НИЯУ МИФИ, факультет Кибернетики, Каф. 17

Компьютерная обработка изображений

Лекция 3: Геометрические преобразования изображений.

Сафонов И.В., Крыжановский К.А., Егорова М.А.

2011

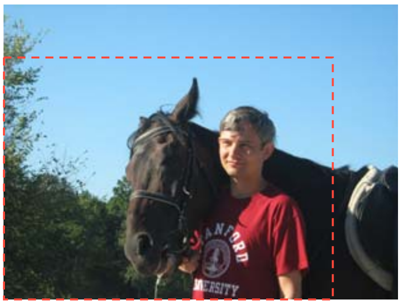
Геометрические преобразования

- Кадрирование
- Масштабирование
- Поворот
- Зеркальные отражения
- Скос
- Коррекция геометрических искажений

Аффинные преобразования

Кадрирование

Кадрирование (*cropping*) – вырезание из изображения прямоугольного фрагмента. Выполняют с целью выделения требуемой части изображения, изменения соотношения сторон (*aspect ratio*) и/или улучшения композиции.



Преобразования на плоскости

Любые преобразования на плоскости задаются с помощью матрицы трансформации для однородных координат:

$$[x'_i \quad y'_i \quad 1] = [x_i \quad y_i \quad 1] \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

где x_i, y_i - координаты пикселов исходного, а x'_i, y'_i – координаты пикселов трансформированного изображений.

Наивным способом трансформации изображения является умножение координат его пикселов на прямую матрицу трансформации, для того чтобы найти координаты этих пикселов в новом изображении. Однако в общем случае такой подход неправильный.

Пример поворота изображения на 20 градусов путем умножения координат исходного изображения на прямую матрицу трансформации: "дырки" на изображении.

Общая схема преобразования

1. Координаты угловых пикселов исходного изображения умножаются на прямую матрицу трансформации для нахождения координат угловых пикселов и размера нового изображения:

$$[x'_i \quad y'_i \quad 1] = [x_i \quad y_i \quad 1] \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

2. Координаты каждого пиксела нового изображения умножают на обратную матрицу трансформации для получения координаты соответствующего ему пиксела в исходном изображении:

$$[x \quad y \quad 1] = [x' \quad y' \quad 1] \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}^{-1}$$

3. Если координата попала в пределы габаритов исходного изображения, то значение пиксела в новом изображении определяют с помощью интерполяции по соседним пикселам исходного изображения.

Масштабирование

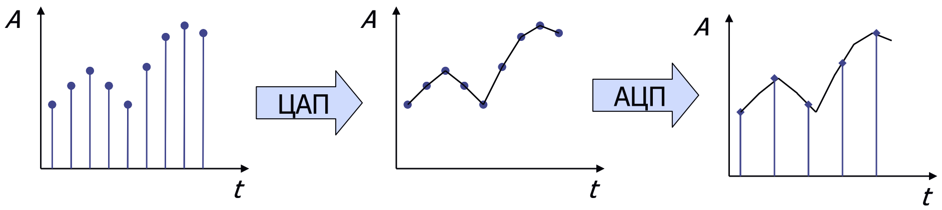
Масштабирование (*scaling*) – изменение размеров (*resizing*) изображения. Различают увеличение размеров (*upsizing*) и уменьшение размеров (*downsizing*).

Матрицы преобразования для масштабирования:

Прямая $\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$ Обратная $\begin{bmatrix} \frac{1}{Sx} & 0 & 0 \\ 0 & \frac{1}{Sy} & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Перевыборка

В качестве синонима масштабирования используют термин перевыборка (*resampling, upsampling, downsampling*). Перевыборка подразумевает цифро-аналоговое преобразование исходного сигнала и новое аналого-цифровое преобразование (взятие отсчетов – *sampling*) с другой частотой.



При уменьшении исходное изображение целесообразно предварительно подвергнуть низкочастотной фильтрации.

