ОГЛАВЛЕНИЕ

Для лабораторных работ рекомендуется использовать VisualStudio 2010 или более поздние версии.

Список сокращений:

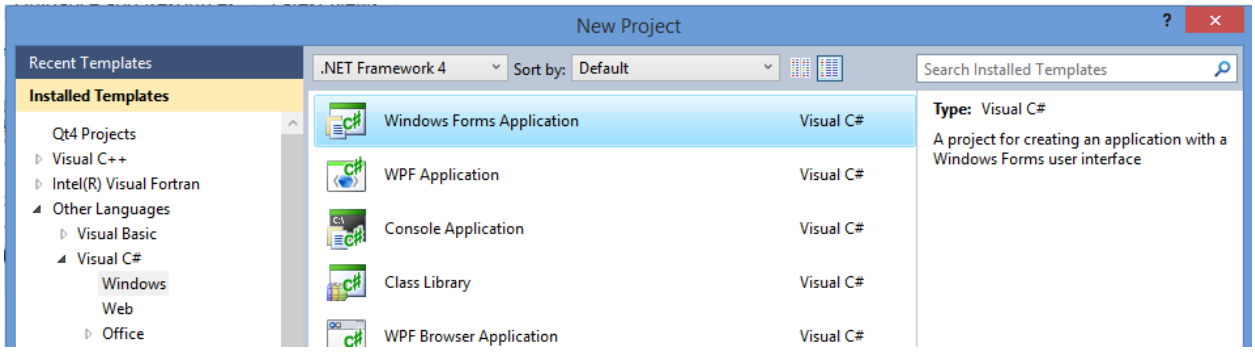
ЛКМ – левая кнопка мыши.

ПКМ – правая кнопка мыши.

ЛАБОРАТОРНАЯ РАБОТА №1. ФИЛЬТРЫ

1. СОЗДАНИЕ ПРОЕКТА

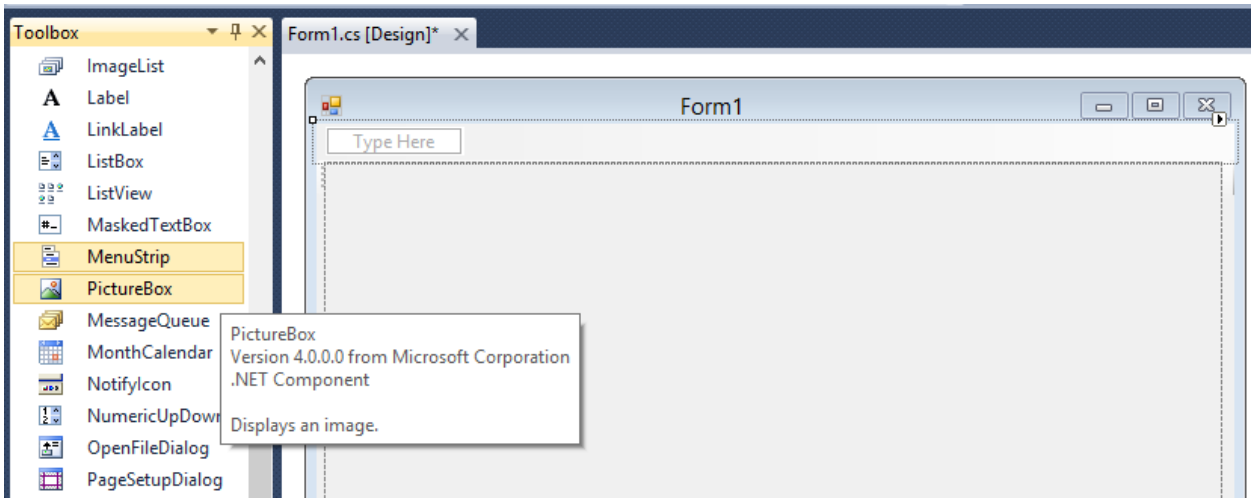
Чтобы создать новый проект, выберитеFile ->New ->Project, или нажмите Ctrl+Shift+N.В открывшемся окне выберите шаблон WindowsFormsApplication. В нижней части окна введите имя для вашего будущего проекта.



