if the image has motion blur, the edge will be streaked in some direction. The streaking tells us how much motion occurred and in what direction it happened.

In both of these cases, a telltale edge or point of light can be used to create an inverse filter. Alternately, it can indicate which frequencies in the frequency image need to be attenuated or accentuated to reverse the original blurring process.

Geometric Transformation Processing

Geometric transformations provide the ability to reposition pixels within an image. Using a mathematical transformation, pixels are relocated from their (x,y) spatial coordinates in the input image to new coordinates in the output image. Geometric transformations are used to move, spin, size, and arbitrarily contort the geometry of an image. These transformations are used for correcting geometric distortions in an image, as well as for adding visual effects such as a perspective foreshortening.

There are two primary forms of geometric transformation operations. First, there are *linear geometric operations*, which include translation, rotation, and scaling. These operations, sometimes referred to as *affine transformations*, do not introduce any curvature to the processed image, hence the term *linear*. Second, there are *nonlinear geometric operations*, known as *warping transformations*. Warping transformations can introduce localized curvatures as well as overall bends and contortions to the processed image. Geometric transformations can be used to register multiple images and restore various geometric distortions caused by lens aberrations and viewing geometry.

All geometric operations are performed by moving pixel brightnesses from their original spatial coordinates in the input image to new coordinates in the output image. The general equation for these operations is

$$I(x,y) \rightarrow O(x',y')$$

where (x',y') are the transformed coordinates of the pixel brightness originally located at coordinates (x,y). It is implied that every input pixel location is processed through this transformation, creating a geometrically transformed output pixel location. Each geometric operation is therefore defined by a coordinate transformation equation that defines the new x' and y' output coordinates in terms of the input pixel at coordinates (x,y).

The process of transforming pixel locations from the input image to the output image is called *source-to-target mapping*. Each pixel of the input image (source) is transformed, pixel by pixel, to its new location in the output image (target). In practice, though, this transformation method does not work well. As the input pixels are transformed, some output pixel locations may be missed because no input pixels were transformed there. The missed locations will be devoid of any brightness and will appear black. The black holes create a poor resulting output

image. This phenomenon occurs especially when a scaling or warping operation is involved.

By using a reverse transformation, known as *target-to-source mapping*, the transformation mapping can be reversed, avoiding black holes in the output image. The general equation for target-to-source geometric transformations is

$$O(x,y) \leftarrow I(x',y')$$

where (x',y') are the transformed coordinates of the input image pixel brightness to be relocated to coordinates (x,y) in the output image. For target-to-source mapping, it is implied that every output pixel location is processed through this transformation, creating a geometrically transformed input pixel location. With the target-to-source form, each geometric operation is defined by a coordinate transformation equation that defines the x' and y' input coordinates in terms of the output pixel at coordinates (x,y).

This transformation method maps target (output) pixel locations to source (input) pixel locations. By processing each output image pixel location through the transformation, an input pixel location is identified. This input pixel location's brightness is then relocated to the new output pixel location. By stepping through each output pixel location and filling it with its transformed input pixel brightness, it is impossible to miss any output pixel locations. The geometric transformation result is identical to the source-to-target mapping result, except that there are no black holes in the output image. Figure 4.29 shows the source-to-target and target-to-source mappings.

In the following discussions, both the source-to-target and target-to-source transformation equations will be given for the various geometric transformation operations. In practice, we will generally implement the target-to-source transformations to avoid the black hole problem. But because the source-to-target mapping is easier to envision, our examples will use the source-to-target mapping convention.

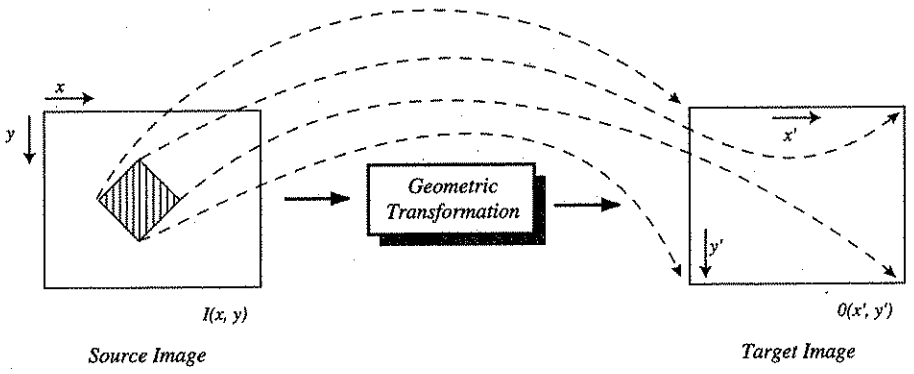

Figure 4.29a *The source-to-target mapping. The transformation sequences, pixel by pixel, through the input pixel locations and places their brightnesses into resulting transformed output pixel locations.*

