2. SPATIAL TRANSFORMATIONS

This chapter describes common spatial transformations derived for digital image warping applications in computer vision and computer graphics. A spatial transformation is a mapping function that establishes a spatial correspondence between all points in an image and its warped counterpart. The most elementary formulations are those that stem from a general homogeneous transformation matrix. They span two classes of simple planar mappings: affine and perspective transformations.

2.1. DEFINITIONS

A *spatial transformation* defines a geometric relationship between each point in the input and output images. An input image consists entirely of reference points whose coordinate values are known precisely. The output image is comprised of the observed (warped) data. The general mapping function can be given in two forms: either relating the output coordinate system to that of the input, or vice versa. Respectively, they can be expressed as

$$[x,y] = [X(u,v), Y(u,v)] \qquad \textbf{2.1}$$

or

$$[u,v] = [U(x,y), V(x,y)] \qquad \textbf{2.2}$$

where [*u,v*] refers to the input image coordinates corresponding to output pixel [*x,y*], and *X, Y, U*, and *V* are arbitrary mapping functions that uniquely specify the spatial transformation. Since *X* and *Y* map the input onto the output, they are referred to as the forward mapping. Similarly, the *U* and *V* functions are known as the inverse mapping since they map the output onto the input.

2.1.1. Forward Mapping

The *forward mapping* consists of copying each input pixel onto the output image at positions determined by the *X* and *Y* mapping functions. Figure 2.1 illustrates the forward mapping for the 1-D case. The discrete input and output are each depicted as a string of pixels lying on an integer grid (dots). Each input pixel is passed through the spatial transformation where it is assigned new output coordinate values. Notice that the input pixels are mapped from the set of integers to the set of real numbers. In the figure, this corresponds to the regularly spaced input samples and the irregular output distribution.

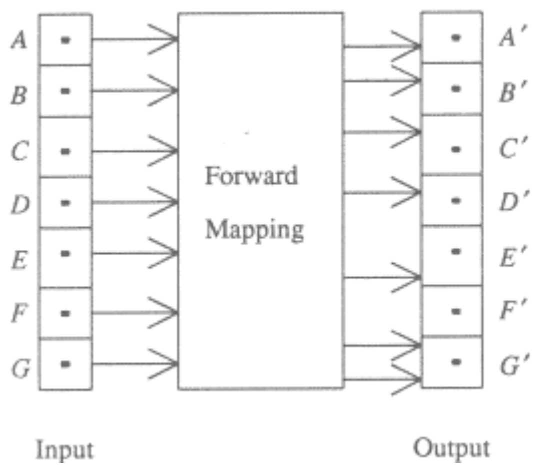

Figure 2.1 Forward mapping.

The real-valued output positions assigned by *X* and *Y* present complications at the discrete output. In the continuous domain, where pixels may be viewed as points, the mapping is straightforward. However, in the discrete domain pixels are now taken to be finite elements defined to lie on a (discrete) integer lattice. It is therefore inappropriate to implement the spatial transformation as a point-to-point mapping. Doing so can give rise to two types of problems: holes and overlaps. Holes, or patches of undefined pixels, occur when mapping contiguous input samples to sparse positions on the output grid. In Figure 2.1, *F*' a hole since it is bypassed in the input-output mapping. In contrast, overlaps occur when consecutive input samples collapse into one output pixel, as depicted in Figure 2.1 by output pixel *G*'. The shortcomings of a point-to-point mapping are avoided by using a *four-corner mapping* paradigm. This considers input pixels as square patches that may be transformed into arbitrary quadrilaterals in the output image. This has the effect of allowing the input to remain contiguous after the mapping.

Due to the fact that the projected input is free to lie anywhere in the output image, input pixels often straddle several output pixels or lie embedded in one. These two instances are illustrated in Figure 2.2. An *accumulator array* is required to properly integrate the input contributions at each output pixel. It does so by determining which fragments contribute to each output pixel and then integrating over all contributing fragments. The partial contributions are handled by scaling the input intensity in proportion to the fractional part of the pixel that it covers. Intersection tests must be performed to compute the coverage. Thus, each position in the accumulator array evaluates $\sum_{i=0}^{N} w_i f_i$ where f_i is the input value, w_i is the weight reflecting its coverage of the output pixel, and *N* is the total number of deposits into the cell. Note that *N* is free to vary among pixels and is determined only by the mapping function and the output discretization.

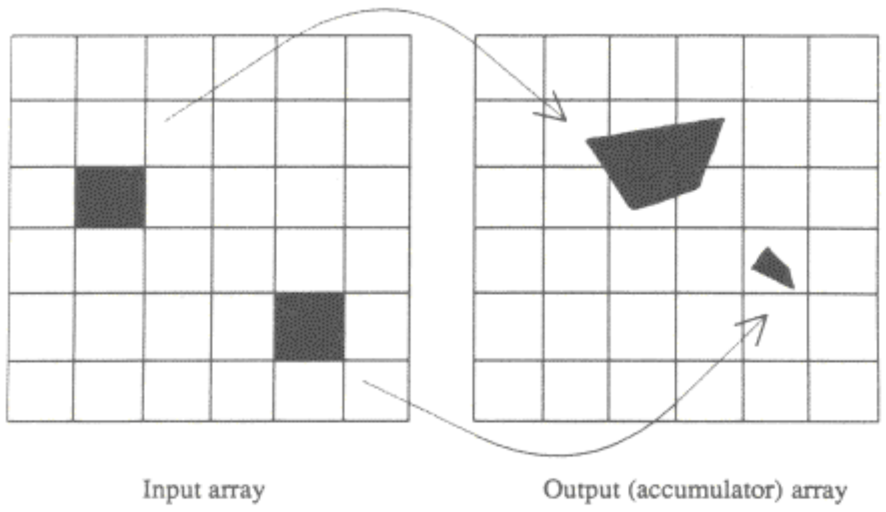

Figure 2.2 Accumulator array.

Formulating the transformation as a four-corner mapping problem allows us to avoid holes in the output image. Nevertheless, this paradigm introduces two problems in the forward mapping process. First, costly intersection tests are needed to derive the weights. Second, magnification may cause the same input value to be applied onto many output pixels unless additional filtering is employed.

Both problems can be resolved by adaptively sampling the input based on the size of the projected quadrilateral. In other words, if the input pixel is mapped onto a large area in the output image, then it is best to repeatedly subdivide the input pixel until the projected area reaches some acceptably low limit, i.e., one pixel size. As the sampling rate rises, the weights converge to a single value, the input is resampled more densely, and the resulting computation is performed at higher precision.

