Lecture 2: Geometric Image Transformations

Harvey Rhody
Chester F. Carlson Center for Imaging Science
Rochester Institute of Technology
rhody@cis.rit.edu

September 8, 2005

Abstract

Geometric transformations are widely used for image registration and the removal of geometric distortion. Common applications include construction of mosaics, geographical mapping, stereo and video.

Spatial Transformations of Images

A spatial transformation of an image is a geometric transformation of the image coordinate system.

It is often necessary to perform a spatial transformation to:

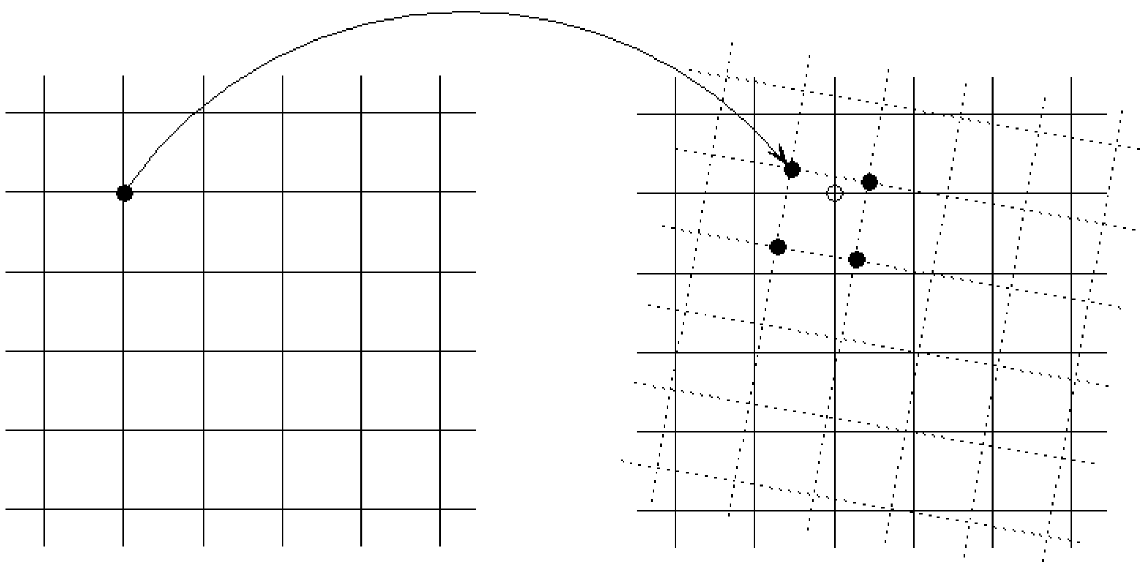
- Align images that were taken at different times or with different sensors
- Correct images for lens distortion
- Correct effects of camera orientation
- Image morphing or other special effects

Spatial Transformation

In a spatial transformation each point (x, y) of image A is mapped to a point (u, v) in a new coordinate system.

$$u = f_1(x, y)$$

$$v = f_2(x, y)$$



Mapping from (x, y) to (u, v) coordinates. A digital image array has an implicit grid that is mapped to discrete points in the new domain. These points may not fall on grid points in the new domain.

Affine Transformation

An affine transformation is any transformation that preserves collinearity (i.e., all points lying on a line initially still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation).

In general, an affine transformation is a composition of rotations, translations, magnifications, and shears.