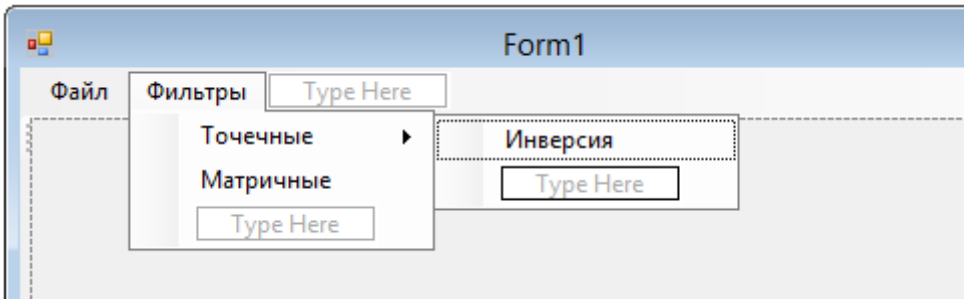
Нажмите F5. На экране должно открыться пустое окно вашей программы.

2. ДОБАВЛЕНИЕ ГРАФИЧЕСКИХ ЭЛЕМЕНТОВ НА ФОРМУ

Чтобы добавить графические элементы на форму, откройте SolutionExplorer (Ctrl+Alt+L). В открывшемся окне выберите файл, содержащий код вашей формы (по умолчанию Form1.cs), по щелчку ПКМ выберите пункт ViewDesigner (или нажмите Shift+F7), откроется окно с формой. Потяните за правый нижний маркер, чтобы увеличить размеры формы. Нажмите Ctrl+Alt+X, чтобы открыть панель Toolbox. С Toolbox на форму перетащите элементы PictureBox и MenuStrip, они появятся на форме. MenuStrip автоматически займет место под заголовком формы, а PictureBox появится на том месте, куда вы его перетащили, растяните его до размеров самой формы. Окно пример следующий вид:



Щелкните ЛКМ по панели MenuStrip, в появившемся текстовом поле введите строку «Файл». После этого появится возможность создать вложенное текстовое поле, впишите строку «Открыть». По аналогии сделайте главный пункт меню «Фильтры», а вложенными элементами «Точечные» и «Матричные». В точечные фильтры аналогично добавьте пункт «Инверсия». В результате получится такая иерархия.



3. ЗАГРУЗКА ИЗОБРАЖЕНИЯ В ПРОГРАММУ

Откройте исходный код формы (На форме ПКМ ->ViewCode или Ctrl+Alt+0). Найдите место, где начинается код нашей формы Form1, и создайте объект Bitmap.

```
public partial class Form1 : Form
{
    Bitmap image;
```

Возвращаемся к графическому представлению формы, и делаем двойной щелчок по элементу меню «Открыть», у нас автоматически создастся функция *открытьToolStripMenuItem_Click*, в которую мы будем добавлять код.

```
private void открытьToolStripMenuItem_Click(object sender, EventArgs e)
{
}
```

Создайте объект типа OpenFileDialog и инициализируйте его конструктором по умолчанию (конструктором без параметров)

```
private void открытьToolStripMenuItem_Click(object sender, EventArgs e)
{
    // создаем диалог для открытия файла
    OpenFileDialog dialog = new OpenFileDialog();
```

Для удобства открытия только изображений, чтобы в окне проводника вы было видно других файлов, добавьте фильтр:

```
OpenFileDialog dialog = new OpenFileDialog();
dialog.Filter = "Image files | *.png; *.jpg; *.bmp | All Files (*.*) | *.*";
```

Проверить, выбрал ли пользователь файл, можно с помощью следующего условия:

```
if (dialog.ShowDialog() == DialogResult.OK)
{
}
```

