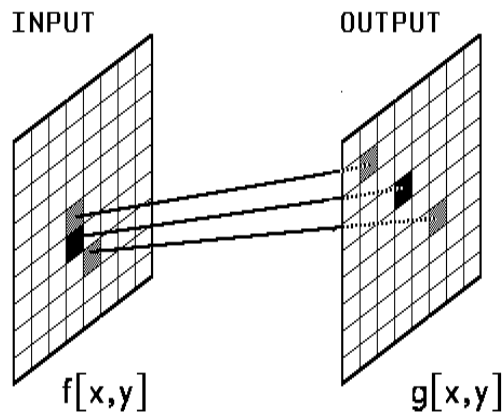


Chapter 18

Geometric Operations

To this point, the image processing operations have computed the gray value (digital count) of the output image pixel based on the gray values of one or more input pixels; in other words, the operation “changed” the gray value to something new. Geometrical image processing operations are fundamentally different; instead of modifying the gray values of pixels, they redefine the pixel locations without changing their values (except to interpolate the values to the new pixel grid). In other words, geometrical operations change the spatial relationships between image pixels to correct distortions due to recording geometry, scale changes, rotations, perspective (keystoning), or due to curved or irregular object surfaces. In this section, we will define the procedures for specifying and implementing a range of geometrical operations.



In theory, it is possible (though exhausting!) to describe geometric transformations via a lookup table of input/output coordinates, i.e., the table would specify new output coordinates $[x', y']$ for each input location $[x, y]$. Equivalently, the coordinate lookup table could specify the input

coordinates $[x, y]$ that map to a specific output pixel $[x', y']$. For an $N \times M$ image, such a lookup table would contain $N \cdot M$ ordered pairs (\implies 262,144 pairs for a

512 × 512 image). The lookup table can specify any arbitrary geometric transform, i.e., input pixels in the same neighborhood may move to locations that are very far apart in the output image. Besides using large blocks of computer memory, coordinate lookup tables are more general than usually necessary. In realistic applications, neighboring pixels in the input image will usually remain in close proximity in the output image, i.e., their coordinates will be transformed in similar manner. Such coordinate mappings are also called rubber-sheet transformations or image warping and may be specified by a set of parametric equations that specify the output coordinates for a given input position:

$$\begin{aligned}x' &= \alpha [x, y] \\y' &= \beta [x, y] \\ \implies \mathcal{O} \{f [x, y]\} &= f [x', y'] = f [\alpha [x, y], \beta [x, y]].\end{aligned}$$

The gray level f at the location $[x, y]$ is transferred to the new coordinates $[x', y']$ to create the output image $f[x', y']$. This sequence of operations equivalent to defining a “new” image $g[x, y]$ in the same coordinates $[x, y]$ but with different gray levels; hence:

$$g [x, y] = f [\alpha [x, y], \beta [x, y]].$$

To preserve the arrangement of pixel gray values in neighborhoods, the transformation of the continuous coordinates should be continuous, i.e., the derivative must be finite everywhere. A power series of the input coordinates for positive powers satisfies this requirement:

$$\begin{aligned}x' &= \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} a_{nm} x^n y^m = a_{00} + a_{10}x + a_{01}y + a_{11}xy + a_{20}x^2 + \dots \\y' &= \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} b_{nm} x^n y^m = b_{00} + b_{10}x + b_{01}y + b_{11}xy + b_{20}x^2 + \dots\end{aligned}$$

In practice, the infinite series is truncated, thus limiting the range of possible transformations. Under many conditions, only four terms are necessary in each expression, i.e., a_{nm} and $b_{nm} = 0$ for $n, m \geq 2$:

$$\begin{aligned}x' &= a_{00} + a_{10}x + a_{01}y + a_{11}xy \\y' &= b_{00} + b_{10}x + b_{01}y + b_{11}xy\end{aligned}$$