$$u = c_{11}x + c_{12}y + c_{13}$$

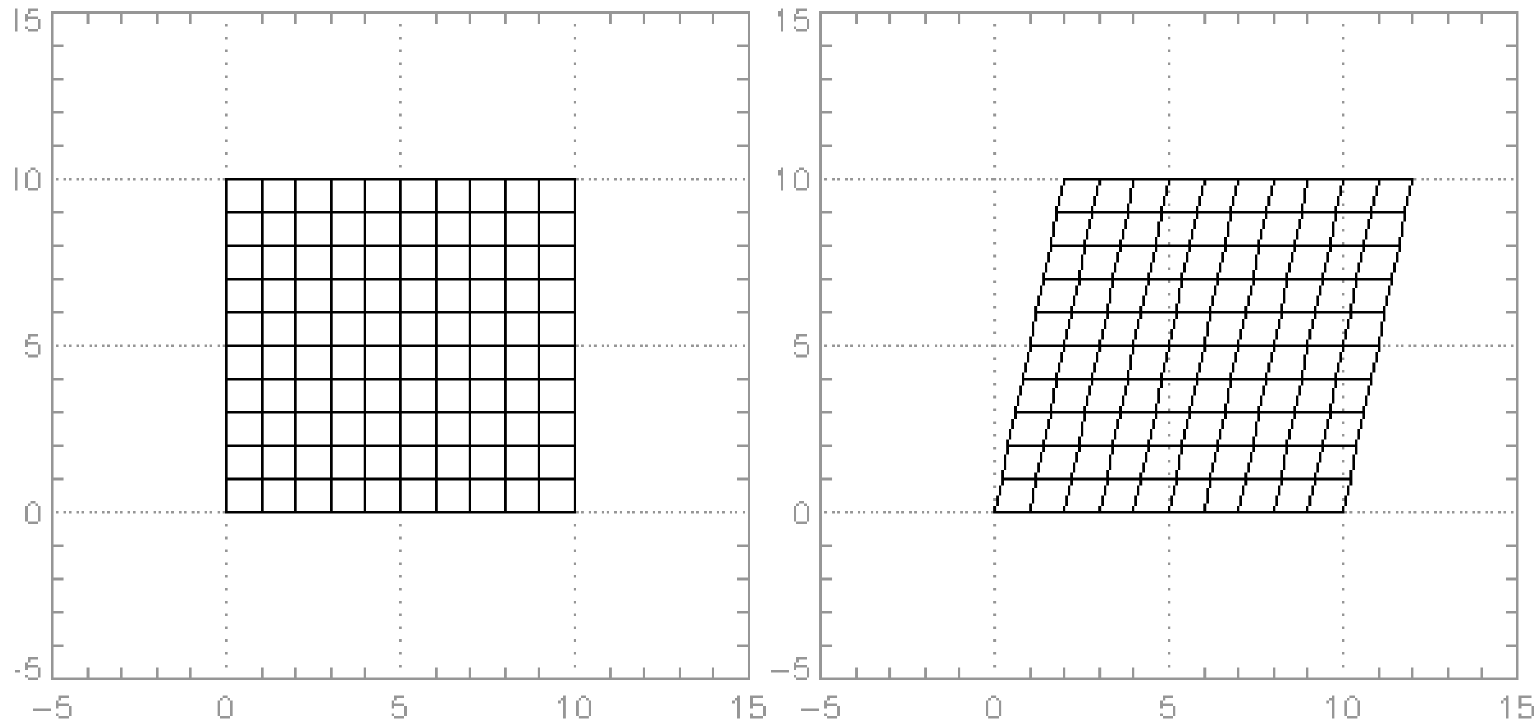
$$v = c_{21}x + c_{22}y + c_{23}$$

c_{13} and c_{23} affect translations, c_{11} and c_{22} affect magnifications, and the combination affects rotations and shears.