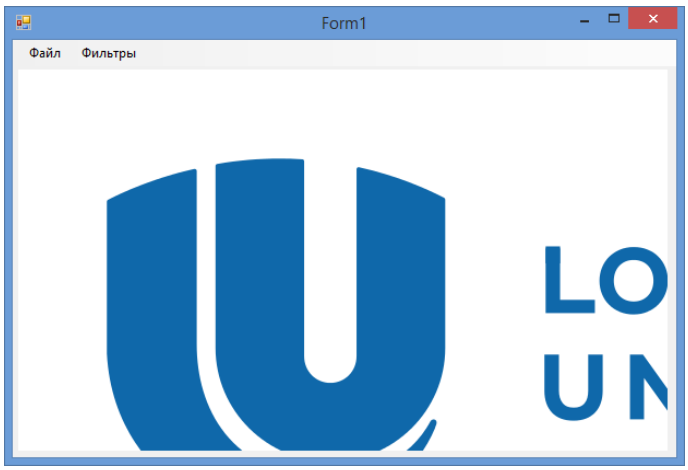
В случае выполнения данного условия инициализируйте вашу переменную image выбранным изображением. Для этого воспользуйтесь конструкцией ниже.

```
image = new Bitmap(dialog.FileName);
```

После того, как вы загрузили картинку в программу, необходимо ее визуализировать на форме, для этого image присвойте свойству pictureBox.Image и обновите ваш pictureBox.

```
pictureBox1.Image = image;
pictureBox1.Refresh();
```

Проверьте, что ни одна из строчек кода выше не пропущена. Нажмите F5, чтобы запустить программу. Выберите картинку и посмотрите на получившийся результат. Закройте программу.



В данном случае размеры изображения больше размера окна, и оно полностью не входит. Чтобы изменить вариант отображения, откройте свойства PictureBox. Параметру SizeMode установите значение Zoom. Снова запустите программу. Поэкспериментируйте с другими значениями этого параметра.

4. СОЗДАНИЕ КЛАССА ДЛЯ ФИЛЬТРОВ И ФИЛЬТРА «ИНВЕРСИЯ»

Каждый фильтр будем представлять в коде отдельным классом. С другой стороны все фильтры будут иметь абсолютно одинаковую функцию, запускающую процесс обработки и перебирающую в цикле все пиксели результирующего изображения. Исходя из этих соображений, создадим родительский абстрактный класс Filters, который и будет содержать эту функцию. Для этого создайте новый файл. Откройте окно SolutionExplorer (Ctrl+Alt+L), нажмите ПКМ по имени вашего проекта, выберите Add ->Class. В открывшемся окне введите имя класса - Filters. В SolutionExplorer появится файл Filters.cs, двойным щелчком ПКМ откройте файл для редактирования. Чтобы использовать классы для работы с изображениями, входящие в состав библиотеки базовых классов (BCL), в раздел объявления зависимостей добавьте строку using System.Drawing. Сделайте класс Filters абстрактным, добавив модификатор abstract. Код пустого абстрактного класса должен выглядеть так:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

namespace Filters_Ivanov
{
    abstract class Filters
    {

    }
}
```

В классе Filters создайте функцию processImage, принимающую на вход объект типа Bitmap и возвращающую объект типа Bitmap. В этой функции будет находиться общая для всех фильтров часть.

```
public Bitmap processImage(Bitmap sourceImage)
{
    Bitmap resultImage = new Bitmap(sourceImage.Width, sourceImage.Height);

    return resultImage;
}
```

На данный момент функция создает пустое изображение такого же размера, как и попадающееся ей на вход. Чтобы обойти все пиксели изображения, создайте два вложенных цикла от 0 до ширины и от 0 до высоты изображения. Внутри цикла с помощью метода SetPixel установите пикселю с текущими координатами значение функции calculateNewPixelColor.

```
Bitmap resultImage = new Bitmap(sourceImage.Width, sourceImage.Height);
for (int i = 0; i < sourceImage.Width; i++)
{
    for (int j = 0; j < sourceImage.Height; j++)
    {
        resultImage.SetPixel(i, j, calculateNewPixelColor(sourceImage, i, j));
    }
}
return resultImage;
```

VisualStudio подчеркивает имя calculateNewPixelColor, потому что она еще не создана. Создайте эту функцию в классе Filters и сделайте абстрактной. Данная функция будет вычислять значение пикселя отфильтрованного изображения, и для каждого из фильтров будет уникальной.

