Лабораторная работа №2:
Работа с цветами

Цель:

Целью данной работы является изучение базовых операций над цветовыми каналами изображений и реализация некоторых фильтров на их основе.

Задание:

Необходимо разработать приложение Windows Forms, способное осуществлять:

1. загрузку и отображение двух изображений по выбору пользователя;
2. возможность применения базовых операций к загруженным изображениям;
3. возможность применения оконных и комбинированных фильтров к загруженным изображениям;

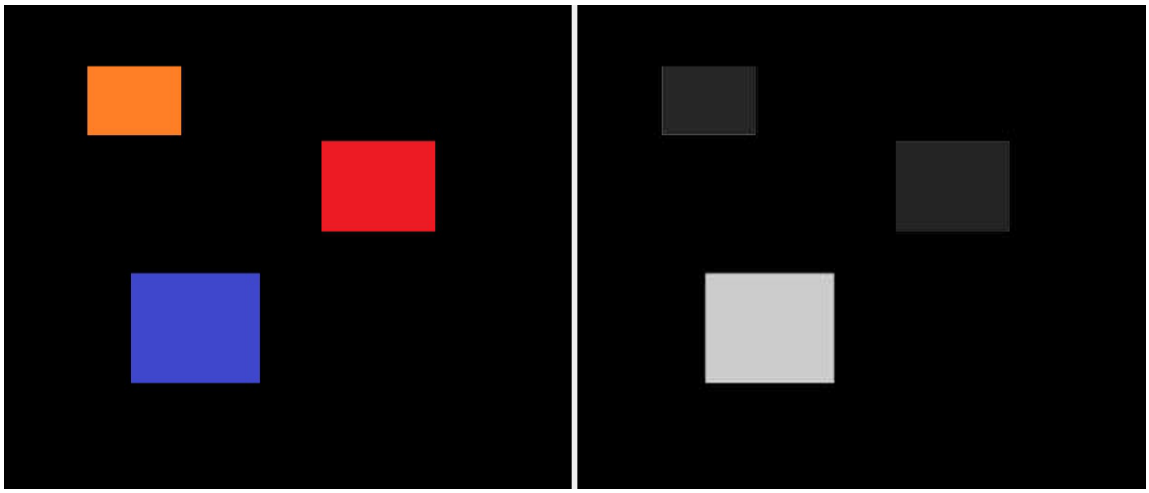
Базовые операции:

Получить данные одного цветового канала можно при помощи метода:

```
var channel = sourceImage.Split()[channelIndex];
```

где `channelIndex` – номер требуемого канала.

Пример получения значений синего канала:



Чем светлее участки изображения на изображении справа, тем более высокие значения содержались в синем цветовом канале на изображении слева.

Объединить слои можно при помощи функции:

```
VectorOfMat vm = new VectorOfMat();
vm.Push(channels[0]); vm.Push(channels[1]); vm.Push(channels[2]);
CvInvoke.Merge(vm, destImage);
```

Доступ к пикселям изображения можно получить следующим образом:

```
var image = new Image<Bgr, byte>(sourceImage.Size);
...
// x, y - координаты пикселя
// 0 - номер канала, синий
byte color = image.Data[y, x, 0];
```

Конвертация изображения в чёрно-белое может быть проведена по следующей формуле:

```
// создание "серого" изображения
var grayImage = new Image<Gray, byte>(sourceImage.Size);

...

// заполнение значений "серого" изображения должно осуществляться в цикле
grayImage.Data[y, x, 0] = Convert.ToByte(0.299 * sourceImage.Data[y, x, 2] + 0.587 *
sourceImage.Data[y, x, 1] + 0.114 * sourceImage.Data[y, x, 0]);
```

где 0.299, 0.587 и 0.114 – коэффициенты влияния компонент RGB на итоговую интенсивность цвета, а sourceImage – исходное изображение.