Поворот (1)

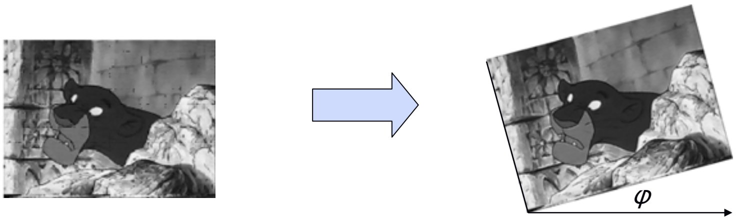
Поворот на угол φ против часовой стрелки вокруг начала координат выполняют с помощью следующей матрицы преобразования:

Прямая

$$\begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Обратная

$$\begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Для повышения скорости обработки поворот на углы кратные 90° реализуют как частные случаи.

Поворот (2)

Часто выполняют поворот вокруг центра изображения (x_c, y_c). В этом случае прямая матрица преобразования есть результат произведения матрицы переноса начала координат в центр изображения, матрицы поворота и матрицы переноса начала координат в начальную позицию:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_c & -y_c & 1 \end{bmatrix} \times \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_c & y_c & 1 \end{bmatrix}$$

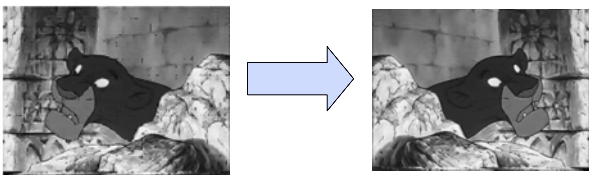
Иногда требуется выполнить поворот на произвольный угол в том же буфере памяти. Такие алгоритмы называют *in place* или *in situ*. В этом случае можно заменить 1 поворот на 2 последовательных сдвига (*shear, skew*). Например, возможна такая декомпозиция обратной матрицы поворота:

$$\begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\varphi & 0 & 0 \\ \sin\varphi & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & -\tan\varphi & 0 \\ 0 & \sec\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

9

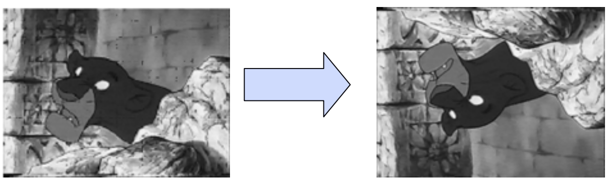
Зеркальное отражение

Зеркальное отражение (*flip*) относительно вертикальной оси:



$$T = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Зеркальное отражение (*flip*) относительно горизонтальной оси:



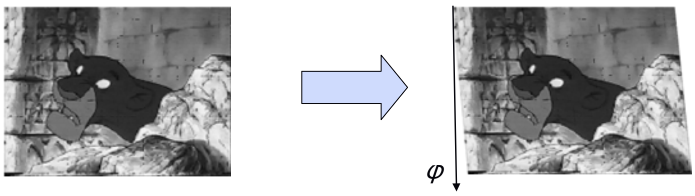
$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Обычно реализуют как частные случаи.

10

Скос

Скос (*skew, shear*) по *x*:

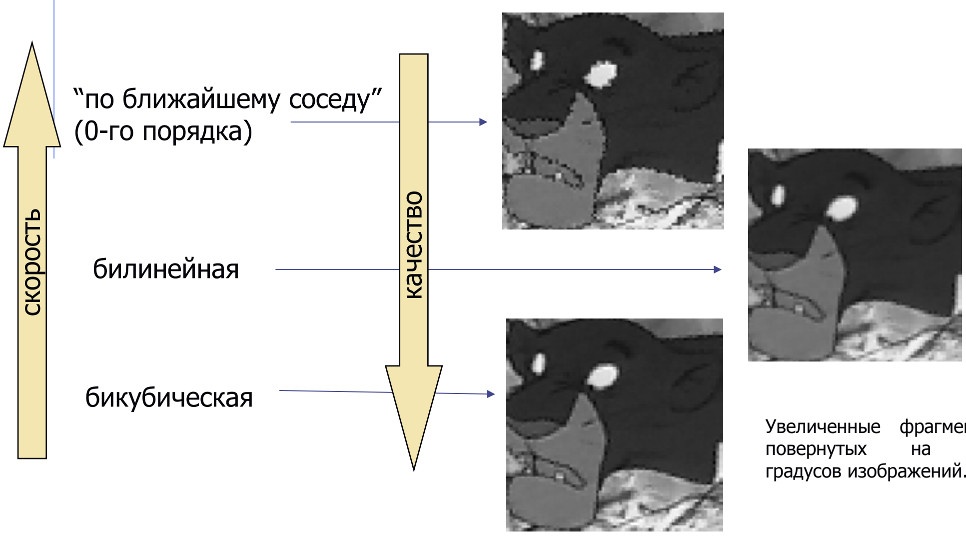


$$T = \begin{bmatrix} 1 & 0 & 0 \\ tg\varphi & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Скос по *y* получается транспонированием матрицы

Интерполяция

Существует много способов интерполяции изображений. Наиболее распространенными и известными являются:



Увеличенные фрагменты повернутых на 20 градусов изображений.

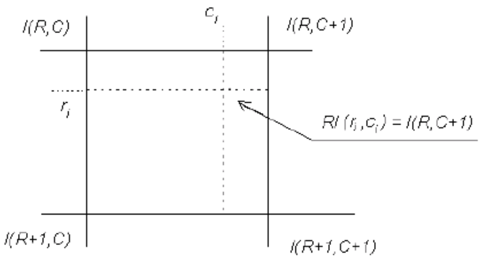
Интерполяция "по ближайшему соседу"

В интерполяции "по ближайшему соседу" (*nearest neighbor*) результатом является значение ближайшего пиксела:

$$RI_{r_i,c_i} = I(R_i, C_i)$$

$R_i = INT(r_i + 0.5)$ где *INT(x)* – возвращает целую часть

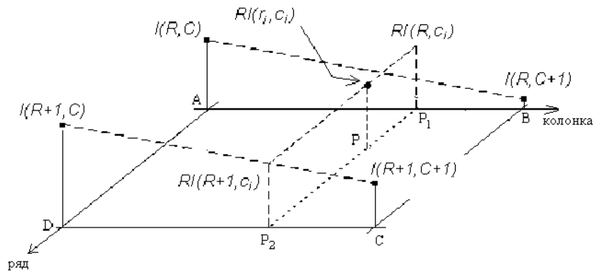
$C_i = INT(c_i + 0.5)$



13

Билинейная интерполяция

Билинейная (*bilinear*) – это интерполяция по билинейной поверхности, построенной по значениям 4-х ближайших пикселов.
Может быть вычислена как 3 интерполяции 1-го порядка:



$$RI(R, c_i) = I(R, C) + (I(R, C+1) - I(R, C)) \times (c_i - C)$$

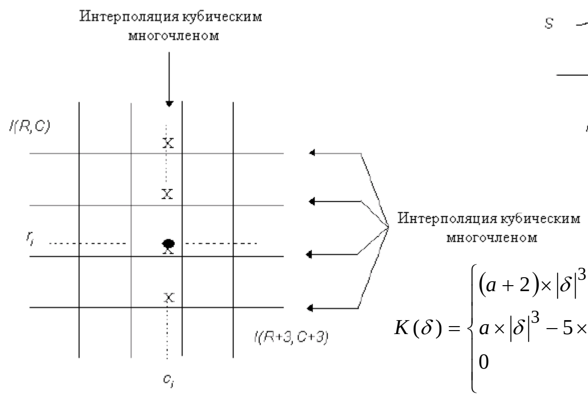
$$RI(R+1, c_i) = I(R+1, C) + (I(R+1, C+1) - I(R+1, C)) \times (c_i - C)$$

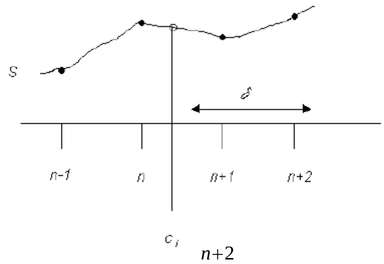
$$RI(r_i, c_i) = RI(R, c_i) + (RI(R+1, c_i) - RI(R, c_i)) \times (r_i - R)$$