```
abstract class Filters
{
    protected abstract Color calculateNewPixelColor(Bitmap sourceImage, int x, int y);

    public Bitmap processImage(Bitmap sourceImage)
```

Цвет пикселя в VisualStudio представляется тремя (если не считать прозрачность) компонентами, каждая из которых может принимать значение от 0 до 255. В некоторых фильтрах результат может выходить за эти рамки, что приведет к падению программы. В классе Filters напишите функцию Clamp, чтобы привести значения к допустимому диапазону.

```
public int Clamp(int value, int min, int max)
{
    if (value < min)
        return min;
    if (value > max)
        return max;
    return value;
}
```

Создайте класс InvertFilter, наследник класса Filter, с переопределенной функцией calculateNewPixelColor.

```
protected override Color calculateNewPixelColor(Bitmap sourceImage, int x, int y)
{
}
```

В теле функции получите цвет исходного пикселя, а затем вычислите инверсию этого цвета, и верните как результат работы функции.

```
protected override Color calculateNewPixelColor(Bitmap sourceImage, int x, int y)
{
    Color sourceColor = sourceImage.GetPixel(x, y);
    Color resultColor = Color.FromArgb(255 - sourceColor.R,
                                       255 - sourceColor.G,
                                       255 - sourceColor.B);
    return resultColor;
}
```

Откройте графический редактор формы. Сделайте двойной щелчок ЛКМ по элементу меню «Инверсия». Создастся новая функция, которая будет вызываться при выборе элемента «Инверсия».

```
private void инверсияToolStripMenuItem_Click(object sender, EventArgs e)
{
}
```

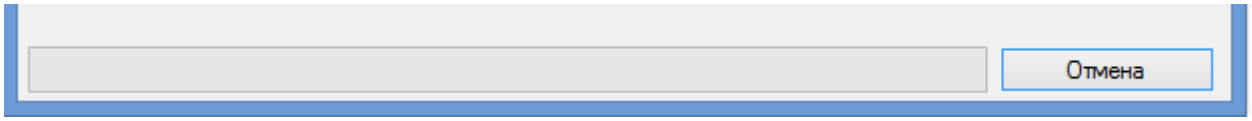
Создайте новый объект класса InvertFilter и инициализируйте его значением по умолчанию. Создайте новый экземпляр класса Bitmap для измененного фильтром изображения, и присвойте этому экземпляру результат функции processImage().

```
InvertFilter filter = new InvertFilter();
Bitmap resultImage = filter.processImage(image);
pictureBox1.Image = resultImage;
pictureBox1.Refresh();
```

Запустите программу, проверьте работоспособность фильтра. Должно получиться инвертированное изображение, как на рисунке ниже:

5. СОЗДАНИЕ ИНДИКАТОРА ПРОГРЕССА

Некоторые фильтры работают сравнительное долгое время, и желательно знать прогресс выполнения и иметь возможность безболезненного прекращения операции. Воспользуемся компонентами ProgressBar и BackgroundWorker для реализации этой функциональности. Для этого откройте ToolBox и перетащите на форму элементы BackgroundWorker, ProgressBar и Button, измените надпись на кнопке на «Отмена». Значок BackgroundWorker появится под окном формы, т.к. компонент не относится к пользовательскому интерфейсу и не является видимым.

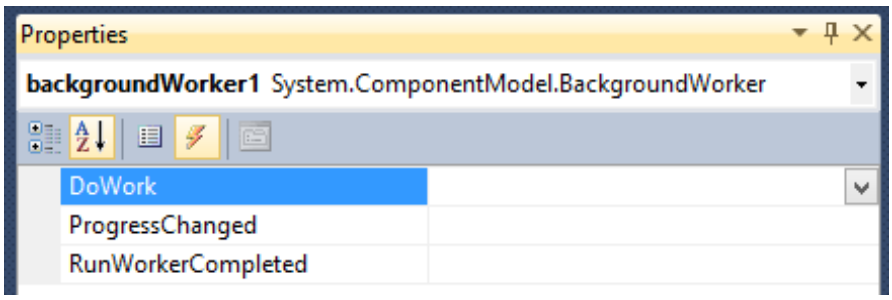


BackgroundWorker - (System.ComponentModel.BackgroundWorker) класс, предназначенный для создания и управления работой потоков. Он предоставляет следующие возможности:

- Стандартизированный протокол создания, сигнализации о ходе выполнения и завершения потока.
- Возможность прерывания потока.
- Возможность обработки исключений в фоновом потоке.
- Возможность связи с основным потоком через сигнализацию о ходе выполнения и окончания.

Таким образом, при использовании BackgroundWorker нет необходимости включения try/catch в рабочий поток и есть возможность выдачи информации в основной поток без явного вызова Control.Invoke.

Откройте свойства BackgroundWorker, установите параметру WorkerReportProgress значение True, параметру WorkerSupportsCancellation тоже значение True, переключитесь на вкладку Events, на которой расположены доступные события для элемента. Двойным щелчком создайте функцию DoWork.



Добавьте в функцию код, который будет выполнять код одного из фильтров.

