockBits(bmpData);

// обновляем изображение на форме

ImageBox.Source = Convert(bmp);

IsSavedImage = false;

}

catch(Exception ex)

{

MessageBox.Show("Ошибка при наложении фильтра: " + ex.Message, "Ошибка фильтра", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

// функция наложения чёрно-белого фильтра

private byte[] MakeBlackWhiteFilter(byte[] rgbValues)

{

// проходимся по всем пикселям

for (int i = 0; i < rgbValues.Length; i += 4)

{

// получаем среднее значение для цветов RGB

int averageValue = (rgbValues[i] + rgbValues[i + 1] + rgbValues[i + 2]) / 3;

// устанавливаем среднее значение каждому цвету RGB

rgbValues[i] = rgbValues[i + 1] = rgbValues[i + 2] = (byte)averageValue;

}

return rgbValues;

}

// функция наложения негатива

private byte[] MakeNegativeFilter(byte[] rgbValues)

{

// проходимся по всем пикселям

for (int i = 0; i < rgbValues.Length; i += 4)

{

// получаем текущие значения ARGB

int R = rgbValues[i];

int G = rgbValues[i + 1];

int B = rgbValues[i + 2];

int A = rgbValues[i + 3];

// меняем местами значения ARGB

rgbValues[i] = (byte)(255 - R);

rgbValues[i + 1] = (byte)(255 - G);

rgbValues[i + 2] = (byte)(255 - B);

rgbValues[i + 3] = (byte)(255 - A);

}

return rgbValues;

}

// функция наложения шума

private byte[] MakeNoiseFilter(byte[] rgbValues)

{

Random random = new Random();

// проходимся по всем пикселям

for (int i = 0; i < rgbValues.Length; i += 4)

{

// с вероятностью 60% устанавливаем серый цвет текущему пикселю

int q = random.Next(100);

if (q <= 40)

{

rgbValues[i] = Color.Gray.R;

rgbValues[i + 1] = Color.Gray.G;

rgbValues[i + 2] = Color.Gray.B;

}

}

return rgbValues;

}

// функция наложения сепии

private byte[] MakeSepiaFilter(byte[] rgbValues)

{

// получаем негатив изображения

rgbValues = MakeNegativeFilter(rgbValues);

// проходимся по всем пикселям

for (int i = 0; i < rgbValues.Length; i += 4)

{

// получаем текущее значение RGB

int R = rgbValues[i];

int G = rgbValues[i + 1];

int B = rgbValues[i + 2];

// определяем новые значения для RGB сепия

int tr = (int)(0.393 \* R + 0.769 \* G + 0.189 \* B);

int tg = (int)(0.349 \* R + 0.686 \* G + 0.168 \* B);

int tb = (int)(0.272 \* R + 0.534 \* G + 0.131 \* B);

// устанавливаем новые значения RGB учитывая переполнение

rgbValues[i] = (byte)(tr > 255 ? 255 : tr);

rgbValues[i + 1] = (byte)(tg > 255 ? 255 : tg);

rgbValues[i + 2] = (byte)(tb > 255 ? 255 : tb);

}

// убираем негатив и возвращаем полученный массив

return MakeNegativeFilter(rgbValues);

}

// обработчик клика на пункт меню чёрно-белого шума

private void BlackWhiteFilter\_Click(object sender, RoutedEventArgs e)

{

MakeFilter(MakeBlackWhiteFilter);

}

// обработчик клика на пункт меню негатива

private void NegativeFilter\_Click(object sender, RoutedEventArgs e)

{

MakeFilter(MakeNegativeFilter);

}

// обработчик клика на пункт меню шума

private void NoiseFilter\_Click(object sender, RoutedEventArgs e)

{

MakeFilter(MakeNoiseFilter);

}

// обработчик клика на пункт меню сепии

private void SepiaFilter\_Click(object sender, RoutedEventArgs e)

{

MakeFilter(MakeSepiaFilter);

}

// функция для перевода BitmapSource в Bitmap

private Bitmap GetBitmap(BitmapSource source)

{

Bitmap bmp = new Bitmap(source.PixelWidth, source.PixelHeight, PixelFormat.Format32bppPArgb);

BitmapData data = bmp.LockBits(new Rectangle(System.Drawing.Point.Empty, bmp.Size), ImageLockMode.WriteOnly, PixelFormat.Format32bppPArgb);

source.CopyPixels(Int32Rect.Empty, data.Scan0, data.Height \* data.Stride, data.Stride);

bmp.UnlockBits(data);

return bmp;

}

// функция для перевода Bitmap в BitmapSource

private BitmapImage Convert(Bitmap src)

{

MemoryStream ms = new MemoryStream();

src.Save(ms, ImageFormat.Bmp);

BitmapImage image = new BitmapImage();

image.BeginInit();

ms.Seek(0, SeekOrigin.Begin);

image.StreamSource = ms;

image.EndInit();

return image;

}

// функция для проверки расширения файла изображения

private bool IsCorrectImageExtension(string path)

{

string extension = Path.GetExtension(path).ToLower();

if (extension == ".png" || extension == ".jpg" || extension == ".jpeg" || extension == ".bmp")

{

return true;

}

return false;

}

// функция для получения формата изображения по выбранному индексу фильтра

private ImageFormat GetImageFormatByFilterIndex(int index)

{

switch (index)

{

case 0:

return ImageFormat.Png;

case 1:

return ImageFormat.Jpeg;

case 2:

return ImageFormat.Bmp;

default:

return ImageFormat.Png;

}

}

}

}