It is important to note that uniformly sampling the input image does not guarantee uniform sampling in the output image unless X and Y are affine (linear) mappings. Thus, for nonaffine mappings (e.g., perspective or bilinear) the input image must be adaptively sampled at rates that are spatially varying. For example, the oblique surface shown in Figure 2.3 must be sampled more densely near the horizon to account for the foreshortening due to the bilinear mapping. In general, forward mapping is useful when the input image must be read sequentially or when it does not reside entirely in memory.

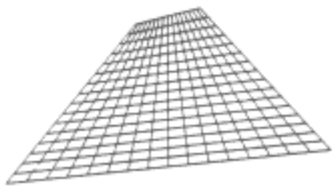

Figure 2.3 An oblique surface requiring adaptive sampling.

2.1.2. Inverse Mapping

The *inverse mapping* operates in screen order, projecting each output coordinate into the input image via *U* and *V*. The value of the data sample at that point is copied onto the output pixel. Again, filtering is necessary to combat the aliasing artifacts described in more detail later. This is the most common method since no accumulator array is necessary and since output pixels that lie outside a clipping window need not be evaluated. This method is useful when the screen is to be written sequentially, *U* and *V* are readily available, and the input image can be stored entirely in memory.

Figure 2.4 depicts the inverse mapping, with each output pixel mapped back onto the input via the spatial transformation (inverse) mapping function. Notice that the output pixels are centered on integer coordinate values. They are projected onto the input at real-valued positions. As we will see later, an interpolation stage must be introduced in order to retrieve input values at undefined (nonintegral) input positions.

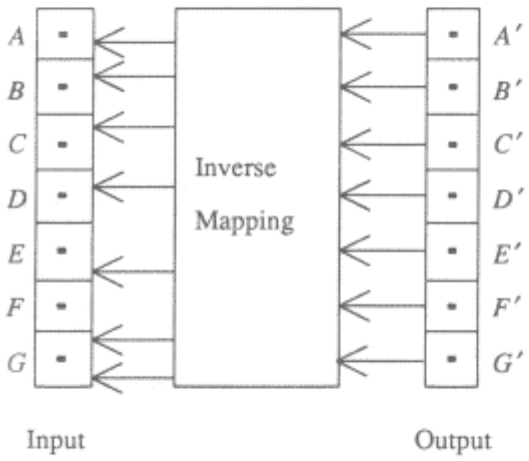

Figure 2.4 Inverse mapping.

Unlike the point-to-point forward mapping scheme, the inverse mapping guarantees that all output pixels are computed. However, the analogous problem remains to determine whether large holes are left when sampling the input. If this is the case, large amounts of input data may have been discarded while evaluating the output, thereby giving rise to artifacts. Thus, filtering is necessary to integrate the area projected onto the input. In general, though, this arrangement has the advantage of allowing interpolation to occur in the input space instead of the output space. This proves to be a much more convenient approach than forward mapping. Graphically, this is equivalent to the dual of Figure 2.2, where the input and output captions are interchanged.

In their most unconstrained form, *U* and *V* can serve to scramble the image by defining a discontinuous function. The image remains coherent only if *U* and *V* are piecewise continuous. Although there exists an infinite number of possible mapping functions, several common forms of *U* and *V* have been isolated for geometric correction and geometric distortion. The remainder of this chapter addresses these formulations.

2.2. GENERAL TRANSFORMATION MATRIX

Many simple spatial transformations can be expressed in terms of the general 3x3 transformation matrix T_1 shown in It handles scaling, shearing, rotation, reflection, translation, and perspective in 2-D. Without loss of generality, we shall ignore the component in the third dimension since we are only interested in 2-D image projections (e.g., mappings between the *uv*- and *xy*-coordinate systems).

$$[x', y', w'] = [u, v, w] \cdot T_1 \qquad \mathbf{2.3}$$

where

$$T_1 = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad \mathbf{2.4}$$

The 3x3 transformation matrix can be best understood by partitioning it into four separate sections. The 2x2 submatrix

$$T_2 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad \mathbf{2.5}$$

specifies a linear transformation for scaling, shearing, and rotation. The 1x2 matrix $[a_{31}\ a_{32}]$ produces translation. The 2x1matrix $[a_{13}\ a_{23}\ ]^T$ produces perspective transformation. Note that the superscript *T* denotes matrix transposition, whereby rows and columns are interchanged. The final element a_{33} is responsible for overall scaling.

For consistency, the transformations that follow are cast in terms of forward mapping functions *X* and *Y* that transform source images in the *uv*-coordinate system onto target images in the *xy*-coordinate system. Similar derivations apply for inverse mapping functions *U* and *V*. We note that the transformations are written in postmultiplication form. That is, the transformation matrix is written *after* the position row vector. This is equivalent to the premultiplication form where the transformation matrix precedes the position column vector. The latter form is more common in the remote sensing, computer vision, and robotics literature.

2.2.1. Homogeneous Coordinates

The general 3x3 matrix used to specify 2-D coordinate transformations operates in the *homogeneous coordinate system*. The use of homogeneous coordinates was introduced into computer graphics to provide a consistent representation for affine and perspective transformations.

Elementary 2-D mapping functions can be specified with the general 2x2 transformation matrix T_2. Applying T_2 to a 2-D position vector $[u,v]$ yields the following linear mapping functions for *X* and *Y*.

$$x = a_{11}u + a_{21}v$$
$$y = a_{12}u + a_{22}v \qquad \textbf{2.6}$$

Equations 2.6 are said to be linear because they satisfy the following two conditions necessary for any linear function $\mathcal{L}(x)$: $\mathcal{L}(x+y) = \mathcal{L}(x)+\mathcal{L}(y)$ and $\mathcal{L}(cx) = c\mathcal{L}(x)$ for any scalar c, and position vectors x and y. Unfortunately, linear transformations do not account for translations since there is no facility for adding constants. Therefore, we define $\mathcal{A}(x)$ to be an affine transformation if and only if there exists a constant t and a linear transformation $\mathcal{L}(x)$ such that $\mathcal{A}(x)=\mathcal{L}(x)+t$ for all x. Clearly linear transformations are a subset of affine transformations.