```
private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
{
        Bitmap newImage = ((Filters)e.Argument).processImage(image, backgroundWorker1);
        if (backgroundWorker1.CancellationPending != true)
            image = newImage;
}
```

Подключите зависимость System.ComponentModel и измените объявление функции processImage() в классе Filters.

```
public Bitmap processImage(Bitmap sourceImage, BackgroundWorker worker)
{
```

В функции processImage во внешний цикл добавьте сроку, которая будет сигнализировать элементу BackgroundWorker о текущем прогрессе. Используйте приведения типов для корректных расчетов.

```
for (int i = 0; i < sourceImage.Width; i++)
{
    worker.ReportProgress((int)((float)i / resultImage.Width * 100));
    if (worker.CancellationPending)
        return null;
```

Аналогично созданию функции DoWork создайте функцию ProgressChanged. Добавьте строку кода, которая будет изменять цвет полосы.

```
private void backgroundWorker1_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    progressBar1.Value = e.ProgressPercentage;
}
```

Создайте функцию, которая будет визуализировать обработанное изображение на форме и обнулять полосу прогресса.

```
private void backgroundWorker1_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
    if (!e.Cancelled)
    {
        pictureBox1.Image = image;
        pictureBox1.Refresh();
    }
    progressBar1.Value = 0;
}
```

Измените функцию вызова фильтра инверсии, чтобы фильтр запускался в отдельном потоке.

```
private void инверсияToolStripMenuItem_Click(object sender, EventArgs e)
{
    Filters filter = new InvertFilter();
    backgroundWorker1.RunWorkerAsync(filter);
}
```

Сделайте двойной клик ЛКМ по кнопке «Отмена» для создания функции, выполняющей по нажатию кнопки. Используйте функцию CancelAsyn с класса BackgroundWorker, чтобы остановить выполнение фильтра.

```
private void button1_Click(object sender, EventArgs e)
{
    backgroundWorker1.CancelAsync();
}
```

6. МАТРИЧНЫЕ ФИЛЬТРЫ

Главная часть матричного фильтра – ядро. Ядро – это матрица коэффициентов, которая покомпонентно умножается на значение пикселей изображения для получения требуемого результата (не то же самое, что матричное умножение, коэффициенты матрицы являются весовыми коэффициентами для выбранного подмассива изображения).

Создайте класс MatrixFilter, содержащий в себе двумерный массив kernel.

```
class MatrixFilter : Filters
{
    protected float[,] kernel = null;
    protected MatrixFilter() { }
    public MatrixFilter(float[,] kernel)
    {
        this.kernel = kernel;
    }
}
```

Создайте функцию calculateNewPixelColor, которая будет вычислять цвет пикселя на основании своих соседей. Первым делом найдите радиусы фильтра по ширине и по высоте на основании матрицы.

```
protected override Color calculateNewPixelColor(Bitmap sourceImage, int x, int y)
{
    int radiusX = kernel.GetLength(0) / 2;
    int radiusY = kernel.GetLength(1) / 2;
```

Создайте переменные типа float, в которых будут храниться цветовые компоненты результирующего цвета. Создайте два вложенных цикла, которые будут перебирать окрестность пикселя. В каждой из точек окрестности вычислите цвет, умножьте на значение из ядра и прибавьте к результирующим компонентам цвета. Чтобы на граничных пикселях не выйти за границы изображения, используйте функцию Clamp.