14

Бикубическая интерполяция

Бикубическая (*bicubic)* интерполяция или кубическая свертка (*cubic convolution*) состоит из пяти одномерных кубических интерполяций, четыре из которых делаются по рядам вокруг интерполируемой точки, а пятый делается по четырем значениям интерполированных рядов:





$$RI(c_i) = \sum_{j=n-1}^{n+2} S(j) \times K(c_i - j)$$

$$K(\delta) = \begin{cases} (a+2)\times|\delta|^3 - (a+3)\times|\delta|^2 + 1 & 0 \le |\delta| \le 1 \\ a\times|\delta|^3 - 5\times|\delta|^2 + 8a\times|\delta| - 4a & 1 \le |\delta| \le 2 \\ 0 & |\delta| \ge 2 \end{cases}$$

Параметр a может быть равен -1 или -0.5

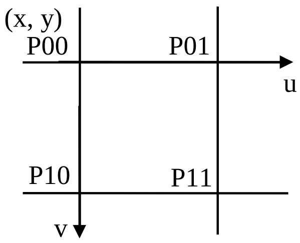
Пример масштабирования (1)

Пример масштабирования изображения 8 bpp, строки изображений не выровнены. Используется билинейная интерполяция и вычисления в арифметике с фиксированной точкой (22:10).

```
void ResizeBilinearFixedPoint( const unsigned char* pbIn, int lWidthIn,
        int lHeightIn, unsigned char *pbOut, int lWidthOut, int lHeightOut )
{
const int fxONE = 0x0400,          // Единица в формате с фиксированной точкой
    fxFRACTIONAL_NUM = 10,         // Количество бит в дробной части
    fxFRACTIONAL_MASK = 0x03FF;    // Маска дробной части чисел с фиксированной т.
    unsigned char* pbRawOut = pbOut;
    for( int i = 0; i < lHeightOut; ++i )
    {
        int yy = (int)(((long long)i<<fxFRACTIONAL_NUM)*lHeightIn/lHeightOut),
            y = yy >> fxFRACTIONAL_NUM,      // целая часть y
            u = yy & fxFRACTIONAL_MASK;      // дробная часть y
        for( int j = 0; j < lWidthOut; ++j )
        {
            int xx = (int)(((long long)j<<fxFRACTIONAL_NUM)*lWidthIn/lWidthOut),
                x = xx >> fxFRACTIONAL_NUM, // целая часть x
                v = xx & fxFRACTIONAL_MASK; // дробная часть x
```

Пример масштабирования (2)

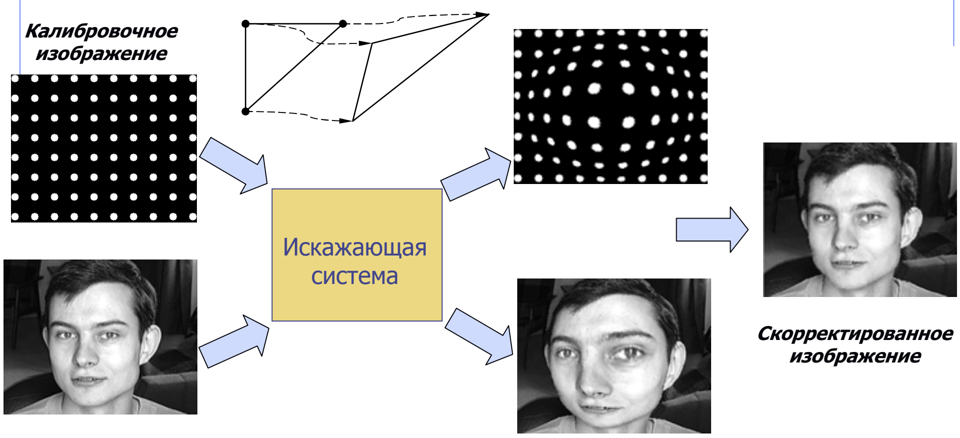
```
const unsigned char* pbRawIn = pbIn + y*lWidthIn + x; // значения соседних пикселов
            int lP00 = *pbRawIn,
                lP01 = *(pbRawIn + 1),
                lP10 = *(pbRawIn + lWidthIn),
                lP11 = *(pbRawIn + lWidthIn + 1);
            // билинейная интерполяция
            *pbRawOut = (unsigned char)( ((fxONE - u)*(fxONE - v)*lP00 +
                u*(fxONE - v)*lP10 + v*(fxONE - u)*lP01 +
                u*v*lP11)>>(2 * fxFRACTIONAL_NUM) );
            ++pbRawOut;
        }
    }
}
```



