Volume 5, Issue 1, January 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Study of Geometric Image Operations

Dr. Govind N Sarage
Department of Computer Sceince, National Defence Academy,
Khadakwasla Pune, Maharashtra, India

Abstract*— Geometric operations change the spatial relationships between objects in an image. They do this by moving objects around and changing the size and shape of objects. Geometric operations help rearrange an image so we can see what we want to see a little better. This paper discussed geometric operations. These powerful and exible operations change the relationships, size, and shape of objects in images. They allow you to manipulate images for better display, comparison, etc.

***Keywords*— Include at least 5 keywords or phrases**

I. INTRODUCTION

Geometric image operations[1] are, in a sense, the opposite of point operations[1]: they modify the spatial positions and spatial relationships of pixels, but they do not modify gray level values. A spatial transformation (also known as a geometric operation) modifies the spatial relationship between pixels in an image, mapping pixel locations in an input image to new locations in an output image.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.

Computer graphics[2], however, is primarily concerned with creating images of an unreal world, or at least a visually modified reality, and subsequently geometric distortions are commonly used in that discipline.

A geometric image operation generally requires two steps: First, a spatial mapping of the coordinates of an original image f to define a new image g :

$$g(\mathbf{n}) = f(\mathbf{n'}) = f[\mathbf{a}(\mathbf{n})]. \quad (1)$$

Thus, geometric image operations[1,2] are defined as functions of position rather than intensity. The 2D, two-valued mapping function $\mathbf{a}(\mathbf{n}) = [a1(n1,n2), a2(n1,n2)]$ is usually defined to be continuous and smoothly changing, but the coordinates $\mathbf{a}(\mathbf{n})$ that are delivered are not generally integers. For example, if $\mathbf{a}(\mathbf{n}) = (n1/3, n2/4)$, then $g(\mathbf{n}) = f(n1/3, n2/4)$, which is not defined for most values of $(n1,n2)$. The question then is which value of f are used to define $g(\mathbf{n})$, when the mapping does not fall on the standard discrete lattice?

This implies the need for the second operation: *interpolation* of non integer coordinates $a1(n1,n2)$ and $a2(n1,n2)$ to integer values, so that g can be expressed in a standard row-column format. There are many possible approaches for accomplishing interpolation; we will look at two of the simplest: *nearest neighbor interpolation* and *bilinear interpolation[3]*. The first of these is too simplistic for many tasks, while the second is effective for most.

II. INTERPOLATION METHOD

Interpolation[3] is the process used to estimate an image value at a location in between image pixels. When imresize(matlab function) enlarges an image, the output image contains more pixels than the original image. The imresize(function uses interpolation to determine the values for the additional pixels. Interpolation methods determine the value for an interpolated pixel by finding the point in the input image that corresponds to a pixel in the output image and then calculating the value of the output pixel by computing a weighted average of some set of pixels in the vicinity of the point. The weightings are based on the distance each pixel is from the point. By default, imresize uses bicubic interpolation to determine the values of pixels in the output image, but you can specify other interpolation methods and interpolation kernels. In the following example, imresize uses the bilinear interpolation method. See the imresize reference page for a complete list of interpolation methods and interpolation kernels available. You can also specify your own custom interpolation kernel[3,4].

A. Nearest Neighbour Interpolation

Here, the geometrically transformed coordinates are mapped to the nearest integer coordinates of f :

$$g(\mathbf{n}) = f\{\mathrm{INT}[a1(n1,n2) + 0.5], \mathrm{INT}[a2(n1,n2) + 0.5]\}, \quad (2)$$

where INT[R] denotes the nearest integer that is less than or equal to R. Hence, the coordinates are *rounded* prior to assigning them to g. This certainly solves the problem of finding integer coordinates of the input image, but it is quite simplistic, and, in practice, it may deliver less than impressive results. For example, several coordinates to be mapped

may round to the same values, creating a block of pixels in the output image of the same value. This may give an impression of "blocking," or of structure that is not physically meaningful. The effect is particularly noticeable along sudden changes in intensity, or "edges," which may appear jagged following nearest neighbor interpolation.