Affine Transformation

A shear in the x direction is produced by

$$u = x + 0.2y$$
$$v = y$$

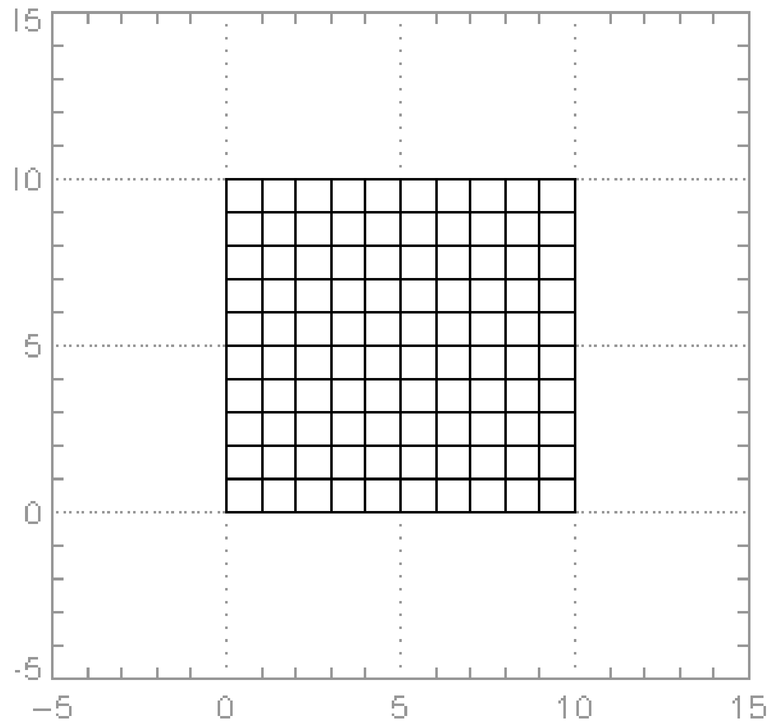


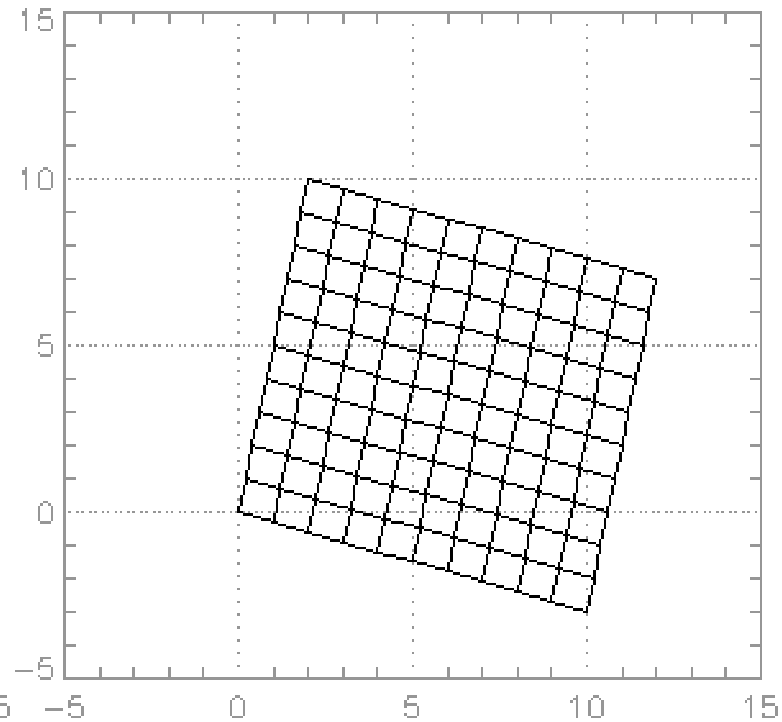
Affine Transformation

This produces as both a shear and a rotation.

$$u = x + 0.2y$$

$$v = -0.3x + y$$



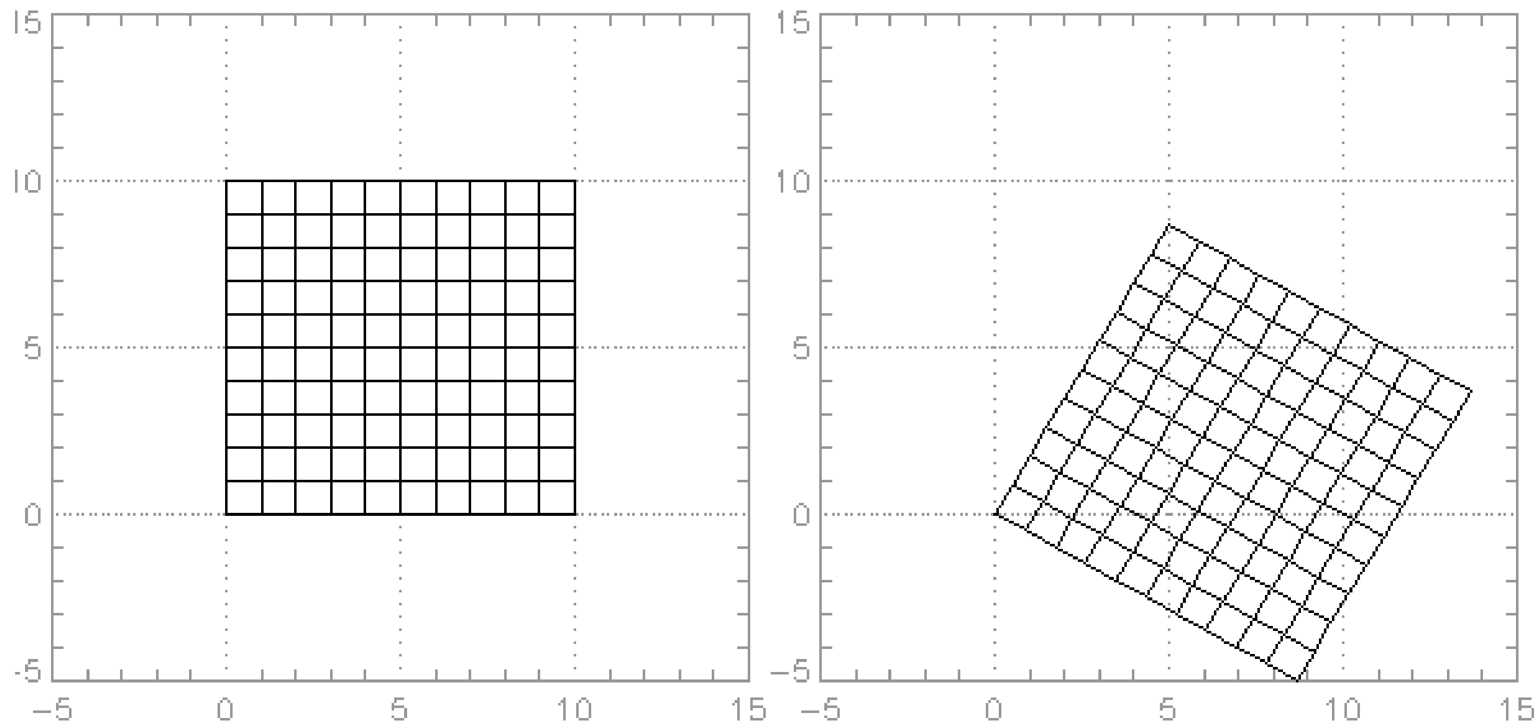


Affine Transformation

A rotation is produced by θ is produced by

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

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



Combinations of Transforms

Complex affine transforms can be constructed by a sequence of basic affine transforms.