17

Коррекция дисторсии (1)

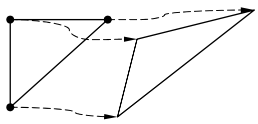
Изображения подвергнутые геометрическим искажениям (дисторсии) могут быть скорректированы на основе данных калибровки.



18

Коррекция дисторсии (2)

Существует взаимно однозначное соответствие между координатами треугольников исходного и искаженного калибровочного изображений, что позволяет для каждого треугольника определить матрицу трансформации путем решения системы линейных уравнений.



$$\begin{bmatrix} x'_i & y'_i & 1 \end{bmatrix} = \begin{bmatrix} x_i & y_i & 1 \end{bmatrix} \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}, \quad i = 0,1,2.$$

$$\begin{cases}
a = \dfrac{-x'_1 \cdot y_0 + x'_1 \cdot y_2 + y_1 \cdot x'_0 - y_1 \cdot x'_2 + x'_2 \cdot y_0 - y_2 \cdot x'_0}{x_0 \cdot y_1 - x_0 \cdot y_2 - y_0 \cdot x_1 + y_0 \cdot x_2 - x_2 \cdot y_1 + y_2 \cdot x_1} \\
b = \dfrac{y'_2 \cdot y_0 - y_2 \cdot y'_0 - y'_1 \cdot y_0 + y'_1 \cdot y_2 + y_1 \cdot y'_0 - y_1 \cdot y'_2}{x_0 \cdot y_1 - x_0 \cdot y_2 - y_0 \cdot x_1 + y_0 \cdot x_2 - x_2 \cdot y_1 + y_2 \cdot x_1} \\
c = \dfrac{x'_0 \cdot x_2 - x'_0 \cdot x_1 + x_0 \cdot x'_1 - x_0 \cdot x'_2 + x'_2 \cdot x_1 - x_2 \cdot x'_1}{x_0 \cdot y_1 - x_0 \cdot y_2 - y_0 \cdot x_1 + y_0 \cdot x_2 - x_2 \cdot y_1 + y_2 \cdot x_1} \\
d = \dfrac{y'_0 \cdot x_2 - y'_0 \cdot x_1 - x_0 \cdot y'_2 + x_0 \cdot y'_1 - y'_1 \cdot x_2 + x_1 \cdot y'_2}{x_0 \cdot y_1 - x_0 \cdot y_2 - y_0 \cdot x_1 + y_0 \cdot x_2 - x_2 \cdot y_1 + y_2 \cdot x_1} \\
t_x = \dfrac{y_1 \cdot x_0 \cdot x'_2 - x'_2 \cdot y_0 \cdot x_1 + x'_1 \cdot y_0 \cdot x_2 - y_1 \cdot x'_0 \cdot x_2 + y_2 \cdot x'_0 \cdot x_1 - x'_1 \cdot x_0 \cdot y_2}{x_0 \cdot y_1 - x_0 \cdot y_2 - y_0 \cdot x_1 + y_0 \cdot x_2 - x_2 \cdot y_1 + y_2 \cdot x_1} \\
t_y = \dfrac{-y_2 \cdot x_0 \cdot y'_1 + y'_1 \cdot y_0 \cdot x_2 - y'_2 \cdot y_0 \cdot x_1 + y_2 \cdot y'_0 \cdot x_1 - y_1 \cdot y'_0 \cdot x_2 + y'_2 \cdot x_0 \cdot y_1}{x_0 \cdot y_1 - x_0 \cdot y_2 - y_0 \cdot x_1 + y_0 \cdot x_2 - x_2 \cdot y_1 + y_2 \cdot x_1}
\end{cases}$$

19