Such a transformation is called *bilinear*. There are eight unknown coefficients in the transformation, and thus eight independent equations are needed to find a solution. Knowledge of the coordinate transformation of the vertices of a quadrilateral is sufficient to find a solution for this transformation. In other words, knowledge of the

action on four locations

$$\begin{aligned} [x_1, y_1] &\rightarrow [x'_1, y'_1] \\ [x_2, y_2] &\rightarrow [x'_2, y'_2] \\ [x_3, y_3] &\rightarrow [x'_3, y'_3] \\ [x_4, y_4] &\rightarrow [x'_4, y'_4] \end{aligned}$$

will allow calculation of the eight coefficients. The problem may be cast in matrix notation where the known inputs $[x_i, y_i]$ and outputs $[x'_i, y'_i]$ are arranged as column vectors in the matrices $\underline{\mathbf{X}}$ and $\underline{\mathbf{X}'}$, respectively, and the unknown coefficients a_{ij} and b_{ij} are the rows of the matrix $\underline{\mathbf{A}}$ in the expression:

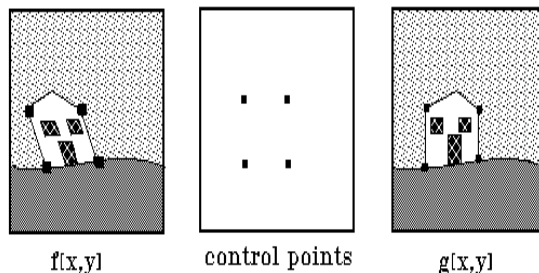
$$\underline{\mathbf{A}} \bullet \underline{\mathbf{X}} = \underline{\mathbf{X}'}$$

$$\begin{bmatrix} a_{00} & a_{10} & a_{01} & a_{11} \\ b_{00} & b_{10} & b_{01} & b_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ x_1y_1 & x_2y_2 & x_3y_3 & x_4y_4 \end{bmatrix} = \begin{bmatrix} x'_1 & x'_2 & x'_3 & x'_4 \\ y'_1 & y'_2 & y'_3 & y'_4 \end{bmatrix}$$

If $\underline{\mathbf{X}}$ is square (and if no three of the known points lie on the same straight line), then the inverse matrix $\underline{\mathbf{X}}^{-1}$ exists; the coefficients a_{ij} and b_{ij} may be found via a straightforward calculation:

$$\begin{aligned} (\underline{\mathbf{A}} \bullet \underline{\mathbf{X}}) \underline{\mathbf{X}}^{-1} &= \underline{\mathbf{X}'} \bullet \underline{\mathbf{X}}^{-1} \\ \implies \underline{\mathbf{A}} &= \underline{\mathbf{X}'} \bullet \underline{\mathbf{X}}^{-1} \end{aligned}$$

The four known vertices in the input and output images are sometimes called “control points”. More control points may be used in a bilinear fit, thus making the problem “overdetermined” (more equations than unknowns). A unique solution of an overdetermined problem may not exist if there is uncertainty (“noise”) in the data. Under such conditions, either a least-squares solution is generated or the control points are applied locally to find local transformations for different sections of the image. If the distortion cannot be adequately represented by a power series with eight coefficients, then more than four control points are required.



18.1 Least-Squares Solution for Warping

The procedure for computing the least-squares solution for the coefficients of a geometric transformation is quite easy in matrix notation. For example, consider a the geometric transformation which adds a constant coordinate translation and magnifications in the orthogonal directions. The coordinate equations are:

$$\begin{aligned}x' &= a_{00} + a_{10}x + a_{01}y \\y' &= b_{00} + b_{10}x + b_{01}y\end{aligned}$$

where the coefficients a_{ij} and b_{ij} must be calculated. The system of equations has six unknown quantities and so requires six equations to obtain a solution. Since each control point (input-output coordinate pair) yields equations for both x and y , three control points are needed. If more control points (say five) are available and consistent, the extras may be ignored and the the matrix inverse computed as before. If the positions of the control points are uncertain, the equations will be inconsistent and the matrix inverse will not exist. Under these conditions, the additional control points will improve the estimate of the transformation. The computation of the coefficients which minimizes the squared error in the transformation is called the least-squares solution, and is easily computed using matrix algebra as a pseudoinverse.