In order to accommodate affine mappings, the position vectors are augmented with an additional component, turning $[x, y]$ into $[x, y, 1]$. In addition, the translation parameters are appended to T_2 yielding

$$T_3 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \qquad \textbf{2.7}$$

The affine mapping is given as $[x, y] = [u, v, 1] \cdot T_3$. Note that the added component to $[u, v]$ has no physical significance. It simply allows us to incorporate translations into the general transformation scheme.

The 3x2 matrix T_3 used to specify an affine transformation is not square and thus does not have an inverse. Since inverses are necessary to relate the two coordinate systems (before and after a transformation), the coefficients are embedded into a 3x3 transformation matrix in order to make it invertible. Thus, the additional row introduced to T_2 by translation is balanced by appending an additional column to T_3. This serves to introduce a third component w' the transformed 2-D position vector (Eq. 2.3). The use of homogeneous coordinates to represent affine transformations is derived from this need to retain an inverse for T_3.

All 2-D position vectors are now represented with three components in a representation known as *homogeneous notation.* In general, n-dimensional position vectors now consist of n+1 elements. This formulation forces the homogeneous coordinate w' take on physical significance: it refers to the plane upon which the transformation operates. That is, a 2-D position vector $[u, v]$ lying on the $w = 1$ plane becomes a 3-D homogeneous vector $[u, v, 1]$. For convenience, all input points lie on the $w = 1$ plane to trivially facilitate translation by $[a_{31}\ a_{32}]$.

Since only 2-D transformations are of interest to us, the results of the transformation must lie on the same plane, e.g., $w'= w = 1$. However, since w' is free to take on any value in the general case, the homogeneous coordinates must be divided by w' order to be left with results in the plane $w' = w = 1$. This leads us to an important property of the homogeneous notation: *the representation of a point is no longer unique.*

Consider the implicit equation of a line in two dimensions, $ax + by + c = 0$. The coefficients a, b, and c are not unique. Instead, it is the *ratio* among coefficients that is important. Not surprisingly, equations of the form $f(x) = 0$ are said to be homogeneous equations because equality is preserved after scalar multiplication. Similarly, scalar multiples of a 2-D position vector represent the same point in a homogeneous coordinate system.

Any 2-D position vector *p* = [*x,y*] is represented by the homogeneous vector $ph = [x' \quad y' \quad w'] = [xw' \quad yw' \quad w']$, where $w' \neq 0$. To recover *p* from *ph*, we simply divide by the homogeneous coordinate *w*' so that $[x \quad y] = [x'/w' \quad y'/w']$. Consequently, vectors of the form $[xw' \quad yw' \quad w']$ form an equivalence class of homogeneous representations for the vector *p*. The division that cancels the effect of multiplication with *w*' rresponds to a projection onto the *w*'=1 plane using rays passing through the origin.

2.3. AFFINE TRANSFORMATIONS

The general representation of an *affine transformation* is

$$[x \quad y \quad 1] = [u \quad v \quad 1] \cdot \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix} \qquad \textbf{2.8}$$

Division by the homogeneous coordinate *w*' avoided by selecting *w* = *w'* = 1. Consequently, an affine mapping is characterized by a transformation matrix whose last column is equal to $[0 \ 0 \ 1]^T$. This corresponds to an *orthographic* or *parallel plane projection* from the source *uv*-plane onto the target *xy*-plane. As a result, affine mappings preserve parallel lines, allowing us to avoid foreshortened axes when performing 2-D projections. Furthermore, equispaced points are preserved (although the actual spacing in the two coordinate systems may differ). As we shall see later, affine transformations accommodate planar mappings. For instance, they can map triangles to triangles. They are, however, not general enough to map quadrilaterals to quadrilaterals. That is reserved for perspective transformations. Examples of three affine warps applied to the Checkerboard image are shown in Figure 2.5.

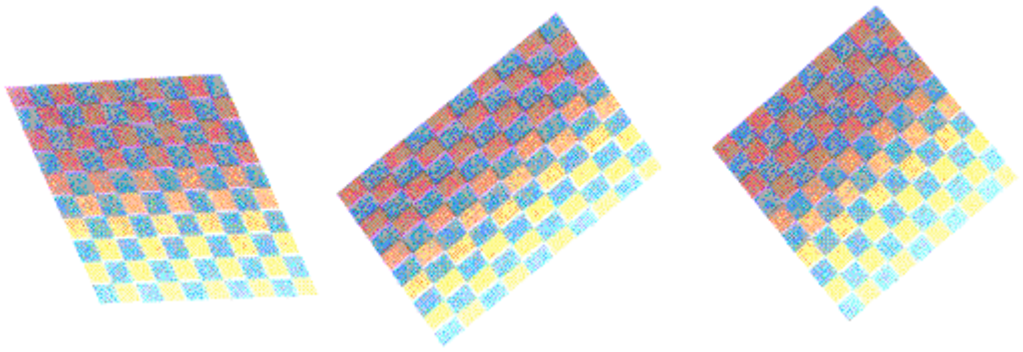

Figure 2.5 Affine warps.

For affine transformations, the forward mapping functions are

$$x = a_{11}u + a_{21}v + a_{31} \qquad \textbf{2.9}$$

$$y = a_{12}u + a_{22}v + a_{32} \qquad \textbf{2.10}$$

This accommodates translations, rotations, scale, and shear. Since the product of affine transformations is also affine, they can be used to perform a general orientation of a set of points relative to an arbitrary coordinate system while still maintaining a unity value for the homogeneous coordinate. This is necessary for generating composite transformations. We now consider special cases of the affine transformation and its properties.

2.3.1. Translation

All points are translated to new positions by adding offsets T_u and T_v to *u* and *v*, respectively. The translate transform is

$$[x \quad y \quad 1] = [u \quad v \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_u & T_v & 1 \end{bmatrix} \qquad \textbf{2.11}$$

2.3.2. Rotation

All points in the *uv*-plane are rotated about the origin through the counterclockwise angle q.

$$[x \quad y \quad 1] = [u \quad v \quad 1] \cdot \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \textbf{2.12}$$

2.3.3. Scale

All points are scaled by applying the scale factors *Su* and *Sv* to the *u* and *v* coordinates, respectively. Enlargements (reductions) are specified with positive scale factors that are larger (smaller) than unity. Negative scale factors cause the image to be reflected, yielding a mirrored image. Finally, if the scale factors are not identical, then the image proportions are altered resulting in a differentially scaled image.

$$[x \quad y \quad 1] = [u \quad v \quad 1] \cdot \begin{bmatrix} S_u & 0 & 0 \\ 0 & S_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \textbf{2.13}$$