B. Bilinear Interpolation

Bilinear interpolation[3] produces a smoother interpolation than does the nearest neighbour approach. Given four neighboring image coordinates $f(n10,n20)$, $f(n11,n21)$, $f(n12,n22)$, and $f(n13,n23)$ (these can be the four nearest neighbors of $f[\mathbf{a}(\mathbf{n})]$), then the geometrically transformed image $g(n1,n2)$ is computed as

$$g(n1,n2) = A0 + A1n1 + A2n2 + A3n1n2, \tag{3}$$

which is a bilinear function in the coordinates $(n1,n2)$. The bilinear weights $A0, A1, A2$, and $A3$ are found by solving

$$\begin{bmatrix} A0 \\ A1 \\ A2 \\ A3 \end{bmatrix} = \begin{bmatrix} 1 & n10 & n20 & n10n20 \\ 1 & n11 & n21 & n11n21 \\ 1 & n12 & n22 & n12n22 \\ 1 & n13 & n23 & n13n23 \end{bmatrix}^{-1} + \begin{bmatrix} f(n10,n20 \\ f(n11,n21 \\ f(n12,n22 \\ f(n13,n23 \end{bmatrix} \tag{4}$$

Thus, $g(n1,n2)$ is defined to be a linear combination of the gray levels of its four nearest neighbors. The linear combination defined by (4) is in fact the value assigned to $g(n1,n2)$ when the best (least squares) planar fit is made to these four neighbors. This process of optimal averaging produces a visually smoother result. Regardless of the interpolation approach that is used, it is possible that the mapping coordinates $a1(n1,n2)$, $a2(n1,n2)$ do not fall within the pixel ranges

$$0 \leq a1(n1,n2) \leq M - 1$$
$$\text{and/or} \tag{5}$$
$$0 \geq a2(n1,n2) \geq N - 1,$$

in which case it is not possible to define the geometrically transformed image at these coordinates. Usually a nominal value is assigned, such as $g(\mathbf{n}) = 0$, at these locations.

III. IMAGE TRANSLATION

The most basic geometric transformation is the *image translation[1,5]*, where $(b1,b2)$ are integer constants. In this case $g(n1,n2) = f(n1-b1,n2-b2)$, which is a simple shift or translation of g by an amount $b1$ in the vertical (row) direction and an amount $b2$ in the horizontal direction. This operation is used in image display systems, when it is desired to move an image about, and it is also used in algorithms, such as image convolution, where images are shifted relative to a reference. Since integer shifts can be defined in either direction, there is no need for the interpolation step.

IV. IMAGE ROTATION

Rotation of the image g by an angle θ relative to the horizontal $(n1)$ axis is accomplished by the following transformations:

$$a1(n1,n2) = n1\cos\theta - n2\sin\theta$$
$$\text{and} \tag{6}$$
$$a2(n1,n2) = n1\sin\theta + n2\cos\theta.$$

The simplest cases are $\theta = 90^0$, where $[a1(n1,n2),a2(n1,n2)] = (-n2,n1)$; $\theta = 180^0$, where $[a1(n1,n2),a2(n1,n2)] = (-n1,-n2)$; and $\theta = 90^0$, where $[a1(n1,n2),a2(n1,n2)] = (n2, -n1)$. Since the rotation point is not defined here as the center of the image, the arguments (6) may fall outside of the image domain. This may be ameliorated by applying an image translation either before or after the rotation to obtain coordinate values in the nominal range.

A. Rotating an Image.

To rotate an image, use the *imrotate* function. When you rotate an image, you specify the image to be rotated and the rotation angle, in degrees. If you specify a positive rotation angle, imrotate rotates the image counterclockwise; if you specify a negative rotation angle, imrotate rotates the image clockwise. By default, imrotate creates an output image large enough to include the entire original image. Pixels that fall outside the boundaries of the original image are set to 0 and appear as a black background in the output image. You can, however, specify that the output image be the same size as the input image, using the 'crop' argument. Similarly, imrotate uses nearest-neighbor interpolation by default to determine the value of pixels in the output image, but you can specify other interpolation methods. See the imrotate reference page for a list of supported interpolation methods.

This example rotates an image 35° counterclockwise and specifies bilinear interpolation.

```
I = imread('circuit.tif');
J = imrotate(I,35,'bilinear');
imshow(I)
figure, imshow(J)
```