Transform combinations are most easily described in terms of matrix operations. To use matrix operations we introduce *homogeneous coordinates*. These enable all affine operations to be expressed as a matrix multiplication. Otherwise, translation is an exception.

The affine equations are expressed as

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

An equivalent expression using matrix notation is

$$\mathbf{q} = \mathbf{T}\mathbf{p}$$

where $\mathbf{q}$, $\mathbf{T}$ and $\mathbf{q}$ are the defined above.

Transform Operations

The transformation matrices below can be used as building blocks.

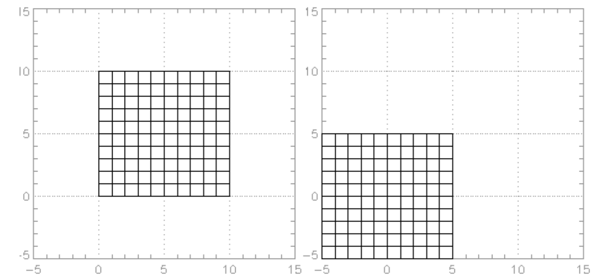
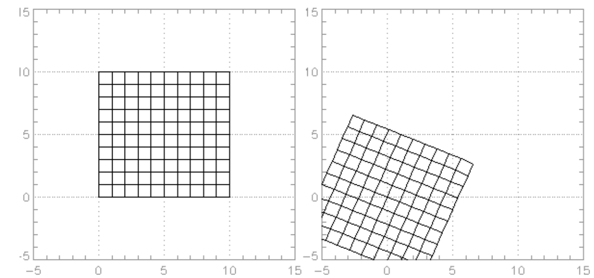
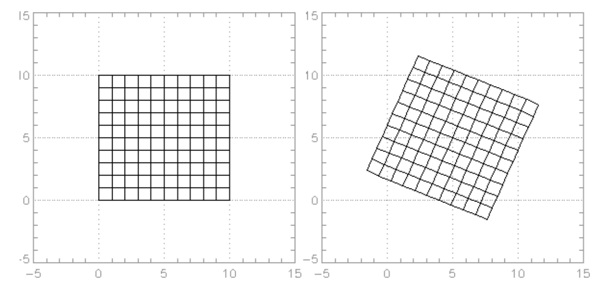
$$\mathbf{T} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Translation by } (x_0, y_0)$$

$$\mathbf{T} = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Scale by } s_1 \text{ and } s_2$$

$$\mathbf{T} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{Rotate by } \theta$$

You will usually want to translate the center of the image to the origin of the coordinate system, do any rotations and scalings, and then translate it back.

Combined Transform Operations

Operation	Expression	Result
Translate to Origin	$$\mathbf{T}_1 = \begin{bmatrix} 1.00 & 0.00 & -5.00 \\ 0.00 & 1.00 & -5.00 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$$	
Rotate by 23 degrees	$$\mathbf{T}_2 = \begin{bmatrix} 0.92 & 0.39 & 0.00 \\ -0.39 & 0.92 & 0.00 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$$	
Translate to original location	$$\mathbf{T}_3 = \begin{bmatrix} 1.00 & 0.00 & 5.00 \\ 0.00 & 1.00 & 5.00 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$$	

Composite Affine Transformation

The transformation matrix of a sequence of affine transformations, say $\mathbf{T}_1$ then $\mathbf{T}_2$ then $\mathbf{T}_3$ is

$$\mathbf{T} = \mathbf{T}_3\mathbf{T}_2\mathbf{T}_3$$

The composite transformation for the example above is

$$\mathbf{T} = \mathbf{T}_3\mathbf{T}_2\mathbf{T}_1 = \begin{bmatrix} 0.92 & 0.39 & -1.56 \\ -0.39 & 0.92 & 2.35 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$$

Any combination of affine transformations formed in this way is an affine transformation.

The inverse transform is

$$\mathbf{T}^{-1} = \mathbf{T}_1^{-1}\mathbf{T}_2^{-1}\mathbf{T}_3^{-1}$$

If we find the transform in one direction, we can invert it to go the other way.

Composite Affine Transformation RST

Suppose that you want the composite representation for translation, scaling and rotation (in that order).

$$H = RST = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_0 & 0 & 0 \\ 0 & s_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & x_1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_0 \cos\theta & s_1 \sin\theta & s_0 x_0 \cos\theta + s_1 x_1 \sin\theta \\ -s_0 \sin\theta & s_1 \cos\theta & s_1 x_1 \cos\theta - s_0 x_0 \sin\theta \end{bmatrix}$$

Given the matrix H one can solve for the five parameters.

How to Find the Transformation

Suppose that you are given a pair of images to align. You want to try an affine transform to register one to the coordinate system of the other. How do you find the transform parameters?