2.3.4. Shear

The coordinate scaling described above involves only the diagonal terms a_{11} and a_{22}. We now consider the case where $a_{11} = a_{22} = 1$, and $a_{12} = 0$. By allowing a_{21} to be nonzero, *x* is made linearly dependent on both *u* and *v*, while *y* remains identical to *v*. A similar operation can be applied along the *v*-axis to compute new values for *y* while *x* remains unaffected. This effect, called shear, is therefore produced by using the off-diagonal terms. The shear transform along the *u*-axis is

$$[x \quad y \quad 1] = [u \quad v \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ H_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.14}$$

where *Hv* is used to make *x* linearly dependent on *v* as well as *u.* Similarly, the shear transform along the *v*-axis is

$$[x \quad y \quad 1] = [u \quad v \quad 1] \cdot \begin{bmatrix} 1 & H_u & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

2.3.5. Composite Transformations

Multiple transforms can be collapsed into a single composite transformation. The transforms are combined by taking the product of the 3 matrices. This is generally not a commutative operation. An example of a composite transforrnation representing a translation followed by a rotation and a scale change is given below.

$$[x', y', w'] = [u, v, w] \cdot M_{comp} \tag{2.16}$$

where

$$\begin{aligned} M_{comp} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \mathrm{T_u} & \mathrm{T_v} & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathrm{S_u} & 0 & 0 \\ 0 & \mathrm{S_v} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} S_u \cos\theta & S_v \sin\theta & 0 \\ -S_u \sin\theta & S_v \cos\theta & 0 \\ S_u(T_u \cos\theta - T_v \sin\theta) & S_v(T_u \sin\theta + T_v \cos\theta) & 1 \end{bmatrix} \end{aligned} \tag{2.17}$$

2.3.6. Inverse

The inverse of an affine transformation is itself affine. It can be readily computed from the adjoint *adj*(T_1) and determinant *det*(T_1) of transformation matrix T_1 From linear algebra, we know that $T_1^{-1} = adj(T_1) \,/\, det(T_1)$ where the adjoint of a matrix is simply the transpose of the matrix of cofactors [Strang 80]. This yields

$$\begin{aligned} [u \quad v \quad 1] &= [x \quad y \quad 1] \cdot \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & 0 \\ A_{31} & A_{32} & 1 \end{bmatrix} \\ &= [x \quad y \quad 1] \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} & 0 \\ -a_{21} & a_{11} & 0 \\ a_{21}a_{32} - a_{31}a_{22} & a_{31}a_{12} - a_{11}a_{32} & a_{11}a_{22} - a_{12}a_{21} \end{bmatrix} \end{aligned} \tag{2.18}$$

2.3.7. Inferring Affine Transformations

An affine transformation has six degrees of freedom, relating directly to coefficients a_{11}, a_{21}, a_{31}, a_{12}, a_{22}, and a_{32}. In computer graphics, these coefficients are known by virtue of the applied coordinate transformation. If an affine mapping is deemed adequate to describe the transformation, the six coefficients may be derived by specifying the coordinate correspondence of three noncollinear points in both images. Let (u_k, v_k) and (x_k,y_k) for k=0,1,2 be these three points in the reference and observed images, respectively. Equation 2.19 expresses their relationship in the form of a matrix equation. The six unknown coefficients of the affine mapping are determined by solving the system of six linear equations contained in Eq. 2.19.

$$\begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} = \begin{bmatrix} u_0 & v_0 & 1 \\ u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix} \qquad \textbf{2.19}$$

Let the system of equations given above be denoted as $X = U{\cdot}A$. In order to determine the coefficients, we isolate A by multiplying both sides with U^{-1}, the inverse of the matrix containing points (u_k, v_k). As before, $U^{-1} = adj(U) / \det(U)$ where $adj(U)$ is the adjoint of U and $det(U)$ is the determinant. Although the adjoint is always computable, an inverse will not exist unless the determinant is nonzero. Fortunately, the constraint on U to consist of *noncollinear* points serves to ensure that U is nonsingular. Consequently, the inverse U^{-1} is guaranteed to exist. Solving for the coefficients in terms of the known (u_k, v_k) and (x_k,y_k) pairs, we have $A = U^{-1} \cdot X$, or equivalently,

$$\begin{bmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & 1 \end{bmatrix} = \frac{1}{\det(U)} \begin{bmatrix} v_1 - v_2 & v_2 - v_0 & v_0 - v_1 \\ u_2 - u_1 & u_0 - u_2 & u_1 - u_0 \\ u_1 v_2 - u_2 v_1 & u_2 v_0 - u_0 v_2 & u_2 v_1 - u_1 v_0 \end{bmatrix} \cdot \begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} \qquad \textbf{2.20}$$

where

$$\det(U) = u_o(v_1 - v_2) - v_0(u_1 - u_2) + (u_1 v_2 - u_2 v_1)$$

When more than three correspondence points are available, and when these points are known to contain errors, it is common practice to approximate the coefficients by solving an overdetermined system of equations. In that case, matrix U is no longer a square 3x3 matrix and it must be inverted using any technique that solves a leastsquares linear system problem.

Since only three points are needed to infer an affine mapping, it is clear that affine transformations realize a limited set of planar mappings. Essentially, they can map an input triangle into an arbitrary triangle at the output. An input rectangle can be mapped into a parallelogram at the output. More general distortions, however, cannot be handled by affine transformations. For example, to map a rectangle into an arbitrary quadrilateral requires a perspective, bilinear, or more complex transformation.

2.4. PERSPECTIVE TRANSFORMATIONS

The general representation of a *perspective transformation* is

$$[x' \quad y' \quad w'] = [u \quad v \quad 1] \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad \textbf{2.21}$$

where $x = x'/w'$ and $y = y'/w'$.

A *perpective transformation,* or *projective mapping,* is produced when $[a_{13}\ a_{23}]^T$ is nonzero. It is used in conjunction with a projection onto a viewing plane in what is known as a *perspective* or *central projection.* Perspective transformations preserve parallel lines only when they are parallel to the projection plane. Otherwise, lines converge to a vanishing point. This has the property of foreshortening distant lines, a useful technique for rendering realistic images. For perspective transformations, the forward mapping functions are

$$x = \frac{x'}{w'} = \frac{a_{11}u + a_{21}v + a_{31}}{a_{13}u + a_{23}v + a_{33}} \qquad \textbf{2.22}$$

$$y = \frac{y'}{w'} = \frac{a_{12}u + a_{22}v + a_{32}}{a_{13}u + a_{23}v + a_{33}} \qquad \textbf{2.23}$$

They take advantage of the fact that *w*' is allowed to vary at each point and division by *w*' equivalent to a projection using rays passing through the origin. Note that affine transformations are a special case of perspective transformations where *w*' is constant over the entire image, i.e., $a_{13} = a_{23} = 0$.