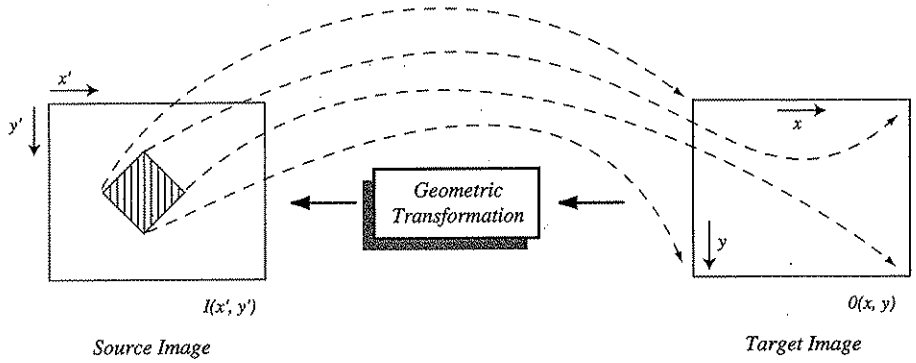

Figure 4.29b *The target-to-source mapping. The transformation sequences, pixel by pixel, through the output pixel locations and fetches transformed input pixel location brightnesses to fill them.*

Translation, Rotation, and Scaling

The linear geometric transformations can spin an image, enlarge and shrink it, and move it left, right, up, and down. These operations cannot introduce curvatures, so straight lines will remain straight in the resulting output image. These transformations are shown in Figure 4.30.

IOS 18 Translation Transformation

Image translation slides an image left, right, up, or down. An x-value defines the amount of left or right slide, while a y-value defines the amount of up or down slide. The coordinate transformation equations for image translation are

Source-to-target mapping

$$x' = x + T_x$$

and

$$y' = y + T_y$$

where x and y comprise the input pixel coordinates, x' and y' are the output coordinates, and T_x and T_y define the amount of translation in the x and y directions.

Target-to-source mapping

$$x' = x - T_x$$

and

$$y' = y - T_y$$

where x and y comprise the output pixel coordinates, x' and y' are the transformed input coordinates, and T_x and T_y define the amount of translation in the x and y directions.

As an example, using the source-to-target equations with translation values of $T_x = -100$ and $T_y = -50$, we get the following transformation equations:

$$x' = x - 100$$

and

$$y' = y - 50$$

The pixel at location (126,68) will be mapped to the location (26,18) in the output image. When applied to all pixels, this transformation shifts the input image to the left by 100 pixels and up by 50 pixels.

In the case where the T_x or T_y translation values are not integers, the transformation will translate the image by some amount that does not fall directly onto a new output pixel location. In this case, we need some sort of pixel interpolation scheme to estimate the resulting pixel brightness at integer pixel locations in the output image. We will discuss various pixel interpolation methods later in this chapter.

Image rotation spins images about a center point. The coordinate transformation equations for image rotation are

Source-to-target mapping

$$x' = x\cos\theta + y\sin\theta$$

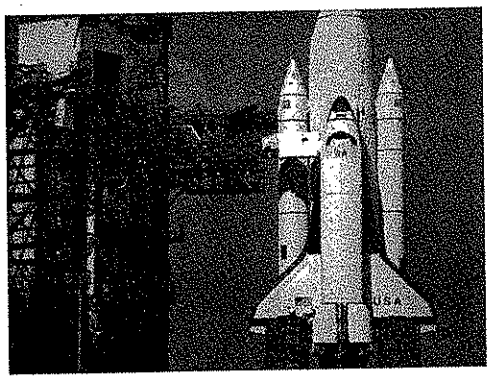

Figure 4.30a *Original Space Shuttle image.*

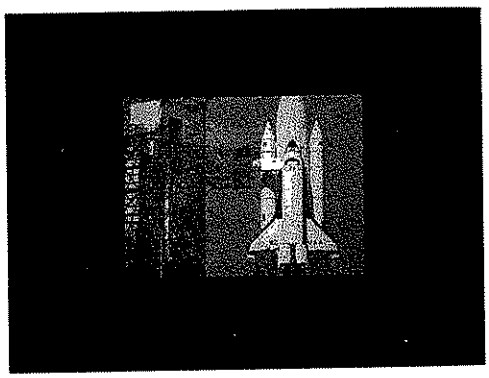

Figure 4.30b *Scaled image where* $S_x = 0.5$ *and* $S_y = 0.5$.