Пример конвертации:

Изменение яркости и контраста осуществляется по следующим формулам:

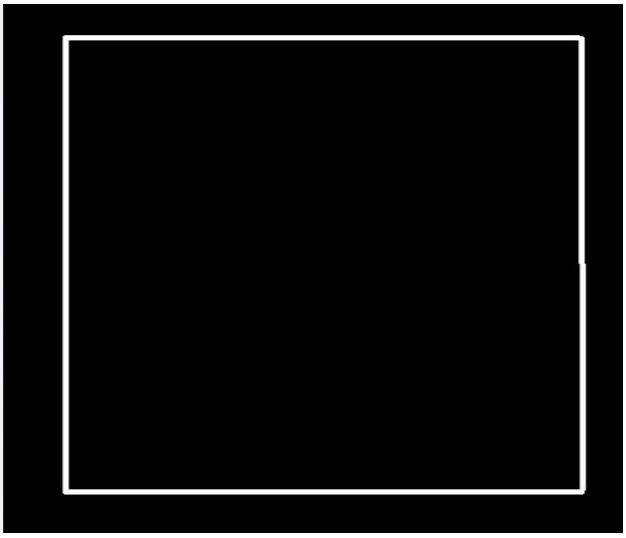
$$color = color + brightness$$
$$color = color * contrast$$

Пример изменения контраста в 2 раза и яркости на 50 единиц:

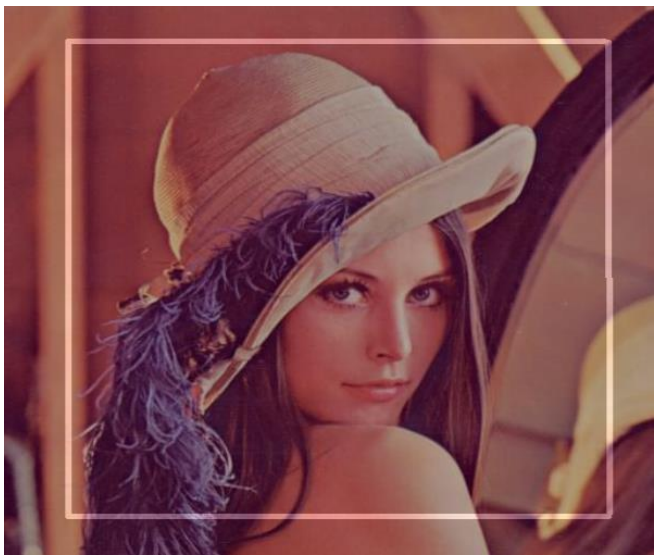
Операции над цветами выполняются по тем же правилам, что и бинарные. Например, сложение цветов можно реализовать следующим образом:

```
private double AddColors(double color1, double color2)
{
       if (color1 + color2 > 255) return 255;
       else if (color1 + color2 < 0) return 0;
       else return color1 + color2;
}
```

Производной операцией от сложения может быть сложение с коэффициентами участия изображений. Например, результатом сложения изображений:

с коэффициентами 0.7 и 0.3 будет:



Для работы с цветовым тоном, насыщенностью и яркостью изображения удобно использовать формат HSV. Преобразовать изображение к цветовому формату, отличному от RGB, можно при помощи стандартной функции:

```
var hsvImage = bgrImage.Convert<Hsv, byte>();
```

Например, цветовой формат HSV удобно использовать для изменения цветового тона. На приведённом ниже изображении цветовой тон установлен в значение 150 (0-360):


Фильтрация изображений:

Простым примером «художественного» фильтра, основанного на базовых операциях, является Sepia:


Формула расчёта значений пикселей выглядит следующим образом:

$$red = red * 0.393 + green * 0.769 + blue * 0.189$$
$$green = red * 349 + green * 0.686 + blue * 0.168$$
$$blue = red * 0.272 + green * 0.534 + blue * 0.131$$

Коэффициенты перед цветами подобраны эмпирическим путём для получения нужного эффекта.

Реализация размытия изображения (blur) с использованием медианного фильтра состоит из следующих этапов:

1. Для каждого пикселя и 8 его соседей выполняется поиск среднего значения:

2. Использование найденного значения в качестве нового значения цвета.