Image A

Image B

Point Matching

Find a number of points $\{\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{n-1}\}$ in image A that match points $\{\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_{n-1}\}$ in image B. Use the homogeneous coordinate representation of each point as a column in matrices $\mathbf{P}$ and $\mathbf{Q}$:

$$\mathbf{P} = \begin{bmatrix} x_0 & x_1 & \ldots & x_{n-1} \\ y_0 & y_1 & \ldots & y_{n-1} \\ 1 & 1 & \ldots & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 & \ldots & \mathbf{p}_{n-1} \end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix} u_0 & u_1 & \ldots & u_{n-1} \\ v_0 & v_1 & \ldots & v_{n-1} \\ 1 & 1 & \ldots & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{q}_0 & \mathbf{q}_1 & \ldots & \mathbf{q}_{n-1} \end{bmatrix}$$

Then

$$\mathbf{q} = \mathbf{H}\mathbf{p} \quad \text{becomes} \quad \mathbf{Q} = \mathbf{H}\mathbf{P}$$

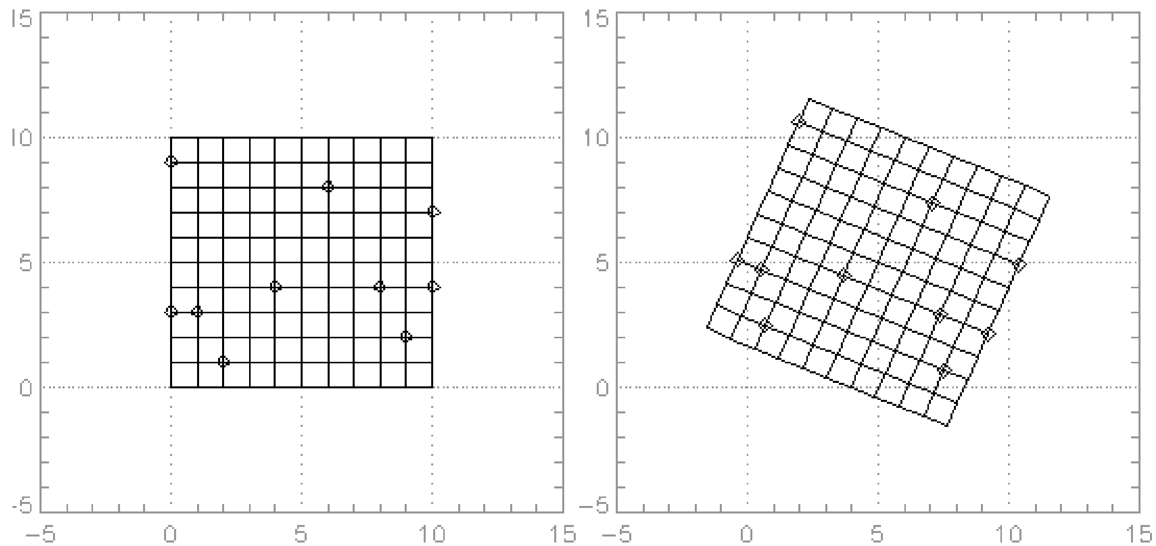
The solution for $\mathbf{H}$ that provides the minimum mean-squared error is

$$\mathbf{H} = \mathbf{Q}\mathbf{P}^T(\mathbf{P}\mathbf{P}^T)^{-1} = \mathbf{Q}\mathbf{P}^\dagger$$

where $\mathbf{P}^\dagger = \mathbf{P}^T(\mathbf{P}\mathbf{P}^T)^{-1}$ is the (right) *pseudo-inverse* of $\mathbf{P}$.

Point Matching

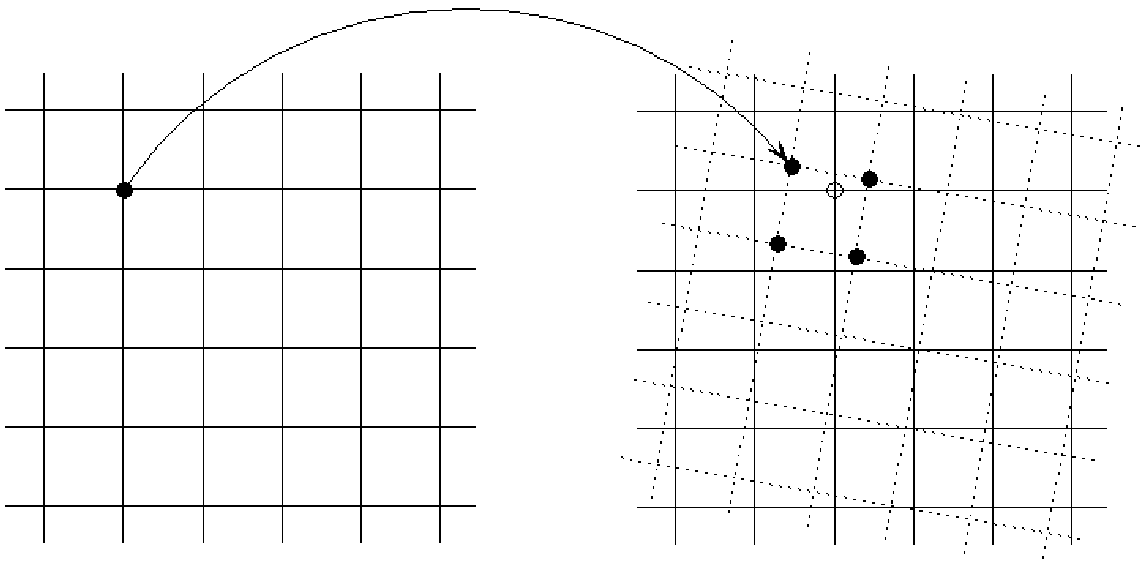
The transformation used in the previous example can be found from a few matching points chosen randomly in each image.