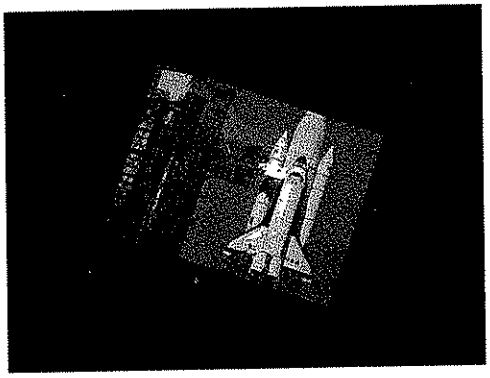

Figure 4.30c *Rotated (and scaled) image where* $\theta = 19°$.

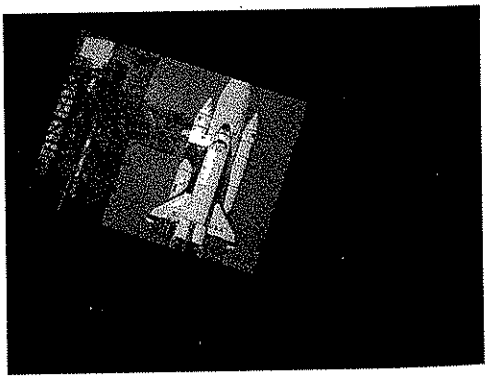

Figure 4.30d *Translated (and scaled and rotated) image where* $T_x = -100$ *and* $T_y = -50$.

and

$$y' = -x\sin\theta + y\cos\theta$$

where x and y comprise the input pixel coordinates, x' and y' are the output coordinates, and T defines the angle of clockwise rotation of the image about the (0,0) pixel location.

Target-to-source mapping

$$x' = x\cos\theta - y\sin\theta$$

and

$$y' = x\sin\theta + y\cos\theta$$

where x and y comprise the output pixel coordinates, x' and y' are the transformed input coordinates, and T defines the angle of clockwise rotation of the image about the (0,0) pixel location. Any rotation angle between 0 and 360 degrees may be specified.

As an example, using the source-to-target equations with the rotation angle of $\theta = 19°$, we get the following transformation equations:

$$\begin{aligned} x' &= x\cos\theta + y\sin\theta \\ &= x\cos(19°) + y\sin(19°) \\ &= x(0.946) + y(0.326) \end{aligned}$$

and

$$\begin{aligned} y' &= -x\sin\theta + y\cos\theta \\ &= -x\sin(19°) + y\cos(19°) \\ &= -x(0.326) + y(0.946) \end{aligned}$$

The pixel at location (126,68) will be transformed to the location (141.4,23.3) in the output image. By applying the rotation transformation to all pixels in the input image, we create an output image that is rotated by 19 degrees.

When rotation angles are selected that are not multiples of 90 degrees, pixels will often get transformed to noninteger pixel locations in the output image, as in the above example. Like the translation transformation, we must use a pixel interpolation scheme to estimate the pixel brightnesses at the integer pixel locations in the output image.

IOS 20 Scaling Transformation

Image scaling enlarges and shrinks an image. An x-value defines the amount of x-direction scaling, while a y-value defines the amount of y-direction scaling. The coordinate transform equations for image scaling are given by the equations

Source-to-target mapping

$$x' = xS_x$$

and

$$y' = yS_y$$

where x and y comprise the input pixel coordinates, x' and y' are the output coordinates, and S_x and S_y define the amount of scaling in the x and y directions.

Target-to-source mapping

$$x' = x/S_x$$

and

$$y' = y/S_y$$

where x and y comprise the output pixel coordinates, x' and y' are the transformed input coordinates, and S_x and S_y define the amount of scaling in the x and y directions.

As an example, using the source-to-target equations with scaling factors of $S_x = 0.5$ and $S_y = 0.5$, we get the following transformation equations:

$$x' = x(0.5)$$

and

$$y' = y(0.5)$$

The pixel at location (126,68) will be transformed to the location (63,34) in the output image. By applying the scaling transformation to all pixels in the input image, we create an output image that is half the size in both the x and y dimensions. Like the other transformations, if the scaling operation transforms input pixels to noninteger output pixel locations, we need a pixel interpolation scheme to estimate pixel brightnesses at integer locations.