Пример использования медианного фильтра для размытия чёрно-белого изображения:


Для хранения, сортировки и нахождения среднего значения можно использовать контейнер List:

```
//создание списка для хранения целых значений
List<int> window = new List<int>();
…
//добавление значения
window.Add(color);
…
//сортировка списка
window.Sort();
```

Реализация повышения резкости изображения (sharpen) с использованием оконного фильтра состоит из следующих этапов:

1. Для каждого пикселя и 8 его соседей выполняется умножение на соответствующий элемент матрицы:
$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$
2. Полученные значения суммируются и нормируются путём деления на сумму элементов матрицы:
$$color[i,j] = \frac{(\sum_n^0 color[i,j] * matrix[i',j'])}{\sum_n^0 matrix[i',j']}$$
3. Результат используется в качестве нового значения цвета.

Пример работы:


Изменяя матрицу оконного фильтра, можно добиться различных эффектов. Например, используя матрицу вида:

$$\begin{pmatrix} -4 & -2 & 0 \\ -2 & 1 & 2 \\ 0 & 2 & 4 \end{pmatrix}$$

можно получить эффект «тиснение» (embos):

А используя шаблон матрицы вида:

$$\begin{pmatrix} 0 & 0 & 0 \\ -4 & 4 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

можно обнаружить грани:

Комбинируя простые фильтры, можно создавать более сложные эффекты. Так, для создания простой вариации «акварельного» фильтра (watercolor), можно применить следующий подход:

1. Коррекция контраста и яркости изображения.
2. Размытие изображения.
3. Сложение изображения с «маской».

Пример:

исходное изображение:

маска:

результат:

В заключении, рассмотрим cartoon filter. Последовательность действий:

1. Создание чёрно-белой копии изображения.
2. Размытие изображения.
3. Бинаризация изображения по пороговому значению.
4. Выполнение операции «пересечение» (and) между исходным изображением и результатом бинаризации.

результат:

Выполнить бинаризацию можно при помощи функции:

```
var edges = sourceImage.Convert<Gray, byte>();
edges = edges.ThresholdAdaptive(new Gray(100), AdaptiveThresholdType.MeanC,
                                                ThresholdType.Binary, 3, new Gray(0.03));
```

Задание:

Реализовать программное средство, позволяющее отображать в одном окне два изображения, «оригинальное» слева и «результат обработки» справа. Реализовать интерфейс, позволяющий по нажатию на соответствующие кнопки выполнять следующие операции:

1. Вывод значений одного из трёх цветовых каналов по выбору пользователя.
2. Вывод чёрно-белой версии изображения.
3. Вывод Sepia версии изображения.
4. Вывод изображения с возможностью изменения его яркости и контраста.
5. Вывод результатов логических операций «дополнение», «исключение» и «пересечение», с возможностью выбора изображения для соответствующей операции.
6. Вывод изображения преобразованного в формат HSV, с возможностью изменения значений HSV.
7. Вывод изображений с применением к ним медианного размытия.
8. Вывод изображений с применением к ним оконного фильтра, с возможностью изменения матрицы фильтра из формы приложения.
9. Вывод изображений с применением к ним «акварельного фильтра», а так же, возможностью выбора яркости, контраста и параметров смешивания изображений.
10. Вывод изображений с применением к ним «cartoon filter» и возможностью изменения порога преобразования изображения.

Рекомендации:

1. Соблюдать codestyle (форматирование кода, наименование переменных, полей и методов и т.д.) для облегчения чтения кода и отладки.
2. Работу с изображением вынести в отдельный класс. В форме не должно быть никакой логики, кроме обработки событий из UI и вызовов соответствующих методов.
3. Каждый фильтр реализовать как отдельный метод, который в качестве результата возвращает объект класса Image. Переиспользовать методы при необходимости и избегать дублирования кода.
4. Грамотно задавать диапазон значений для фильтров на UI. Например, для порогового фильтра диапазон значений определяется разрядность изображения, и для 8-битного серого изображения диапазон значений будет равен [0; 255].
5. Использовать конструкцию try-catch для обработки ошибок (при открытии файлов, проверке входных параметров и др.)