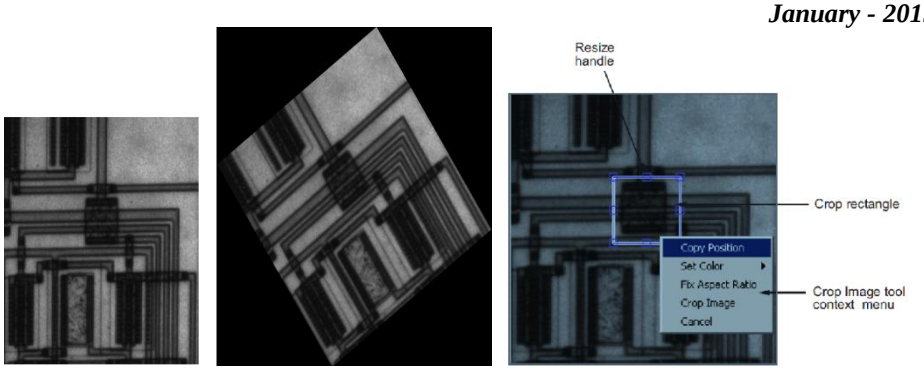

Fig. 1. a) Circuit image b) rotates an image 35° c) crop image

B. Cropping an Image.

To extract a rectangular portion of an image, use the imcrop function. Using imcrop, you can specify the crop region interactively using the mouse or programmatically by specifying the size and position of the crop region. This example illustrates an interactive syntax. The example reads an image into the MATLAB workspace and calls imcrop specifying the image as an argument. imcrop displays the image in a figure window and waits for you to draw the crop rectangle on the image. When you move the pointer over the image, the shape of the pointer changes to cross hairs . Click and drag the pointer to specify the size and position of the crop rectangle. You can move and adjust the size of the crop rectangle using the mouse. When you are satisfied with the crop rectangle, double-click to perform the crop operation, or right-click inside the crop rectangle and select **Crop Image** from the context menu. imcrop returns the cropped image in J.

C. ***Image Zoom***

The *image zoom* either magnifies or minifies the input image according to the mapping functions

$a1(n1,n2) = n1/c$ and $a2(n1,n2) = n2/d$, (7)

where $c \geq 1$ and $d \geq 1$ to achieve magnification[6], and $c < 1$ and $d < 1$ to achieve minification. If applied to the entire image, then the image size is also changed by a factor $c(d)$ along the vertical (horizontal) direction. If only a small part of an image is to be zoomed, then a translation may be made to the corner of that region, the zoom applied, and then the image cropped. The image zoom is a good example of a geometric operation for which the type of interpolation is important, particularly at high magnifications. With nearest neighbor interpolation[1,7] many values in the zoomed image may be assigned the same grayscale, resulting in a severe "blotching" or "blocking" effect. The bilinear interpolation usually supplies a much more viable alternative.

Figure 3.19 depicts a 4x zoom operation applied to the image. The image was first zoomed, creating a much larger image (16 times as many pixels). The image was then translated to a point of interest (selected, e.g., by a mouse), then was cropped to size 256X256 pixels around this point. Both nearest neighbor and bilinear interpolation[7,8] were applied for the purpose of comparison. Both provide a nice "close-up" of the original, making the faces much more identifiable. However, the bilinear result ismuch smoother and does not contain the blocking artifacts[9] that can make recognition of the image difficult.

Fig.2. a) b)

Example of (4x) image zoom followed by interpolation. (a) Nearest-neighbor interpolation;(b) bilinear interpolation.

It is important to understand that image zoom followed by interpolation does not inject *any* new information into the image, although the magnified image may appear easier to see and interpret. The image zoom is only an interpolation of known information.

V. CONCLUSION

Basic geometric operations are somewhat more complex than basic algebraic operations and use less in digital image processing. Generally, these operations can be quite complex and computationally intensive, especially when applied to video sequences. However, the more complex geometric operations are not much used in engineering image processing, although they are heavily used in the computer graphics field. The reason for this is that image processing is primarily concerned with correcting or improving images of the real world, hence complex geometric operations, which distort images, are less frequently used. Bi-linear interpolation removes jagged lines by finding a gray level between pixels. Interpolation finds values between pixels in one direction (interpolating 2/3's of the way between 1 and 10 returns 7). Bi-linear interpolation finds values between pixels in two directions, hence the prefix \ bi."

REFERENCES

[1] R.C. Gonzalez and R.E. Woods, Digital Image Processing, Prentice Hall, Second Edition, 2002.
[2] Roger Bourne ,Fundamentals of Digital Imaging in Medicine springer-verlag london limited 2010.
[3] Geoff Daugherty, Digital Image Processing for Medical Applications, Cambridge University Press 2009.
[4] Atam P Dhawan, H K Huang, Day-Shik Kim.Principles And Advance Method In Medical Imaging And Image Analysis, World Scientific Publishing Co. Pvt. Ltd 2008.
[5] Issac h Bankman Handbook of medical image processing and analysis, Elsevier Inc 2009.
[6] J.C. Russ, The Image Processing Handbook (fifth edition), CRC Press, 2007. S.M. Pizer et al., Adaptive histogram equalization and its variations., Computer Vision, Graphics and Image Processing, September 1987, vol. 39, no. 3, pp 355-368.
[8] V. Caselles, J.L. Lisani, J.M. Morel and G. Sapiro, .Shape preserving local histogram modification., IEEE Trans. Image Processing,1998, vol. 8, no. 2, pp. 220-230.
[9] J.A. Stark, .Adaptive contrast enhancement using generalization of histogram equalization., IEEE Trans. Image Processing, 2000, vol. 9, no. 5, pp. 889-906.