Pixel interpolation is especially important for the scaling transformation when the scaling factor is greater than 1. This is because the input image becomes enlarged and pixels are stretched out. Say, for instance, we use a scaling factor of 2 for both S_x and S_y. The pixel at location (126, 68) will be transformed to the location (252,136). Notice, though, that the neighboring input pixel at location (127,68) is transformed to location (254,136), as Figure 4.31 illustrates. The obvious question is, what happens to the pixel at location (253,136) in the output image? In fact, with the scaling factors of 2, every odd pixel and line location in the output image will have nothing transformed to it. Pixel interpolation schemes are used to fill in these pixels with estimates of the brightnesses that should be there.

Further, let's look back to the scaling example where both S_x and S_y are 0.5. In this case, the pixel at location (126,68) is transformed to the location (63,34). The pixel at (127,68) is transformed to (63.5,34). But, because the output image only has integer pixel locations, the pixel is extraneous—there are more pixels in the

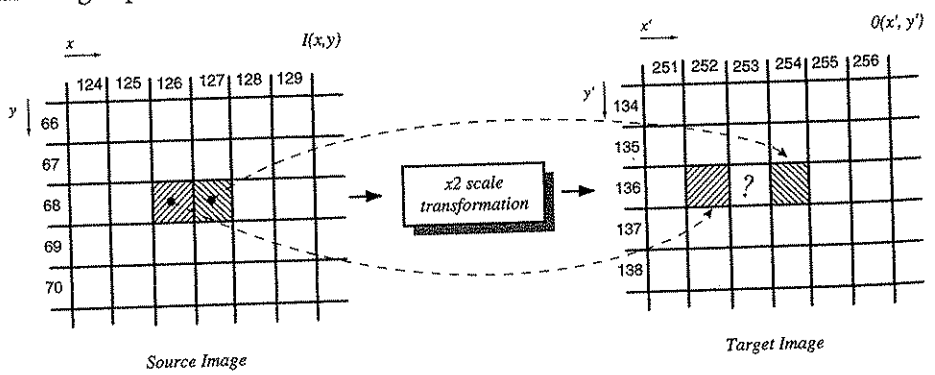

Figure 4.31 *The need for pixel interpolation—the source-to-target mapping skips every other pixel in the output image, leaving distracting black holes.*

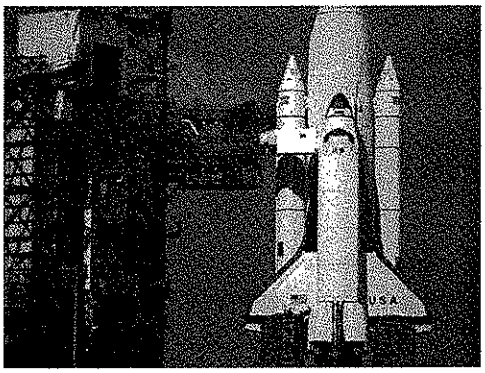

Figure 4.32a *Original Space Shuttle image.*

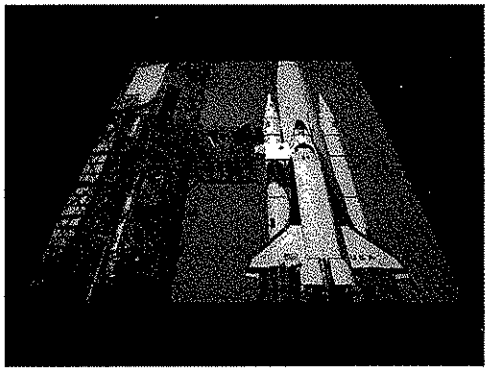

Figure 4.32b *Perspective distortion foreshortens image geometry based on the relative distance of parts of the image from the viewer.*

input image than necessary. The good part is that there is an abundance of input pixels, and therefore pixel interpolation is not of prime importance. The bad part, however, is that spatial aliasing artifacts will appear because the extra pixels must be eliminated. This aliasing effect can be eliminated by using a low-pass filtering operation, as discussed later in the section on downsampling.

Perspective distortions in images caused by the camera-target viewing geometry can be restored by scaling operations. As Figure 4.32 shows, perspective distortion appears as the reduction in scale of an object as it recedes from the foreground of an image to the background. Try holding a book at arm's length and slowly tilting the top away from you. When the book is almost flat, line up the top edge (which is farthest from you) to the bottom edge (which is closest to you). The top edge will appear to be smaller than the bottom edge. Perspective distortion has reduced the scale of the part of the book that is farthest away, relative to that which is closest. By applying two scaling transformations, one that stretches the y axis and one that stretches the x axis with a descending scale factor, we can remove perspective distortion and restore the original flat geometry.

