Лабораторная работа №3:
Аффинные преобразования и гомография изображений

Цель:

Целью данной работы является изучение базовых операций над геометрией изображений и их применение к некоторым задачам обработки изображений.

Задание:

Необходимо разработать приложение Windows Forms, способное осуществлять:

1. загрузку и отображение двух изображений по выбору пользователя;
2. возможность применения аффинных преобразований к загруженным изображениям;
3. возможность проекции области одного изображения на другое.

Аффинные преобразования:

Одним из базовых аффинных преобразований является масштабирование (scaling). При масштабировании координаты каждого пикселя изображения домножаются на соответствующий коэффициент масштабирования:

$$pixelNewX = pixelX * sX$$

$$pixelNewY = pixelY * sY$$

где:

pixelNewX/Y – новые координаты пикселя;

pixelX/Y – старые координаты пикселя;

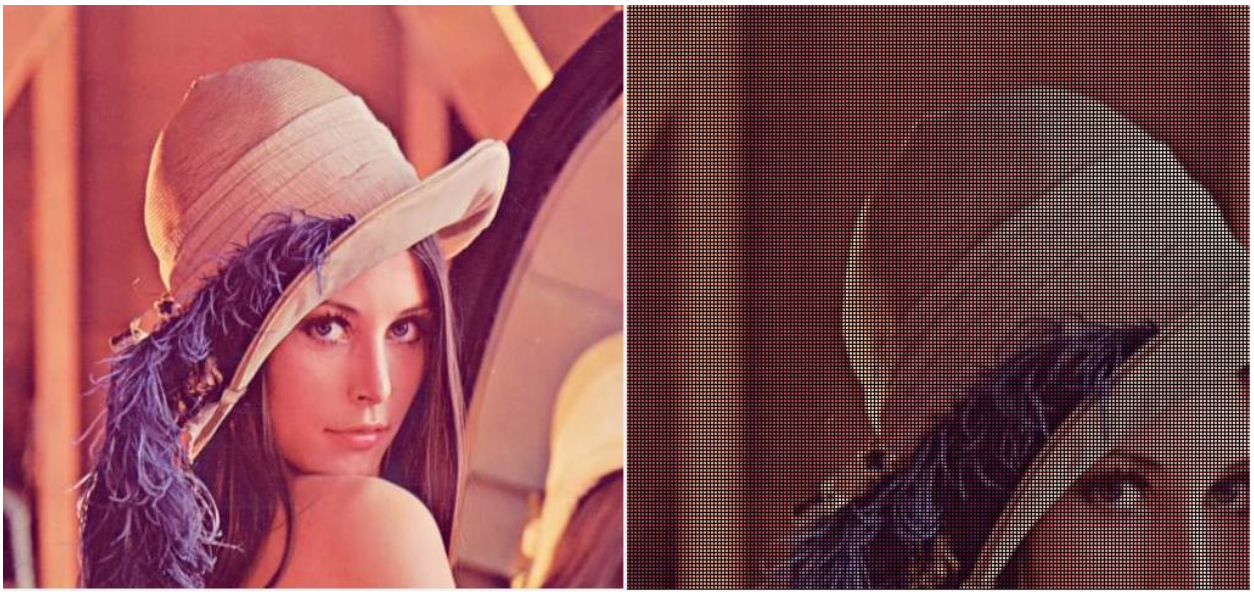
sX/sY – коэффициенты масштабирования по горизонтали и вертикали.