Perspective transformations share several important properties with affine transformations. They are planar mappings, and thus their forward and inverse transforms are single-valued. They preserve lines in all orientations. That is, lines map onto lines (although not of the same orientation). This desirable property is lacking in more general mappings. Furthermore, the eight degrees of freedom in Eq. 2.22 is sufficient to permit planar quadrilateral-to -quadrilateral mappings. In contrast, affine transformations offer only six degrees of freedom (Eq. 2.8) and thereby facilitate only triangle-to-triangle mappings.

Examples of projective warps are shown in Figure 2.6. Note that the intersections along the edges are no longer equispaced. Also, in the rightmost image the horizontal lines remain parallel because they lie parallel to the projection plane.

2.4.1. Inverse

The inverse of a projective mapping can be easily computed in terms of the adjoint of the transformation matrix T_1. Thus, $T_1^{-1} = adj(T_1)/det(T_1)$ where $adj(T_1)$ is the adjoint of T_1 and $det(T_1)$ is the determinant. Since two matrices which are (nonzero) scalar multiples of each other are equivalent in the homogeneous coordinate system, there is no need to divide by the determinant (a scalar). Consequently, the adjoint matrix can be used in place of the inverse matrix. This proves to be a very useful result, especially since the adjoint will be well-behaved even if the determinant is very small when the matrix is nearly singular. Note that if the matrix is singular, the inverse is undefined and therefore the adjoint cannot be a scalar multiple of it. Due to these results from linear algebra, the inverse is expressed below in terms of the elements in T_1 that are used to realize the forward mapping.

$$\begin{aligned}[u \quad v \quad 1] &= [x' \quad y' \quad w'] \cdot \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \\ &= [x' \quad y' \quad w'] \cdot \begin{bmatrix} a_{22}a_{33} - a_{23}a_{32} & a_{13}a_{32} - a_{12}a_{33} & a_{12}a_{23} - a_{13}a_{22} \\ a_{23}a_{31} - a_{21}a_{33} & a_{11}a_{33} - a_{13}a_{31} & a_{13}a_{21} - a_{11}a_{23} \\ a_{21}a_{32} - a_{22}a_{31} & a_{12}a_{31} - a_{11}a_{32} & a_{11}a_{22} - a_{12}a_{21} \end{bmatrix} \end{aligned} \qquad \textbf{3.24}$$

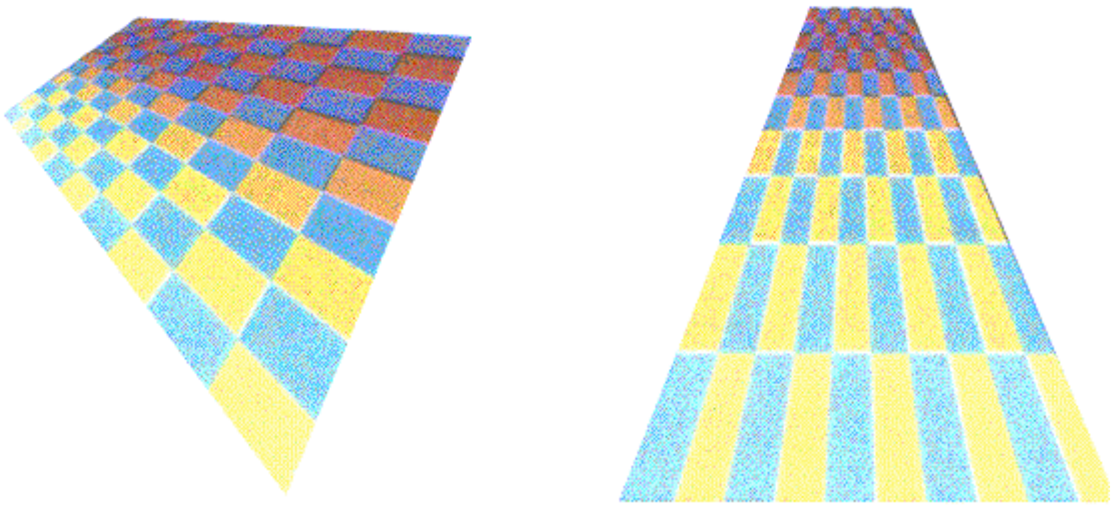
Figure 2.6 Perspective warps.

2.4.2. Inferring Perspective Transformations

A perspective transformation is expressed in terms of the nine coefficients in the general 3x3 matrix T_1. Without loss of generality, T_1 can be normalized so that $a_{33} = 1$. This leaves eight degrees of freedom for a projective mapping. The eight coefficients can be determined by establishing correspondence between four points in the reference and observed images. Let (u_k,v_k) and (x_k,y_k) for $k = 0,1,2,3$ be these four points in the reference and observed images, respectively. Assuming $a_{33} = 1$, Eqs.(2.23) and (2.24) can be rewritten as

$$x = a_{11}u + a_{21}v + a_{31} - a_{13}ux - a_{23}vx \qquad \textbf{2.25}$$

$$y = a_{12}u + a_{22}v + a_{32} - a_{13}uy - a_{23}vy \qquad \textbf{2.26}$$

Applying Eqs. (2.26) and (2.27) to the four pairs of correspondence points yields the 8x8 system of equations shown in Eq. 2.28

$$\begin{bmatrix} u_0 & v_0 & 1 & 0 & 0 & 0 & -u_0x_0 & -v_0x_0 \\ u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1x_1 & -v_1x_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -u_2x_2 & -v_2x_2 \\ u_3 & v_3 & 1 & 0 & 0 & 0 & -u_3x_3 & -v_3x_3 \\ 0 & 0 & 0 & u_0 & v_0 & 1 & -u_0y_0 & -v_0y_0 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -u_1y_1 & -v_1y_1 \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -u_2y_2 & -v_2y_2 \\ 0 & 0 & 0 & u_3 & v_3 & 1 & -u_2y_2 & -v_3y_3 \end{bmatrix} \cdot A = X \qquad \textbf{2.27}$$

where $A = [a_{11}\ a_{21}\ a_{31}\ a_{12}\ a_{22}\ a_{32}\ a_{13}\ a_{23}]^T$ are the unknown coefficients, and $X = [x_0\ x_1\ x_2\ x_3\ y_0\ y_1\ y_2\ y_3]^T$ are the known coordinates in the observed image.