Warping

As mentioned before, linear geometric transformations cannot introduce curvature in their mapping process. Sometimes the ability to introduce curvature is important when an image has been distorted through lens aberrations and other nonlinear processes. *Warping transformations*, often called *rubber sheet transformations*, can arbitrarily stretch and pull the image about defined points, yielding a resulting image that conforms to particular geometric requirements.

Looking back at the source-to-target equations for translation, rotation, and scaling transformations, we can combine them into the following equation:

$$x' = (x\cos\theta + y\sin\theta)S_x + T_x$$
$$= (S_x\cos\theta)x + (S_x\sin\theta)y + T_x$$

which can be generalized to the form

$$x' = a_2x + a_1y + a_0$$

where the coefficients a_2, a_1, and a_0 are constants with the values $S_x\cos\theta$, $S_x\sin\theta$, and T_x, respectively, and

$$y' = (-x\sin\theta + y\cos\theta)S_y + T_y$$
$$= (-S_y\sin\theta)x + (S_y\cos\theta)y + T_y$$

which can be generalized to the form

$$y' = b_2x + b_1y + b_0$$

where the coefficients b_2, b_1, and b_0 are constants with the values $-S_y\sin\theta$, $S_y\cos\theta$, and T_y, respectively.

The source-to-target linear geometric transformations of translation, rotation, and scaling reduce to the following generalized equations, called *polynomials*:

$$x' = a_2x + a_1y + a_0$$

and

$$y' = b_2x + b_1y + b_0$$

Warping transformations are simply these generalized polynomial equations with the addition of higher-order terms, such as x^2, y^2, x^3, y^3, and so on. The warp is said to have an *order*, dependent on the highest exponential term of x or y appearing in the polynomial. A first-order warp means that only the x and y terms exist, which is the linear case of translation, rotation, and scaling. A second-order warp includes the x^2 and/or y^2 terms. A third-order warp includes the x^3 and/or y^3 terms, and so on. The higher the order of a warp, the more complex geometric warping it can offer.

The choice of how high an order warp to perform is totally dependent upon the requirements of the application. The computational time for a warping transformation increases with its order; therefore, performing the minimum order warp is always desired. Also, as in the earlier linear transformations, pixel interpolation is required in higher order transformations, because often input pixel locations will not be transformed to integer output pixel locations.

Controlling the Warp Transformation

Warping transformations can be conveniently discussed using a rubber sheet analogy. We can think of a warping process as operating on an input image printed on a rubber sheet. The rubber sheet can be stretched at arbitrary pixel locations and pinned down to maintain the desired geometric effect. The pins are called *control points*, as shown in Figure 4.33. In practice, we are often interested in aligning two

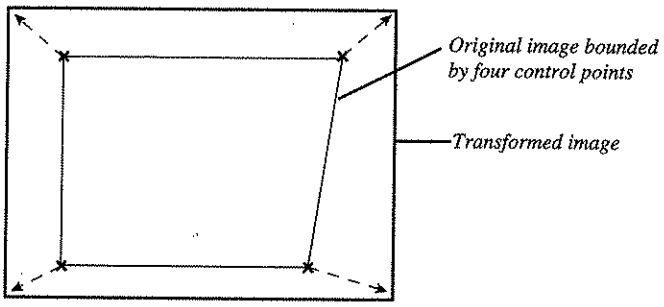

Figure 4.33 *Control points give the warping transformation the necessary information to control the warp parameters.*

images that must be joined together along a seam or overlapped for a subsequent operation. In these cases, we choose control points on each image that correspond to pixel locations that must align together.

The number of control points necessary to control the alignment of an image relates directly to the order of warp required to carry out the transformation. Three control points corresponds to a first-order warp. Six control points corresponds to a second-order warp; ten to a third-order warp, and so on.

The *pincushion lens distortion*, noted in Chapter 2, and its close relative, *barrel distortion*, can be removed from an image by using a third-order warping transformation. The equations for this warp are as follows:

$$x' = a_9x^3 + a_8y^3 + a_7x^2y + a_6y^2x + a_5x^2 + a_4y^2 + a_3x + a_2y + a_1xy + a_0$$

and

$$y' = b_9x^3 + b_8y^3 + b_7x^2y + b_6y^2x + b_5x^2 + b_4y^2 + b_3x + b_2y + b_1xy + b_0$$

Resampling

Whenever we apply a geometric transformation to an image, a *resampling* process occurs. This means that the original sample rate and orientation used to acquire the image change. Resampling occurs in the form of downsampling or upsampling. The processes of low-pass filtering and pixel interpolation can be used to estimate new or intermediate pixel brightnesses so that the resampling process more closely approximates the image appearance, as if it had been originally sampled with the transformed geometry.