Many image processing tools, such as ENVI, have tools to enable point-and-click selection of matching points.

Interpolation

Interpolation is needed to find the value of the image at the grid points in the target coordinate system. The mapping T locates the grid points of A in the coordinate system of B, but those grid points are not on the grid of B.

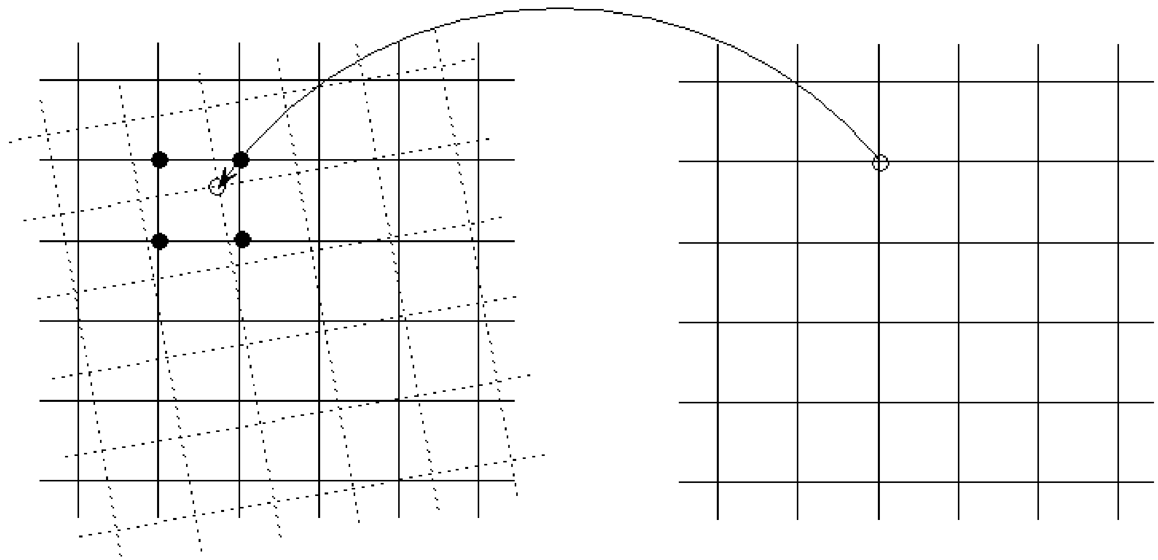


To find the values on the grid points of B we need to interpolate from the values at the projected locations.

Finding the closest projected points to a given grid point can be computationally expensive.

Inverse Projection

Projecting the grid of B into the coordinate system of A maintains the known image values on a regular grid. This makes it simple to find the nearest points for each interpolation calculation.



Let $\mathbf{Q}_g$ be the homogeneous grid coordinates of B and let $\mathbf{H}$ be the transformation from A to B. Then

$$\mathbf{P} = \mathbf{H}^{-1}\mathbf{Q}_g$$

represents the projection from B to A. We want to find the value at each point $\mathbf{P}$ given from the values on $\mathbf{P}_g$, the homogeneous grid coordinates of A.

Methods of Interpolation

There are several common methods of interpolation:

- Nearest neighbor – simplest and fastest
- Triangular – Uses three points from a bounding triangle. Useful even when the known points are not on a regular grid.
- Bilinear – Uses points from a bounding rectangle. Useful when the known points are on a regular grid.

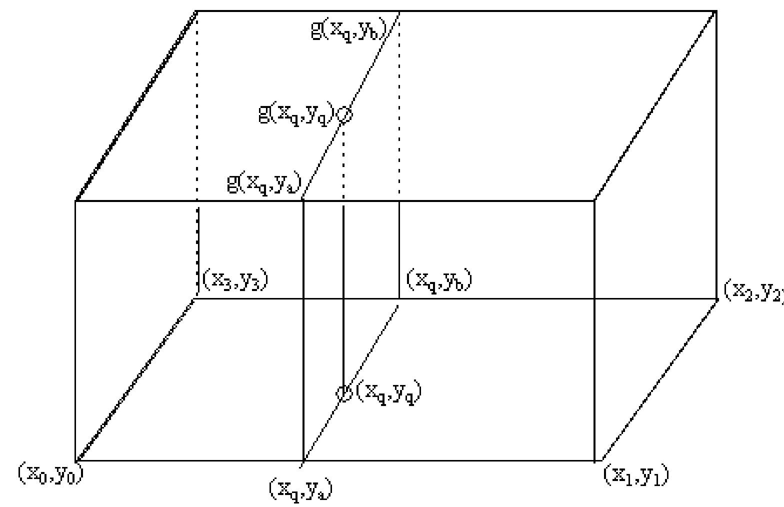
Bilinear Interpolation

Suppose that we want to find the value $g(q)$ at a point q that is interior to a four-sided figure with vertices $\{p_0, p_1, p_2, p_3\}$. Assume that these points are in order of progression around the figure and that p_0 is the point farthest to the left.