```
float resultR = 0;
float resultG = 0;
float resultB = 0;
for (int l = -radiusY; l <= radiusY; l++)
    for (int k = -radiusX; k <= radiusX; k++)
    {
        int idX = Clamp(x + k, 0, sourceImage.Width - 1);
        int idY = Clamp(y + l, 0, sourceImage.Height - 1);
        Color neighborColor = sourceImage.GetPixel(idX, idY);
        resultR += neighborColor.R * kernel[k + radiusX, l + radiusY];
        resultG += neighborColor.G * kernel[k + radiusX, l + radiusY];
        resultB += neighborColor.B * kernel[k + radiusX, l + radiusY];
    }
```

Тщательно разберите приведенный выше код. Переменные x и y – координаты текущего пикселя. Переменные l и k принимают значения от -radius до radius и означают положение точки в матрице ядра, если начало отсчета поместить в центр матрицы. В переменных idX и idY хранятся координаты пикселей-соседей пикселя x,y, для которого происходит вычисления цвета.

В качестве результата работы функции создайте экземпляр класса Color, состоящий из вычисленных вами компонент цвета. Используйте функцию Clamp, чтобы все значения компонент были в допустимом диапазоне.

```
return Color.FromArgb(
    Clamp((int)resultR, 0, 255),
    Clamp((int)resultG, 0, 255),
    Clamp((int)resultB, 0, 255)
    );
```

Создайте класс BlurFilter – наследник класса MatrixFilter. Переопределите конструктор по умолчанию, в котором создайте матрицу 3*3 со значением 1/9 в каждой ячейке.

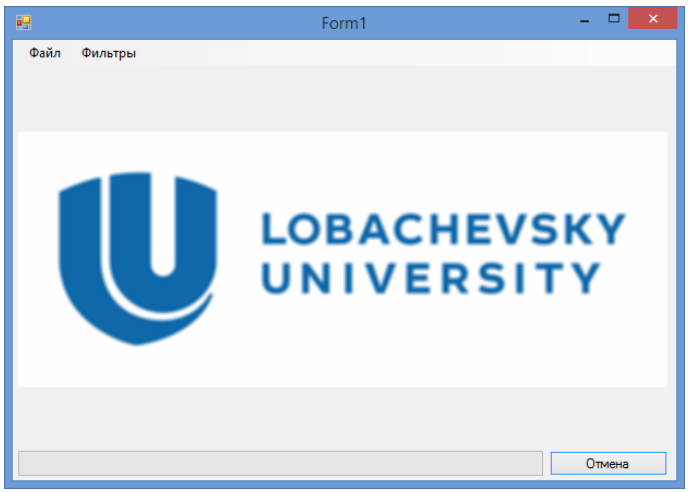
```
class BlurFilter : MatrixFilter
{
    public BlurFilter()
    {
        int sizeX = 3;
        int sizeY = 3;
        kernel = new float[sizeX, sizeY];
        for (int i = 0; i < sizeX; i++)
            for (int j = 0; j < sizeY; j++)
                kernel[i, j] = 1.0f / (float)(sizeX * sizeY);

    }
}
```

На форме в панели матричных фильтров добавьте элемент «Размытие», создайте двойным щелчком функцию для ее применения.