If we have five known control point pairs, the matrix transformation is composed of the 2 row by 3 column matrix $\underline{\mathbf{A}}$, the 3 row by 5 column matrix $\underline{\mathbf{X}}$ and the 2 row by 5 column matrix $\underline{\mathbf{X}'}$:

$$\underline{\mathbf{A}} \bullet \underline{\mathbf{X}} = \underline{\mathbf{X}'}$$

$$\begin{bmatrix} a_{00} & a_{10} & a_{01} \\ b_{00} & b_{10} & b_{01} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 \\ y_1 & y_2 & y_3 & y_4 & y_5 \end{bmatrix} = \begin{bmatrix} x'_1 & x'_2 & x'_3 & x'_4 & x'_5 \\ y'_1 & y'_2 & y'_3 & y'_4 & y'_5 \end{bmatrix}$$

If $\underline{\mathbf{X}}$ were square (and invertible), we would compute $\underline{\mathbf{X}}^{-1}$ to find $\underline{\mathbf{A}}$ via:

$$\underline{\mathbf{A}} = \underline{\mathbf{X}'} \bullet \underline{\mathbf{X}}^{-1}$$

However, $\underline{\mathbf{X}}$ is NOT square in this case, and thus its inverse $\underline{\mathbf{X}}^{-1}$ does not exist. We may evaluate a *pseudoinverse* of $\underline{\mathbf{X}}$ that implements a least-squares solution for $\underline{\mathbf{A}}$ via the following steps for the the general case where $\underline{\mathbf{X}}$ has p rows and q columns ($p = 3$ and $q = 5$ in the example above).

1. multiply both sides of the equation from the right by the *transpose* matrix $\underline{\mathbf{X}}^T$, which is obtained from $\underline{\mathbf{X}}$ by exchanging the rows and columns to obtain a matrix with q rows and p columns. The result is:

$$(\underline{\mathbf{A}} \bullet \underline{\mathbf{X}}) \bullet \underline{\mathbf{X}}^T = \underline{\mathbf{X}'} \bullet \underline{\mathbf{X}}^T$$

2. The associativity of vector multiplication allows the second and third matrices on the left-hand side to be multiplied first:

$$(\underline{\mathbf{A}} \bullet \underline{\mathbf{X}}) \bullet \underline{\mathbf{X}}^T = \underline{\mathbf{A}} \bullet (\underline{\mathbf{X}} \bullet \underline{\mathbf{X}}^T)$$

3. The matrix $\underline{\mathbf{X}} \bullet \underline{\mathbf{X}}^T$ is $p \times p$ square. In the example above, the product of the two matrices is 3×3 square:

$$\underline{\mathbf{X}} \bullet \underline{\mathbf{X}}^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 \\ y_1 & y_2 & y_3 & y_4 & y_5 \end{bmatrix} \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \\ 1 & x_4 & y_4 \\ 1 & x_5 & y_5 \end{bmatrix}$$

$$= \begin{bmatrix} 5 & x_1 + x_2 + x_3 + x_4 + x_5 & y_1 + y_2 + y_3 + y_4 + y_5 \\ x_1 + x_2 + x_3 + x_4 + x_5 & x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 & x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4 + x_5y_5 \\ y_1 + y_2 + y_3 + y_4 + y_5 & x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4 + x_5y_5 & y_1^2 + y_2^2 + y_3^2 + y_4^2 + y_5^2 \end{bmatrix}$$

Since $\underline{\mathbf{X}} \bullet \underline{\mathbf{X}}^T$ is square, there is some chance that its inverse $(\underline{\mathbf{X}} \bullet \underline{\mathbf{X}}^T)^{-1}$ exists. If so, then we can multiply the left-hand side from the right by this inverse; the result is the desired matrix of coefficients $\underline{\mathbf{A}}$:

$$\underline{\mathbf{A}} \bullet (\underline{\mathbf{X}} \bullet \underline{\mathbf{X}}^T) \bullet (\underline{\mathbf{X}} \bullet \underline{\mathbf{X}}^T)^{-1} = \underline{\mathbf{A}}$$