1. Find the point (x_q, y_a) between p_0 and p_1. Compute $g(x_q, y_a)$ by linear interpolation between $f(p_0)$ and $f(p_1)$.

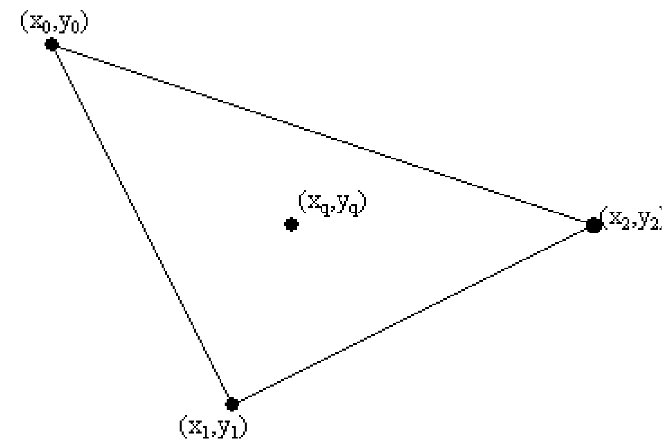
2. Find the point (x_q, y_b) between p_3 and p_2. Compute $g(x_q, y_b)$ by linear interpolation between $f(p_3)$ and $f(p_2)$.

3. Linearly interpolate between $g(x_q, y_a)$ and $g(x_q, y_b)$ to find $g(x_q, y_q)$.



Triangular Interpolation

Let (x_i, y_i, z_i) $i = 0, 1, 2$ be three points that are not collinear. These points form a triangle. Let (x_q, y_q) be a point inside the triangle. We want to compute a value z_q such that (x_q, y_q, z_q) falls on a plane that contains (x_i, y_i, z_i), $i = 0, 1, 2$.



This interpolation will work even if q is not within the triangle, but, for accuracy, we want to use a bounding triangle.

The plane is described by an equation

$$z = a_0 + a_1 x + a_2 y$$

The coefficients must satisfy the three equations that correspond to the corners of the triangle.

Triangular Interpolation

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} 1 & x_0 & y_0 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

In matrix notation we can write

$$\mathbf{z} = \mathbf{C}\mathbf{a}$$

so that

$$\mathbf{a} = \mathbf{C}^{-1}\mathbf{z}$$

The matrix $\mathbf{C}$ is nonsingular as long as the triangle corners do not fall along a line. Then the value z_q is given by

$$z_q = \begin{bmatrix} 1 & x_q & y_q \end{bmatrix} \mathbf{a} = \begin{bmatrix} 1 & x_q & y_q \end{bmatrix} \mathbf{C}^{-1}\mathbf{z}$$

Since $\mathbf{C}$ depends only upon the locations of the triangle corners, (x_i, y_i), $i = 0, 1, 2$ it can be computed once the triangles are known. This is useful in processing large batches of images.

Triangular Interpolation Coefficients

After some algebra we find the following equation for the coefficients needed to calculate z_q.

$$z_q = c_0 z_0 + c_1 z_1 + c_2 z_2$$

$$c_0 = \frac{x_2 y_1 - x_1 y_2 + x_q(y_2 - y_1) - y_q(x_2 - x_1)}{(x_1 - x_2)y_0 + (x_2 - x_0)y_1 + (x_0 - x_1)y_2}$$

$$c_1 = \frac{x_0 y_2 - x_2 y_0 + x_q(y_0 - y_2) - y_q(x_0 - x_2)}{(x_1 - x_2)y_0 + (x_2 - x_0)y_1 + (x_0 - x_1)y_2}$$

$$c_2 = \frac{x_1 y_0 - x_0 y_1 + x_q(y_1 - y_0) - y_q(x_1 - x_0)}{(x_1 - x_2)y_0 + (x_2 - x_0)y_1 + (x_0 - x_1)y_2}$$

Some simple algebra shows that

$$c_0 + c_1 + c_2 = 1$$

This is a useful computational check. It also enables the interpolation calculation to be done with 33% less multiplications.

Image Mapping A to B

If the projection from B to A is known then we can

1. Project the coordinates of the B pixels onto the A pixel grid.

2. Find the nearest A grid points for each projected B grid point.

3. Compute the value for the B pixels by interpolation of the A image.

Note that there is no projection back to B. We have computed the values for each B pixel.