Low-Pass Filtering for Downsampling

Downsampling is a spatial resolution reduction that is present whenever a geometric transformation operation transforms an image, or a portion of an image, to a size

smaller than the original image. Because the resulting image becomes smaller, it has fewer pixels defining it than the original had.

As a result of downsampling, an image has a reduced spatial frequency content capability. Remember, the sampling theorem says that the highest spatial frequency in an image can be no greater than one-half its sampling rate. So, when we shrink an image, high-frequency components in the original image may be aliased in the transformation process, and thus appear as aliasing artifacts in the resulting image.

To keep spatial aliasing from occurring, we can first reduce the frequency content of the image, using a low-pass filtering operation, so that it doesn't exceed the limit of the new sampling rate. Then, we scale the image down to the new size. As an example, if the image is to be scaled by the factors $S_x = 0.5$ and $S_y = 0.5$, then the resulting image size will be one-half that of the original in both the x and y directions. This means that the new sampling rate of the resulting image will also be one-half that of the original image. By first applying a low-pass filtering operation, we can remove all the high frequencies from the original image that exceed the new one-half resampling rate. The scaling transformation can then be applied without causing aliasing artifacts in the resulting image.

In practice, the artifacts of the aliasing phenomenon may have only minimal impact on the visual quality of the resulting image. The worst aliasing artifacts occur when repetitive patterns of high spatial frequency are present in the original image or when a motion sequence of images is involved. When these are not the case, we can often avoid the low-pass filtering operation without significantly sacrificing image quality.

Pixel Interpolation for Upsampling

Upsampling is the process of increasing the spatial resolution of an image. It is present whenever a geometric transformation operation transforms a pixel to, or from, a noninteger location. In the source-to-target transformation, this occurs when the output pixel location is not an integer location. In the target-to-source transformation, this happens when the input pixel location is not an integer location, as shown in Figure 4.34.

Whenever a transformed pixel does not fall directly on an integer pixel location, a form of pixel interpolation is used. Because the interpolation process is an estimation process that determines the pixel brightness that would exist between integer pixel locations, many differing schemes can be used, depending on the application's accuracy requirements. The most rudimentary form of interpolation is that of *nearest neighbor interpolation,* or *zero-order interpolation.* Nearest neighbor interpolation simply determines which pixel location is closest to the one desired and uses it, as shown in Figure 4.35. For instance, in scaling an image to twice its original size in both the x and y directions, the new output pixels created between the original pixels will have brightnesses equal to those of their adjacent neighbors. The visual

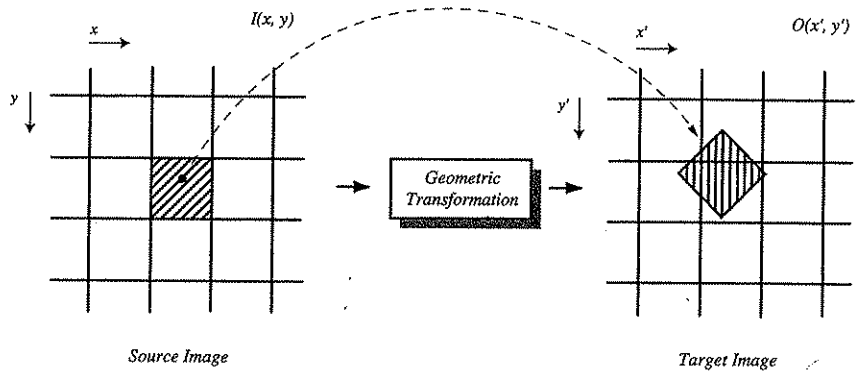

Figure 4.34a *The source-to-target transformation when output pixel locations are not integers.*