Пример кода:

```
// коэффициенты масштабирования
float sX = 1.5f;
float sY = 1.5f;
// создание нового изображения
// высота и ширина нового изображения увеличивается в sX и sY раз соответственно
var newImage = new Image<Bgr, byte>((int)(sourceImage.Width * sX), (int)(sourceImage.Height * sY));
for (int x = 0; x < sourceImage.Width; x++)
{
	for (int y = 0; y < sourceImage.Height; y++)
	{
		// вычисление новых координат пикселя
		int newX = (int)(x * sX);
		int newY = (int)(y * sY);
		// копирование пикселя в новое изображение
		newImage[newY, newX] = sourceImage[y, x];
	}
}
```

Результат преобразования при коэффициентах sX = 1, sY=0.5:

Результат преобразования при коэффициентах sX = 1.5, sY=1.5:



Другим аффинным преобразованием является сдвиг (shearing). Суть преобразования заключается в сдвиге каждой линии изображения параллельно одной из его сторон на величину, пропорциональную дистанции до соответствующей стороны. В общем виде горизонтальный сдвиг относительно нижней границы изображения можно записать как:

$$pixelNewX = pixelX + shift * (height - pixelY)$$

$$pixelNewY = pixelY$$

Где:

pixelNewX/Y – новые координаты пикселя

pixelX/Y – старые координаты пикселя

shift – сдвиг

height – высота изображения

Результат преобразования при *shift* = 0,25:

Результат сдвига относительно левой границы изображения при *shift* = 0.25:

Простым примером применения сдвига является поворот изображения на произвольный угол путём последовательного применения горизонтального, вертикального и снова горизонтального сдвигов.

Третьим аффинным преобразованием является поворот (rotation). В общем виде поворот можно записать через тригонометрические функции:

$$pixelNewX = \cos(angle) * (pixelX - centerX) - \sin(angle) * (pixelY - centerY) + centerX$$

$$pixelNewY = \sin(angle) * (pixelX - centerX) + \cos(angle) * (pixelY - centerY) + centerY$$

Где:

pixelNewX/Y – новые координаты пикселя

centerX/Y – координаты центра вращения

pixelX/Y – старые координаты пикселя

angle – угол поворота изображения в радианах

Результат работы при *angle* = Pi/6:

Как видно из примера, вращение происходит вокруг левой верхней точки изображения (*centerX* = 0, *centerY* = 0).

Последнее из рассматриваемых аффинных преобразований – отражение (reflection). Отражение может быть записано как:

$$pixelNewX = pixelX * qX + width$$

$$pixelNewY = pixelY * qY + height$$

Где:

pixelNewX/Y – новые координаты пикселя

height/width – ширина и высота изображения

qX/qY – параметры отражения, диапазон значений представлен ниже:

Билинейная фильтрация:

При выполнении преобразований над изображением возможны потери данных из-за погрешностей округления, а так же появление новых пикселей. Для заполнения недостающих значений может быть использована интерполяция их соседей. Билинейная интерполяция является одной из простейших и может быть записана как:

$$floorX = Floor(x);\quad floorY = Floor(y);$$

$$ratioX = x - floorX;\ \ ratioY = y - floorY;$$

$$inversXratio = 1 - ratioX;\ inversYratio = 1 - ratioY;$$

$$resColor[i,j] = (image\ [baseX, BaseY] * inversXratio + \ image\ [baseX + 1, BaseY] * ratioX) * inversYratio + \ (image\ [baseX, BaseY + 1] * inversXratio + \ image\ [baseX + 1, BaseY + 1] * ratioX) * ratioY$$

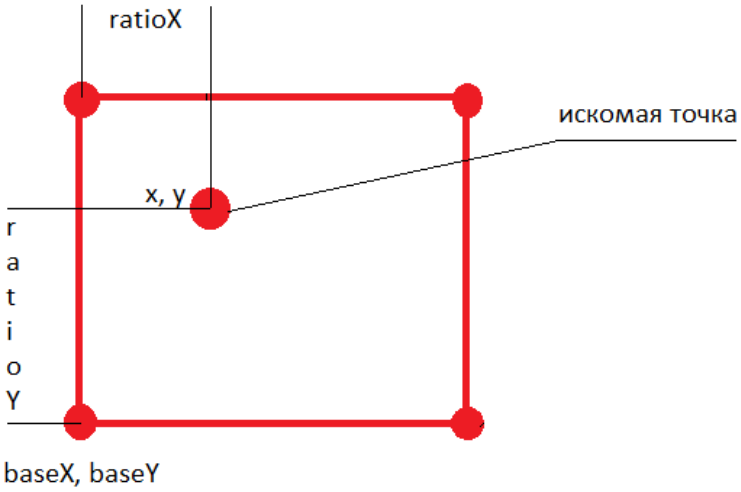
где:

floorX/Y – округлённые до ближайшего нижнего значения координаты искомой точки;

ratioX/Y – дистанция от ближайшей точки исходного изображения до искомой;