4. If we perform the same series of steps on the right-hand side, we obtain the desired formula:

$$\begin{aligned} \underline{\mathbf{A}} &= (\underline{\mathbf{X}}' \bullet \underline{\mathbf{X}}^T) \bullet (\underline{\mathbf{X}} \bullet \underline{\mathbf{X}}^T)^{-1} \\ &= \underline{\mathbf{X}}' \bullet \left(\underline{\mathbf{X}}^T \bullet (\underline{\mathbf{X}} \bullet \underline{\mathbf{X}}^T)^{-1} \right) \end{aligned}$$

$$\begin{aligned}
 & \begin{bmatrix} a_{00} & a_{10} & a_{01} & a_{11} \\ b_{00} & b_{10} & b_{01} & b_{11} \end{bmatrix} \\
 & = \begin{bmatrix} x'_1 & x'_2 & x'_3 & x'_4 & x'_5 \\ y'_1 & y'_2 & y'_3 & y'_4 & y'_5 \end{bmatrix} \\
 & \bullet \left(\begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 \\ 1 & x_2 & y_2 & x_2y_2 \\ 1 & x_3 & y_3 & x_3y_3 \\ 1 & x_4 & y_4 & x_4y_4 \\ 1 & x_5 & y_5 & x_5y_5 \end{bmatrix} \bullet \begin{bmatrix} 1 & & & & \\ x_1 & x_2 & x_3 & x_4 & x_5 \\ y_1 & y_2 & y_3 & y_4 & y_5 \\ x_1y_1 & x_2y_2 & x_3y_3 & x_4y_4 & x_5y_5 \end{bmatrix} \bullet \begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 \\ 1 & x_2 & y_2 & x_2y_2 \\ 1 & x_3 & y_3 & x_3y_3 \\ 1 & x_4 & y_4 & x_4y_4 \\ 1 & x_5 & y_5 & x_5y_5 \end{bmatrix}^{-1} \right)
 \end{aligned}$$

18.2 Common Geometrical Operations

$x' = x, y' = y \implies g[x, y] = f[x', y'] = f[x, y] \implies$ identity transformation, no change

$x' = x + x_0, y' = y + y_0 \implies g[x, y] = f[x + a_{00}, y + b_{00}] \implies$ translation by $[a_{00}, b_{00}]$

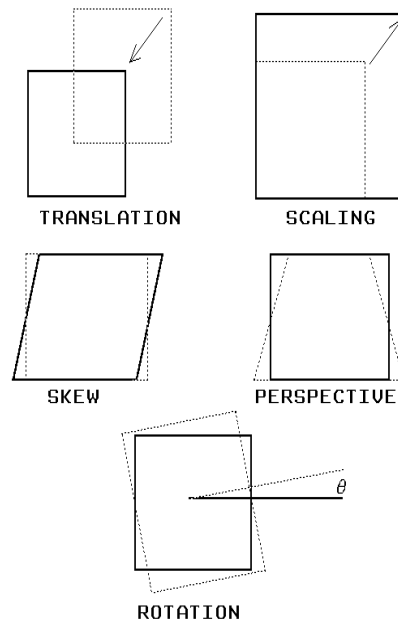
$x' = ax, y' = by \implies g[x, y] = f[a_{10}x, b_{01}y] \implies$ spatial *stretching* or *scaling*

$x' = x + ay, y' = y \implies g[x, y] = f[x + a_{01}y, y] \implies$ *skew*

$x' = x + axy, y' = y \implies g[x, y] = f[x + a_{11}xy, y] \implies$ perspective distortion

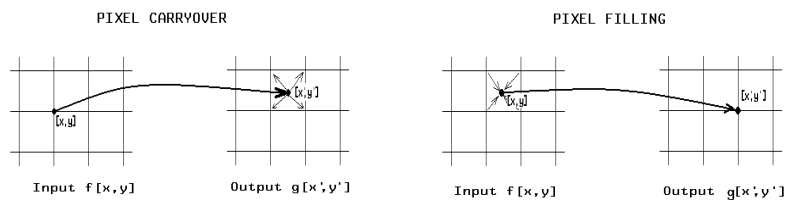
$$\left\{ \begin{array}{l} x' = \alpha[x, y] = x \cos \theta - y \sin \theta \\ y' = \beta[x, y] = x \sin \theta + y \cos \theta \end{array} \right\} \implies \text{rotation thru } \theta$$

$[x', y']$ are the coordinates of the input image that are mapped to $[x, y]$ in the output image.