```
private void размытиеToolStripMenuItem_Click(object sender, EventArgs e)
{
    Filters filter = new BlurFilter();
    backgroundWorker1.RunWorkerAsync(filter);
}
```

Проверьте результат работы. Для проверки результата не берите изображение большого разрешения, потому что матричные фильтры работают намного дольше точечных. Также при использовании большого изображения оно будет уменьшаться до размеров pictureBox из-за чего эффект размытия будет менее заметен.



Более совершенным фильтром для размытия изображений является фильтр Гаусса, коэфициенты для которого рассчитываются по формуле Гаусса:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

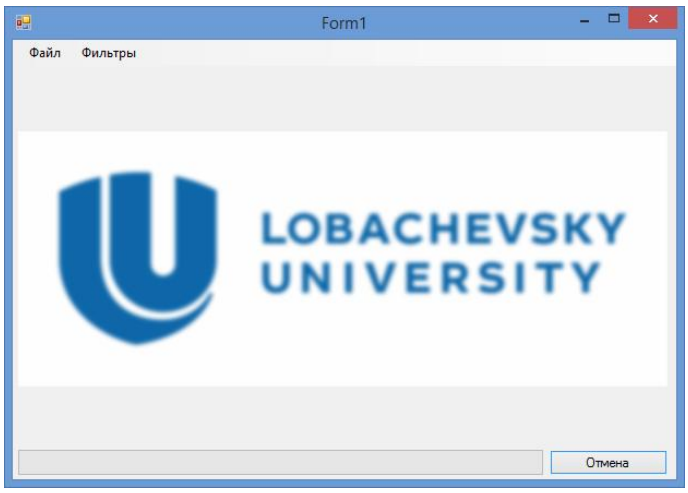
Создайте новый класс GaussianFilter – наследник класса MatrixFilter. Создайте функцию, которая будет рассчитывать ядро преобразования по формуле Гаусса

```
public void createGaussianKernel(int radius, float sigma)
{
    // определяем размер ядра
    int size = 2 * radius + 1;
    // создаем ядро фильтра
    kernel = new float[size, size];
    // коэффициент нормировки ядра
    float norm = 0;
    // рассчитываем ядро линейного фильтра
    for (int i = -radius; i <= radius; i++)
        for (int j = -radius; j <= radius; j++)
        {
            kernel[i + radius, j + radius] = (float)(Math.Exp(-(i * i + j * j) / (sigma * sigma)));
            norm += kernel[i + radius, j + radius];
        }
    // нормируем ядро
    for (int i = 0; i < size; i++)
        for (int j = 0; j < size; j++)
            kernel[i, j] /= norm;
}
```

Создайте конструктор по умолчанию, который будет раздавать фильтр размером 7*7 и с коэффициентом сигма, равным 2.