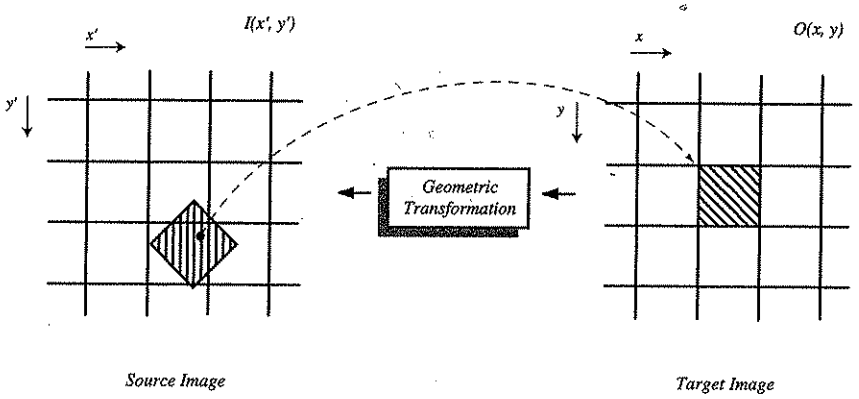

Figure 4.34b *The target-to-source transformation when input pixel locations are not integers.*

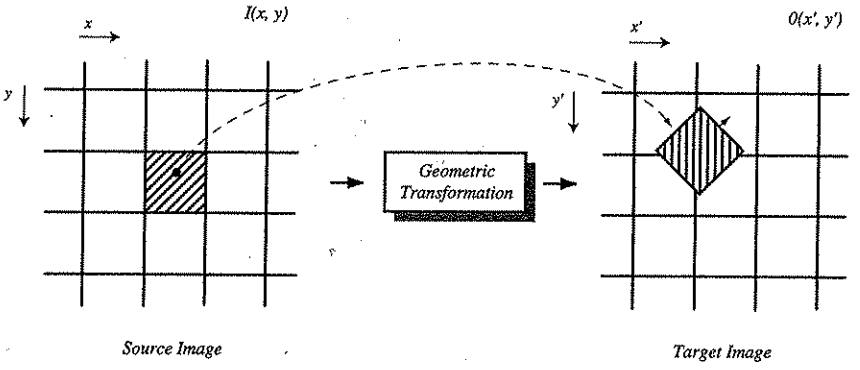

Figure 4.35a *Nearest neighbor interpolation applied to source-to-target transformations.*

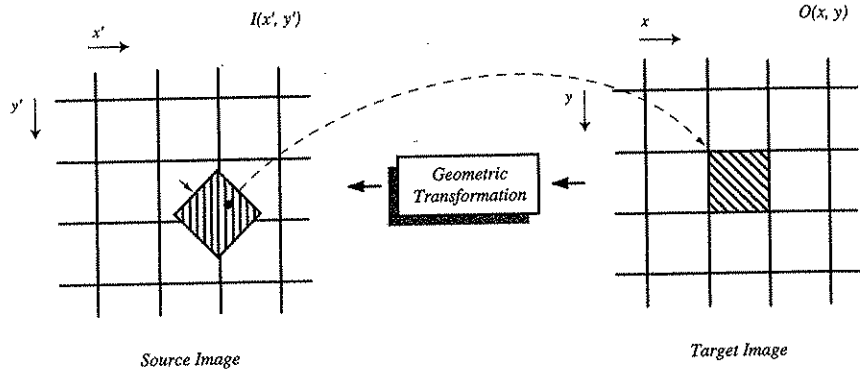

Figure 4.35b *Nearest neighbor interpolation applied to target-to-source transformations.*

effect of pixel blocking can become noticeable, however, and can degrade the resulting image.

A more accurate interpolation scheme is *bilinear interpolation*, or *first-order interpolation*. Bilinear interpolation takes a weighted average of the brightnesses of the four pixels surrounding the pixel location of interest, as shown in Figure 4.36. In this way, some of the brightness trends of the pixel neighborhood are used to estimate the pixel of interest. The weights of the average calculation are proportional to how close each of the neighboring pixels are to the desired pixel location. This is given by the equation

$$O(x,y) = (1-a)(1-b)I(x',y') + (1-a)bI(x',y'+1) + a(1-b)I(x'+1,y') + abI(x'+1,y'+1)$$

where the output pixel at $O(x,y)$ is surrounded by the mapped pixels from locations $I(x',y')$, $I(x'+1,y')$, $I(x',y'+1)$, and $I(x'+1,y'+1)$ for a target-to-source mapping.