The coefficients are determined by solving the linear system. This yields a solution to the general (planar) quadrilateral-to-quadrilateral problem. Speedups are possible when considering several special cases: square-to-quadrilateral, quadrilateral-to-square, and quadrilateral-to-quadrilateral using the results of the last two cases. We now consider each case individually. A detailed exposition is found in [Heckbert 89].

2.4.2.1. Case 1: Square-to-Quadrilateral

Consider mapping a unit square onto an arbitrary quadrilateral. The following four-point correspondences are established from the *uv*-plane onto the *xy*-plane.

$$\begin{aligned} (0,0) &\rightarrow (x_0 \quad y_0) \\ (1,0) &\rightarrow (x_1 \quad y_1) \\ (1,1) &\rightarrow (x_2 \quad y_2) \\ (0,1) &\rightarrow (x_3 \quad y_3) \end{aligned}$$

In this case, the eight equations become

$$\begin{aligned} a_{31} &= x_0 \\ a_{11} + a_{31} - a_{13}x_1 &= x_1 \\ a_{11} + a_{21} + a_{31} - a_{13}x_2 - a_{23}x_2 &= x_2 \\ a_{21} + a_{31} - a_{23}x_3 &= x_3 \\ a_{32} &= y_0 \\ a_{12} + a_{32} - a_{13}y_1 &= y_1 \\ a_{12} + a_{22} + a_{32} - a_{13}y_2 - a_{23}x_2 &= y_2 \\ a_{22} + a_{31} - a_{23}y_3 &= y_3 \end{aligned} \qquad \textbf{2.28}$$

The solution can take two forms, depending on whether the mapping is affine or perspective. We define the following terms for our discussion.

$$\begin{aligned} \Delta x_1 &= x_1 - x_2, \quad \Delta x_2 = x_3 - x_2, \quad \Delta x_3 = x_0 - x_1 + x_2 - x_3 \\ \Delta y_1 &= y_1 - y_2, \quad \Delta y_2 = y_3 - y_2, \quad \Delta y_3 = y_0 - y_1 + y_2 - y_3 \end{aligned} \qquad \textbf{2.29}$$

If $\Delta x_3 = 0$ and $\Delta y_3 = 0$, then the *xy* quadrilateral is a parallelogram. This implies that the mapping is affine. As a result, we obtain the following coefficients.

$$\begin{aligned}
a_{11} &= x_1 - x_0 \\
a_{21} &= x_2 - x_1 \\
a_{31} &= x_0 \\
a_{12} &= y_1 - y_0 \\
a_{22} &= y_2 - y_1 \\
a_{32} &= y_0 \\
a_{32} &= 0 \\
a_{23} &= 0
\end{aligned} \qquad \mathbf{2.30}$$

If, however, $\Delta x_3 \neq 0$ or $\Delta y_3 \neq 0$ then the mapping is projective. The coefficients of the perspective transformation are

$$a_{13} = \frac{\begin{vmatrix} \Delta x_3 & \Delta x_2 \\ \Delta y_3 & \Delta y_2 \end{vmatrix}}{\begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}}, \quad a_{23} = \frac{\begin{vmatrix} \Delta x_1 & \Delta x_3 \\ \Delta y_1 & \Delta y_3 \end{vmatrix}}{\begin{vmatrix} \Delta x_1 & \Delta x_2 \\ \Delta y_1 & \Delta y_2 \end{vmatrix}} \qquad \mathbf{2.31}$$

$$\begin{aligned}
a_{11} &= x_1 - x_0 + a_{13}x_1 \\
a_{21} &= x_3 - x_0 + a_{23}x_3 \\
a_{31} &= x_0 \\
a_{12} &= y_1 - y_0 + a_{13}y_1 \\
a_{22} &= y_3 - y_0 + a_{23}y_3 \\
a_{32} &= y_0 \\
a_{32} &= y_0 \\
a_{23} &= 0
\end{aligned}$$

This proves to be faster than the direct solution with a linear system solver. The computation may be generalized to map arbitrary rectangles onto quadrilaterals by pre-multiplying with a scale and translation matrix.

2.4.2.2. Case 2: Quadrilateral-to-Square

This case is the inverse of the mapping already considered. As discussed earlier, the adjoint of a projective mapping can be used in place of the inverse. Thus, the simplest solution is to compute the square-to-quadrilateral mapping coefficients described above to find the inverse of the desired mapping, and then take its adjoint to compute the quadrilateral-to-square mapping.

2.4.2.3. Case 3: Quadrilateral-to-Quadrilateral

The results of the last two cases may be cascaded to yield a fast solution to the general quadrilateral-to-quadrilateral mapping problem. Figure 2.7 depicts this formulation.

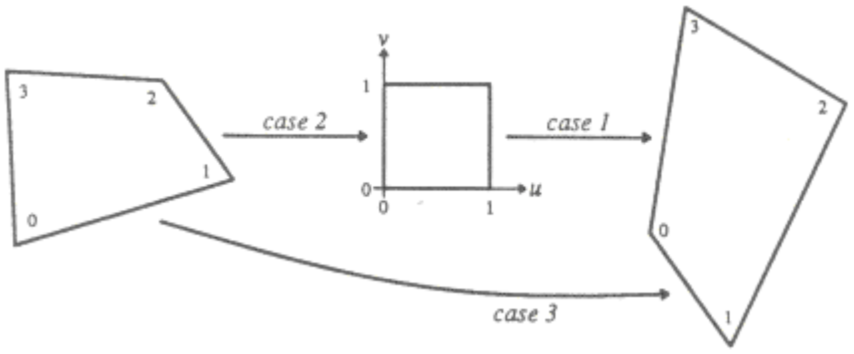

Figure 2.7 Quadrilateral-to-quadrilateral mapping [Heckbert 89].

The general quadrilateral-to-quadrilateral problem is also known as *four-corner mapping*. Perspective transformations offer a planar solution to this problem. When the quadrilaterals become nonplanar, however, more general solutions are necessary. Bilinear transformations are an example of the simplest mapping functions that address four-corner mappings for nonplanar quadrilaterals.