18.3 Pixel Transfers

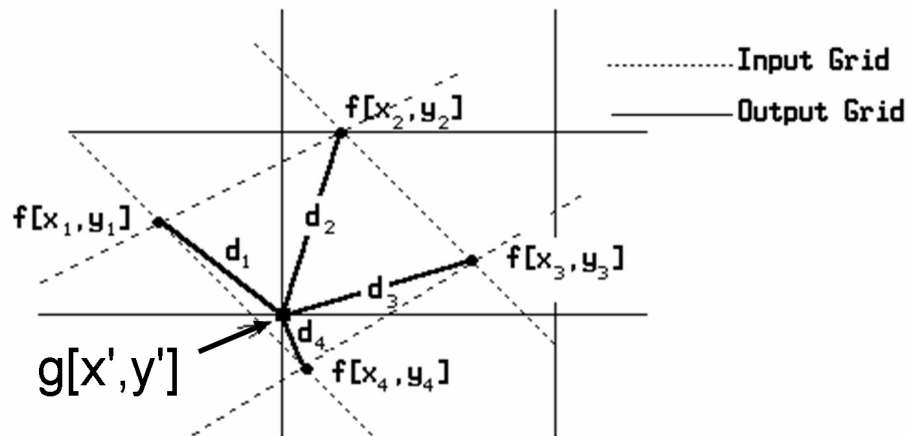
As already mentioned, a geometrical transformation may be implemented by specifying the where each pixel of the input image is located in the output grid, or by specifying the location of each output pixel on the input grid. Except for certain (and usually uninteresting) transformations, pixels of the input image will rarely map exactly to pixels of the output grid. It is therefore necessary to interpolate the gray value from pixels with noninteger coordinates (i.e., nongrid points) to pixels with integer coordinates that lie upon the grid. The method that transfers the input pixel coordinates and interpolates the gray value on the output grid is called *pixel carryover* by Castleman. It may also be called an *input-to-output* mapping. The algorithm that locates the output pixel upon the input grid and interpolates the gray value of the input pixels is called *pixel filling* or an *output-to-input* mapping



18.4 Pixel Interpolation

Gray-value interpolation is based on the relative distances of the nearest neighbors to or from the geometrically transformed pixel. In the simplest case (nearest-neighbor or zeroth-order interpolation), all of the gray value is transferred to or from the nearest pixel. However, since the distances of the four neighbors must be evaluated to determine which is closest, it generally is quite easy to “upgrade” nearest-neighbor calculations to *bilinear* interpolation. This method divides the gray level of the non-integer pixel among its four nearest “integer” neighbors in inverse proportion to the distance; if the transferred pixel is equidistant from the four neighbors, then its gray value is divided equally, if it is much closer to one of the four grid points, most of its gray value is transferred to that pixel. For pixel filling, the gray value at the output pixel $g[x', y']$ is determined from the following equation, where x_n, y_n are the coordinates of the nearest pixel of the input grid to the transformed pixel, and d_n are the respective distances:

$$g[x', y'] = \frac{\frac{1}{d_1} f_1 [x_1, y_1] + \frac{1}{d_2} f_2 [x_2, y_2] + \frac{1}{d_3} f_3 [x_3, y_3] + \frac{1}{d_4} f_4 [x_4, y_4]}{\frac{1}{d_1} + \frac{1}{d_2} + \frac{1}{d_3} + \frac{1}{d_4}}$$



Schematic of interpolation of pixel gray value. The input and output grids of pixels are shown as solid and dashed lines, respectively. The distances of the output pixel from the four nearest neighbors are labeled $d_1 - d_4$.

The gray level of the transformed pixel can be divided among more of the neighboring pixels by using higher-order interpolation, e.g., cubic spline, etc.

18.4.1 Example: Image Rotation

The examples are small (64×64) images rotated by the degree increments specified. The images have been interpolated using the output-to-input mapping and bilinear interpolation. Note that artifacts are visible, particularly at 45° .