```
public GaussianFilter()
{
    createGaussianKernel(3, 2);
}
```

Аналогично остальным фильтрам допишите код для запуска фильтра. Протестируйте.



7. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

Применив полученные знания, добавьте в программу следующие фильтры:

- Создайте точечный фильтр, переводящий изображение из цветного в черно-белое. Для этого создайте фильтр GrayScaleFilter – наследник класса Filters, и создайте функцию calculateNewPixelColor, которая переводит цветное изображение в черно-белое по следующей формуле:

  $$Intensity = 0.36 * R + 0.53 * G + 0.11 * B$$

  Полученное значение записывается во все три канала выходного пикселя.
- Создайте точечный фильтр «Сепия», переводящий цветное изображение в изображение песочно-коричневых оттенков. Для этого найдите интенсивность, как у черно-белого изображения. Цвет выходного пикселя задайте по формуле:

  $$R = Intensity + 2 * k; G = Intensity + 0.5 * k; B = Intensity - 1 * k$$

  Подберите коэффициент k для наиболее оптимального на ваш взгляд оттенка сепии, не забудьте при написании фильтра использовать функцию Clamp для приведения всех значений к допустимому интервалу.
- Создайте точечный фильтр, увеличивающий яркость изображения. Для этого в каждый канал пикселя прибавьте константу, позаботьтесь о допустимости значений.
- Создайте матричный фильтр Собеля. Оператор Собеля представляет собой матрицу 3*3. Оператор Собеля вычисляет приближенное значение градиента яркости изображения. Ниже представлены операторы Собеля, ориентированные по разным осям:

$$\text{По оси Y:} \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \quad \text{По оси } X: \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

- Создайте матричный фильтр, повышающий резкость изображения. Матрица для данного фильтра задается следующим образом:

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

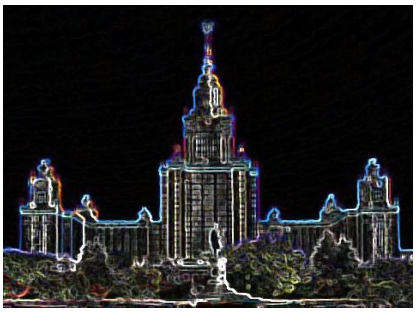
8. СПИСОК ФИЛЬТРОВ

ТИСНЕНИЕ

Ядро фильтра: $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix}$ + сдвиг по яркости + нормировка.

СВЕТЯЩИЕСЯ КРАЯ

Медианный фильтр + выделение контуров + фильтр «максимума»

ПЕРЕНОС/ПОВОРОТ

Перенос: $\begin{bmatrix} x(k,l) = k + 50; & k,l - \text{индексы для которых вычисляется цвет} \\ y(k,l) = l; & x,y - \text{новые индексы по которым берется цвет} \end{bmatrix}$


Поворот: $\begin{bmatrix} x(k,l) = (k - x0)cos(\mu) - (l - y0)sin(\mu) + x0; \\ y(k,l) = (k - x0)sin(\mu) + (l - y0)cos(\mu) + y0; \end{bmatrix}$, где $(x0, y0) -$ центр поворота, $\mu -$ угол поворота

«ВОЛНЫ»

Волны 1: $\begin{bmatrix} x(k,l) = k + 20sin(2\pi l / 60); \\ y(k,l) = l; \end{bmatrix}$

Волны 2: $\begin{bmatrix} x(k,l) = k + 20sin(2\pi k / 30); \\ y(k,l) = l; \end{bmatrix}$

ЭФФЕКТ «СТЕКЛА»

$$\begin{bmatrix} x(k,l) = k + (rand(1) - 0.5) * 10; \\ y(k,l) = l + (rand(1) - 0.5) * 10; \end{bmatrix}$$

MOTION BLUR

Ядро фильтра: $\frac{1}{n}\begin{bmatrix} 1 & 0 & 0 & & & & \\ 0 & 1 & 0 & \cdots & & 0 & \\ 0 & 0 & 1 & & & & \\ & \vdots & & \ddots & & \vdots & \\ & & & & 1 & 0 & 0 \\ & 0 & & \cdots & 0 & 1 & 0 \\ & & & & 0 & 0 & 1 \end{bmatrix}$, n – количество единиц, т. е. количество столбцов или строк

РЕЗКОСТЬ

Ядро фильтра: $\begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}$

ВЫДЕЛЕНИЕ ГРАНИЦ

Оператор Щарра:

По оси Y: $\begin{pmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{pmatrix}$ По оси X: $\begin{pmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{pmatrix}$

Оператор Прюитта:

По оси Y: $\begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ По оси X: $\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$

9. ДОПОЛНИТЕЛЬНЫЕ ЗАДАНИЯ

Для развития навыков программирования на языке C# в среде VisualStudio, предлагается расширить программу дополнительной функциональностью.

- Реализовать возможность сохранять изображения. Для сохранения файлов существует класс SaveFileDialog, который работает аналогично рассмотренному в работе OpenFileDialog. Дополнительную информацию по его использованию можно найти на сайте Microsoft [1].
- Реализовать возможность изменения размера окна, при которой элементы будут перемещаться или растягиваться пропорционально изменению размера окна.
- Реализовать отмену последнего или нескольких последних действий.

10. ЗАДАНИЯ ДЛЯ СДАЧИ ЛАБОРАТОРНОЙ РАБОТЫ

- Выберите из раздела «Список фильтров» 2 фильтра, где операции производятся над индексами пикселей и два матричных фильтра, реализуйте.
- Реализуйте операции мат. морфологии dilation, erosion, opening, closing. Выберите одну из top hat, black hat, grad, реализуйте.
- Реализуйте медианный фильтр
- Добавьте возможность задать структурный элемент для операций мат. морфологии.
- Реализуйте линейное растяжение
- Реализуйте один из фильтров: «Серый мир», «Идеальный отражатель», «Коррекция с опорным цветом»

ССЫЛКИ:

Практическое руководство. Сохранение файлов с помощью компонента SaveFileDialog
https://msdn.microsoft.com/ru-ru/library/sfezx97z%28v=vs.110%29.aspx