2.5. BILINEAR TRANSFORMATIONS

The general representation of a *bilinear transformation* is

$$[x \quad y] = [uv \quad u \quad v \quad 1] \cdot \begin{bmatrix} a_3 & b_3 \\ a_2 & b_2 \\ a_1 & b_1 \\ a_0 & b_0 \end{bmatrix} \qquad \textbf{2.32}$$

A bilinear transformation, or *bilinear mapping*, handles the four-corner mapping problem for nonplanar quadrilaterals. It is most commonly used in the forward mapping formulation where rectangles are mapped onto nonplanar quadrilaterals.

Bilinear mappings preserve lines that are horizontal or vertical in the source image. This follows from the bilinear interpolation used to realize the transformation. Thus, points along horizontal and vertical lines in the source image (including borders) remain equispaced. This is a property shared with affine transformations. However, lines not oriented along these two directions (e.g., diagonals) are not preserved as lines. Instead, diagonal lines map onto quadratic curves at the output. Examples of bilinear warps are shown in Figure 2.8.

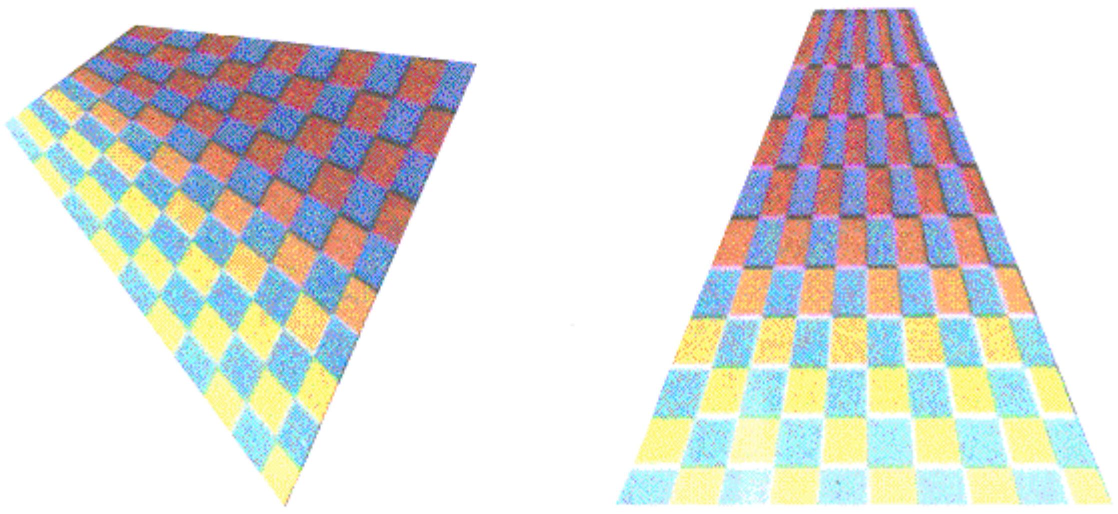

Figure 2.8 Bilinear warps.

Bilinear mappings are defined through piecewise functions that must interpolate the coordinate assignments specified at the vertices. This scheme is based on bilinear interpolation to evaluate the *X* and *Y* mapping functions. We illustrate this method below for computing *X (u,v).* An identical procedure is performed to compute *Y (u,v).*

2.5.1. Bilinear Interpolation

Bilinear interpolation utilizes a linear combination of the four "closest" pixel values to produce a new, interpolated value. Given four points, (u_0,v_0), (u_1,v_1), (u_2,v_2), and (u_3,v_3), and their respective function values x_0, x_1, x_2, and x_3, any intermediate coordinate $X(u,v)$ may be computed by the expression

$$X(u,v) = a_0 + a_1 u + a_2 v + a_3 uv \qquad \textbf{2.33}$$

where the a_i coefficients are obtained by solving

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & u_0 & v_0 & u_0 v_0 \\ 1 & u_1 & v_1 & u\ v_1 \\ 1 & u_1 & v_2 & u_2 v_2 \\ 1 & u_1 & v_3 & u_3 v_3 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad \textbf{2.34}$$

Since the four points are assumed to lie on a rectangular grid, we rewrite them in the above matrix in terms of u_0, u_1, v_0, and v_2. Namely, the points are (u_0,v_0), (u_1,v_0), (u_0,v_2), and (u_1,v_2), respectively. Solving for a_i and substituting into Eq. 2.35 yields

$$X(u',v') = x_0 + (x_1 - x_0)\cdot u' + (x_2 - x_0)\cdot v' + (x_3 - x_2 - x_1 + x_0)\cdot u'v' \qquad \textbf{2.35}$$

where $u', v' \in (0,1)$ are normalized coordinates that span the rectangle, and

$$u = u_0 + (u_1 - u_0) \cdot u'$$
$$v = v_0 + (v_1 - v_0) \cdot v' \qquad \textbf{2.36}$$

Therefore, given coordinates (u,v) and function values (x_0,x_1,x_2,x_3), the normalized coordinates (u',v') are computed and the point correspondence (x,y) in the arbitrary quadrilateral is determined by Eq. 2.37. Figure 2.9 depicts this bilinear interpolation for the X mapping function.

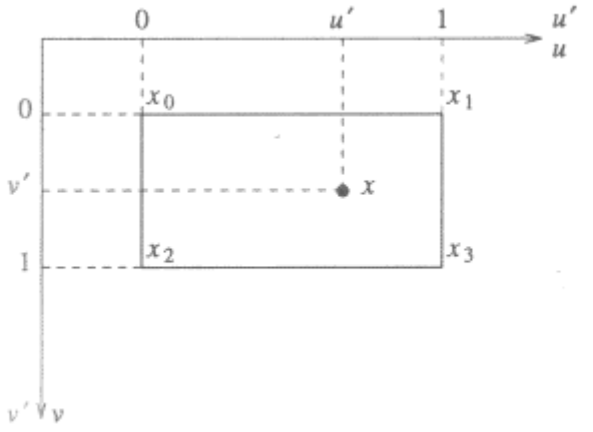

Figure 2.9 Bilinear interpolation.

2.5.2. Separability

The bilinear mapping is a separable transformation: it is the product of two 1-D mappings, each operating along orthogonal axes. This property enables us to easily extend 1-D linear interpolation into two dimensions, resulting in a computationally efficient algorithm. The algorithm requires two passes, with the first pass applying 1-D linear interpolation along the horizontal direction, and the second pass interpolating along the vertical direction. For example, consider the rectangle shown in Figure 2.10. Points x_{01} and x_{23} are interpolated in the first pass. These results are then used in the second pass to compute the final value x.