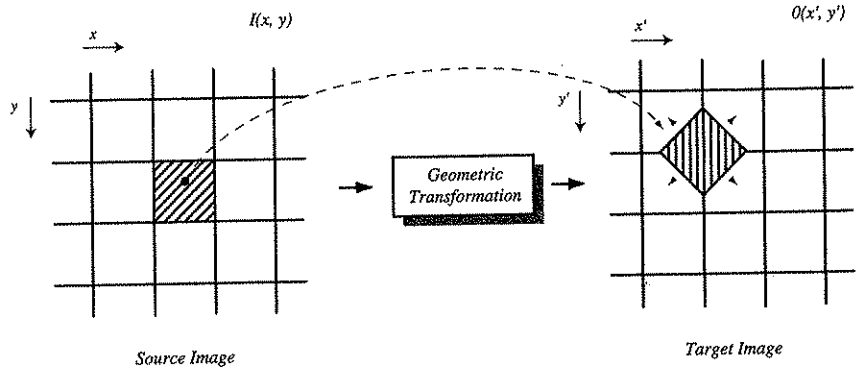

Figure 4.36a *Bilinear interpolation applied to source-to-target transformations.*

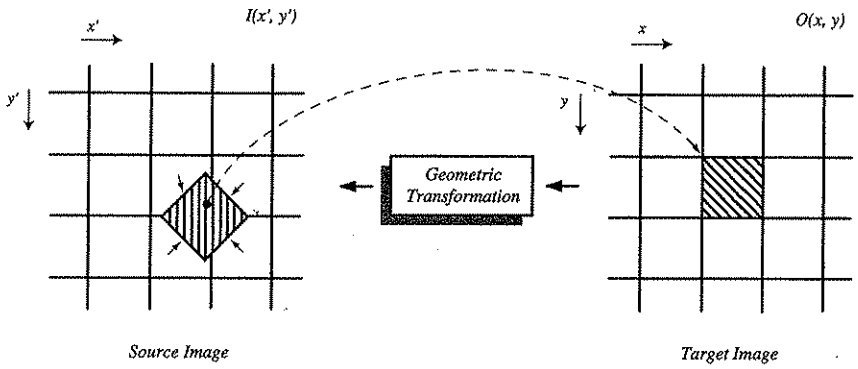

Figure 4.36b *Bilinear interpolation applied to target-to-source transformations.*

If the desired pixel is very close to one of the four neighboring pixels, its brightness will be most influenced by that neighbor. If the desired pixel is perfectly in the center of the four neighbors, each neighbor will have equal weight in the average. Bilinear interpolation produces resampled images that appear smoother and considerably more appealing than those using the nearest neighbor approach, as shown in Figure 4.37.

Other, higher-order interpolation schemes essentially factor more neighboring pixels into the weighted average. The more neighbors involved, the better the brightness estimate of a fractional pixel location. Commonly, the *second-order interpolation* scheme, known as *cubic convolution*, is used when more accuracy is needed. This scheme generally uses a neighborhood of 16 pixels in its computation. In extreme cases, up to 64 neighboring pixels can be used, yielding an interpolated pixel brightness that is virtually a perfect estimate. This type of pixel interpolation is usually reserved for use only on motion sequences where the absolute temporal smoothness of the geometric transformation is paramount. For still images, the

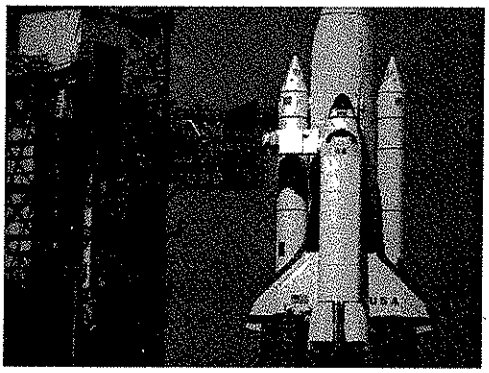

Figure 4.37a *Original Space Shuttle image.*

Figure 4.37b *Image scaled by a factor of eight using nearest neighbor interpolation.*

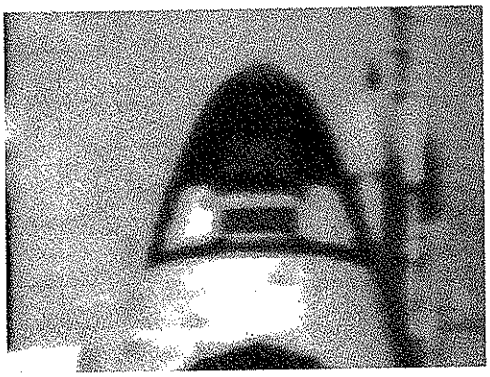

Figure 4.37c *Image scaled by a factor of eight using bilinear interpolation.*

differences between bilinear and larger interpolation neighborhoods is usually undetectable. If an application requires the use of higher order schemes, a greater computational price must be paid for such accuracy.