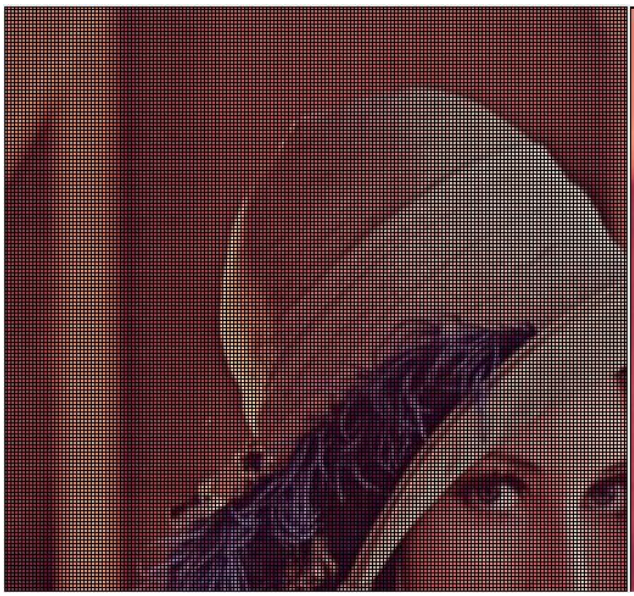
baseX/Y – координаты точек, ближайших к искомой, на исходном изображении;

image – исходное изображение.



Пример работы при масштабировании с параметрами *sX* = 1.5, *sY*=1.5:

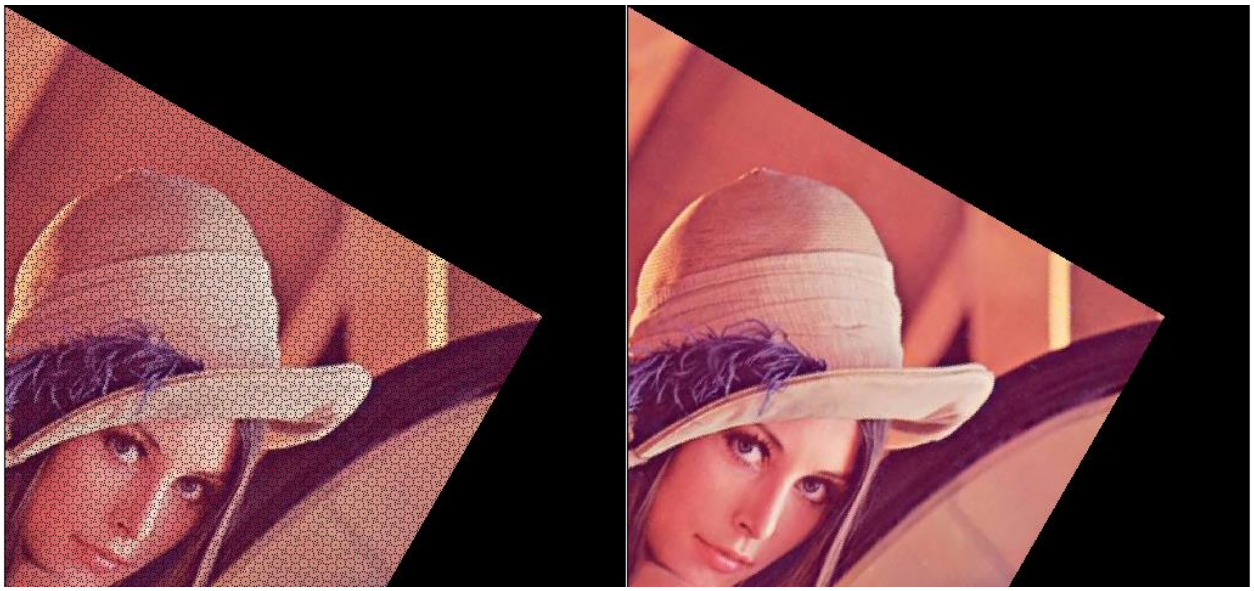
без фильтрации



с фильтрацией

Пример работы поворота при *angle* = Pi/6:

без фильтрации с фильтрацией



Гомография изображения:

В данном случае под гомографией понимается проецирование изображения, либо его фрагмента на произвольную плоскость. В библиотеке EmguCV гомографию можно осуществить следующим образом:

```
var srcPoints = new PointF[]
{
        // четыре точки на исходном изображении
};
var destPoints = new PointF[]
{
    // плоскость, на которую осуществляется проекция,
    // задаётся четыремя точками
    new PointF(0, 0),
    new PointF(0, sourceImage.Height - 1),
    new PointF(sourceImage.Width - 1, sourceImage.Height - 1),
    new PointF(sourceImage.Width - 1, 0)
};

// функция нахождения матрицы гомографической проекции
var homographyMatrix = CvInvoke.GetPerspectiveTransform(srcPoints, destPoints);

var destImage = new Image<Bgr, byte>(sourceImage.Size);
CvInvoke.WarpPerspective(sourceImage, destImage, homographyMatrix, destImage.Size);
```

Пример работы:

Для определения координат пикселя при нажатии на изображение необходимо назначить на компонент ImageBox обработчик нажатия мыши:

```
imageBox1.MouseClick += new MouseEventHandler(imageBox1_MouseClick);
```

и внутри обработчика вычислить координаты с помощью входных параметров:

```
private void imageBox1_MouseClick(object sender, MouseEventArgs e)
{
    int x = (int)(e.Location.X / imageBox1.ZoomScale);
    int y = (int)(e.Location.Y / imageBox1.ZoomScale);
}
```

Для обозначения точек на изображении можно рисовать на нём небольшие круги. Пример кода:

```
Point center = new Point(x, y);
int radius = 2;
int thickness = 2;
var color = new Bgr(Color.Blue).MCvScalar;
// функция, рисующая на изображении круг с заданными параметрами
CvInvoke.Circle(sourceImage, center, radius, color, thickness);
```

Задание:

Разработать программу, позволяющую отображать в одном окне два изображения: оригинальное изображение слева и результат обработки справа. Реализовать интерфейс, позволяющий по нажатию на соответствующие кнопки выполнять следующие операции:

1. масштабирование изображения с параметрами, вводимыми пользователем. Размер результирующего изображения должен изменяться в соответствии с параметрами масштабирования;
2. осуществлять сдвиг изображения на произвольное значение;
3. поворот изображения относительно выбранной пользователем точки на заданный пользователем угол;
4. отражение изображения одним из четырёх способов. При отражении, размер результирующего изображения не должен изменяться;
5. применить билинейную фильтрацию при выполнении поворота и масштабирования для устранения графических дефектов;
6. осуществить проекцию фрагмента изображения на произвольную плоскость. Добавить возможность выбора фрагмента и плоскости пользователем через указание четырех точек с помощью мыши.

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. Каждый фильтр реализовать как отдельный метод, который в качестве результата возвращает объект класса Image. В форме не должно быть никакой логики, кроме обработки событий из UI и вызовов соответствующих методов.
3. Грамотно задавать диапазон значений для фильтров на UI.
4. Использовать конструкцию try-catch для обработки ошибок (при открытии файлов, проверке входных параметров и др.)
5. Для математических операций использовать методы из пространства имен System.Math.
6. При вычислении новых координат пикселя проверять, входят ли они в новые границы изображения
7. Для преобразования вещественных типов данных в целочисленные необходимо использовать:

   – методы класса System.Convert: ToByte, ToInt и др. (округляют до ближайшего целого)
   – метод Math.Floor или приведение типов: (int), (byte) и др. (округляют до меньшего целого)