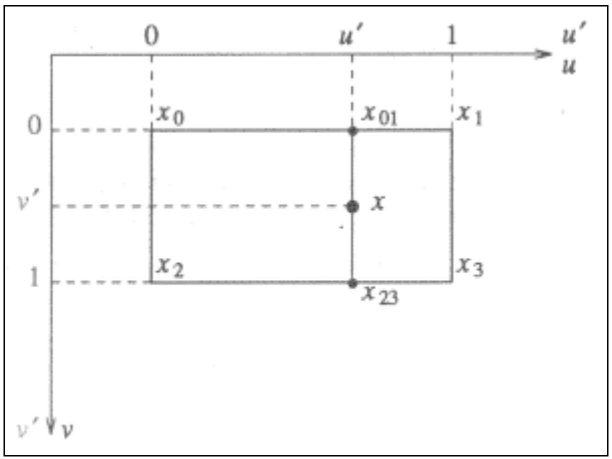

Figure 2.10 Separable bilinear interpolation.

Up to numerical inaccuracies, the separable results can be shown to be identical with the solution given in Eq. 2.37. In the first (horizontal) pass, we compute

$$\begin{aligned} x_{01} &= x_0 + (x_1 - x_0)\cdot u' \\ x_{23} &= x_2 + (x_3 - x_2)\cdot u' \end{aligned} \qquad \textbf{2.37}$$

These two intermediate results are then combined in the second (vertical) pass to yield the final value

$$\begin{aligned} x &= x_{01} + (x_{23} - x_{01})\cdot v' \\ &= x_0 + (x_1 - x_0)\cdot u' + [(x_2 - x_0) + (x_3 - x_2 - x_1 + x_0)\cdot u']\cdot v' \\ &= x_0 + (x_1 - x_0)\cdot u' + (x_2 - x_0)\cdot v' + (x_3 - x_2 - x_1 + x_0)\cdot u'\cdot v' \end{aligned} \qquad \textbf{2.38}$$

Notice that this result is identical with the classic solution derived in Eq. 2.36.

2.5.3. Inverse

The opposite problem is posed: given a normalized coordinate (*x*',y') in an arbitrary (distorted) quadrilateral, find its position in the rectangle. Two solutions are presented below.

By inverting Eq. 2.35, we can determine the normalized coordinate (*u*',v') corresponding to the given coordinate (*x*,*y*). The derivation is given below. First, we rewrite the expressions for *x* and *y* in terms of *u* and *v*, as given in Eq. 2.35.

$$\begin{aligned} x &= a_0 + a_1 u + a_2 v + a_3 uv \\ y &= b_0 + b_1 u + b_2 v + b_3 uv \end{aligned} \qquad \textbf{2.39}$$

Isolating *u* in Eq. (2.41) gives us

$$u = \frac{x - a_0 - a_2 v}{a_1 + a_3 v} \qquad \textbf{2.40}$$

In order to solve this, we must determine *v*. This can be done by substituting Eq.2.42 into Eq. 2.41. Multiplying both sides by ($a_1 + a_3 v$) yields

$$y(a_1 + a_3 y) = b_0(a_1 + a_3 v) + b_1(x - a_0 - a_2 v) + b_2 v(a_1 + a_3 v) + b_3(x - a_0 - a_2 v) \qquad \textbf{2.41}$$

This can be rewritten as

$$c_2 v^2 + c_1 v + c_0 = 0 \qquad \textbf{2.42}$$

where

$$\begin{aligned} c_0 &= a_1(b_0 - y) + b_1(x - a_0) \\ c_1 &= a_3(b_0 - y) + b_3(x - a_0) + a_1 b_2 - a_2 b_1) \\ c_2 &= a_3 b_{20} - a_2 b_3 \end{aligned} \qquad \textbf{2.43}$$

The inverse mapping for v thus requires the solution of a quadratic equation. Once v is determined, it is plugged into Eq. 2.41 to compute u. Clearly, the inverse transform is multi-valued and is more difficult to compute than the forward transform.

2.5.4. Interpolation Grid

Mapping from an arbitrary grid to a rectangular grid is an important step in performing any 2-D interpolation within an arbitrary quadrilateral. The procedure is given as follows.

To any point (x,y) inside an interpolation region defined by four arbitrary points, a normalized coordinate (u',v') is associated in a rectangular region. This makes use of the results derived above. A geometric interpretation is given in Figure 2.11, where the normalized coordinates can be found by determining the grid lines that intersect at (x,y) (point P). Given the positions labeled at the vertices, the normalized coordinates (u',v') are given as

$$u' = \frac{P_{01}P_0}{P_1P_0} = \frac{P_{23}P_2}{P_3P_2} \qquad \mathbf{2.44}$$

$$v' = \frac{P_{02}P_0}{P_2P_0} = \frac{P_{13}P_1}{P_3P_1} \qquad \mathbf{2.45}$$

The function values at the four quadrilateral vertices are assigned to the rectangle vertices.

A rectangular grid interpolation is then performed, using the normalized coordinates to index the interpolation function.

The result is then assigned to point (x,y) in the distorted plane.

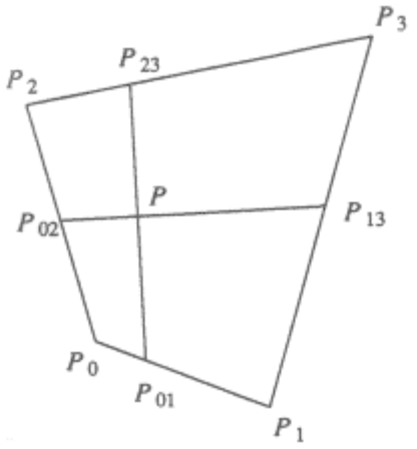

Figure 2.11 Geometric interpretation of arbitrary grid interpolation.

2.6. SUMMARY

Spatial transformations are given by mapping functions that relate the coordinates of two images, e.g., the input image and the transformed output. In computer graphics a general transformation matrix suffices for simple affine and perspective planar mappings. Bilinear transformations are also popular, particularly owing to their computational advantages in terms of separability. They do not preserve straight lines for all orientations, their utility in computer graphics is somewhat undermined with respect to the more predictable results obtained from affine and perspective transformations.

References:

[1] George Wolberg. **Digital Image Warping**. IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.