If we know that the coordinates of A are integers then we can find the four pixels of A that surround (x_q, y_q) by quantizing the coordinate values.

Image Mapping Example

Two images of the same scene with seven selected matching points are shown below.

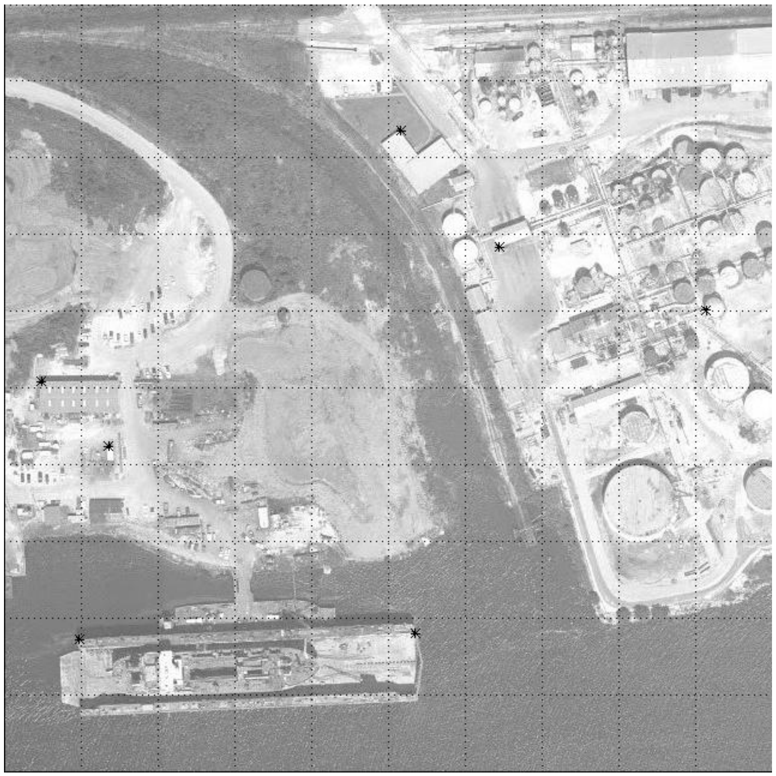


Image A

Image B

Affine Transform

The matching points have coordinates shown in the first four columns of the table below. From this we can form the homogeneous coordinate matrices $\mathbf{P}$ and $\mathbf{Q}$. The mapping $\mathbf{H}$ from A to B is

$$\mathbf{H} = \mathbf{Q}\mathbf{P}^{\dagger} = \begin{bmatrix} 0.92 & -0.39 & 224.17 \\ 0.39 & 0.92 & 10.93 \\ 0.00 & -0.00 & 1.00 \end{bmatrix}$$

The projection of the (X_a, Y_a) points by $\mathbf{H}$ is shown in the last two columns. These are close to the matching (X_b, Y_b) values.

Table of Matching Points					
X_a	Y_a	X_b	Y_b	X'_a	Y'_a
30.5	325.3	125.8	322.5	126.0	322.8
86.8	271.3	199.3	295.3	198.7	294.9
330.3	534.0	320.0	632.0	320.5	632.2
62.0	110.3	238.0	137.0	238.4	136.8
342.0	115.0	494.0	250.0	493.9	250.4
412.0	437.0	434.3	574.8	433.3	574.7
584.5	384.8	611.8	594.0	612.2	593.8

Image Transformation

The transformed A image is on the right and the original B image is on the left. The dark area on is the region of B that is not contained in A. The gray image values were computed by triangular interpolation of the gray values of A.

Mapped A

Original B

Mapping Parameters

We can determine the mapping transform $\mathbf{H}$ and the parameters of the RST matrix (Slide 11) as follows:

$$RST = \begin{bmatrix} s_0 \cos\theta & s_1 \sin\theta & s_0 x_0 \cos\theta + s_1 x_1 \sin\theta \\ -s_0 \sin\theta & s_1 \cos\theta & s_1 x_1 \cos\theta - s_0 x_0 \sin\theta \end{bmatrix}$$

$$= \begin{bmatrix} 0.92 & -0.39 & 224.17 \\ 0.39 & 0.92 & 10.93 \\ 0.00 & -0.00 & 1.00 \end{bmatrix}$$

$$s_0 = \sqrt{H_{1,1}^2 + H_{2,1}^2} = 0.999$$

$$s_1 = \sqrt{H_{1,2}^2 + H_{2,2}^2} = 1.001$$

$$\theta = -\arctan(H_{2,1}, H_{1,1}) = -23^\circ$$

$$x_0 = (H_{1,3} \cos\theta - H_{2,3} \sin\theta)/s_0 = 210.9$$

$$x_1 = (H_{1,3} \sin\theta + H_{2,3} \cos\theta)/s_1 = -77.5$$

Mapping Program

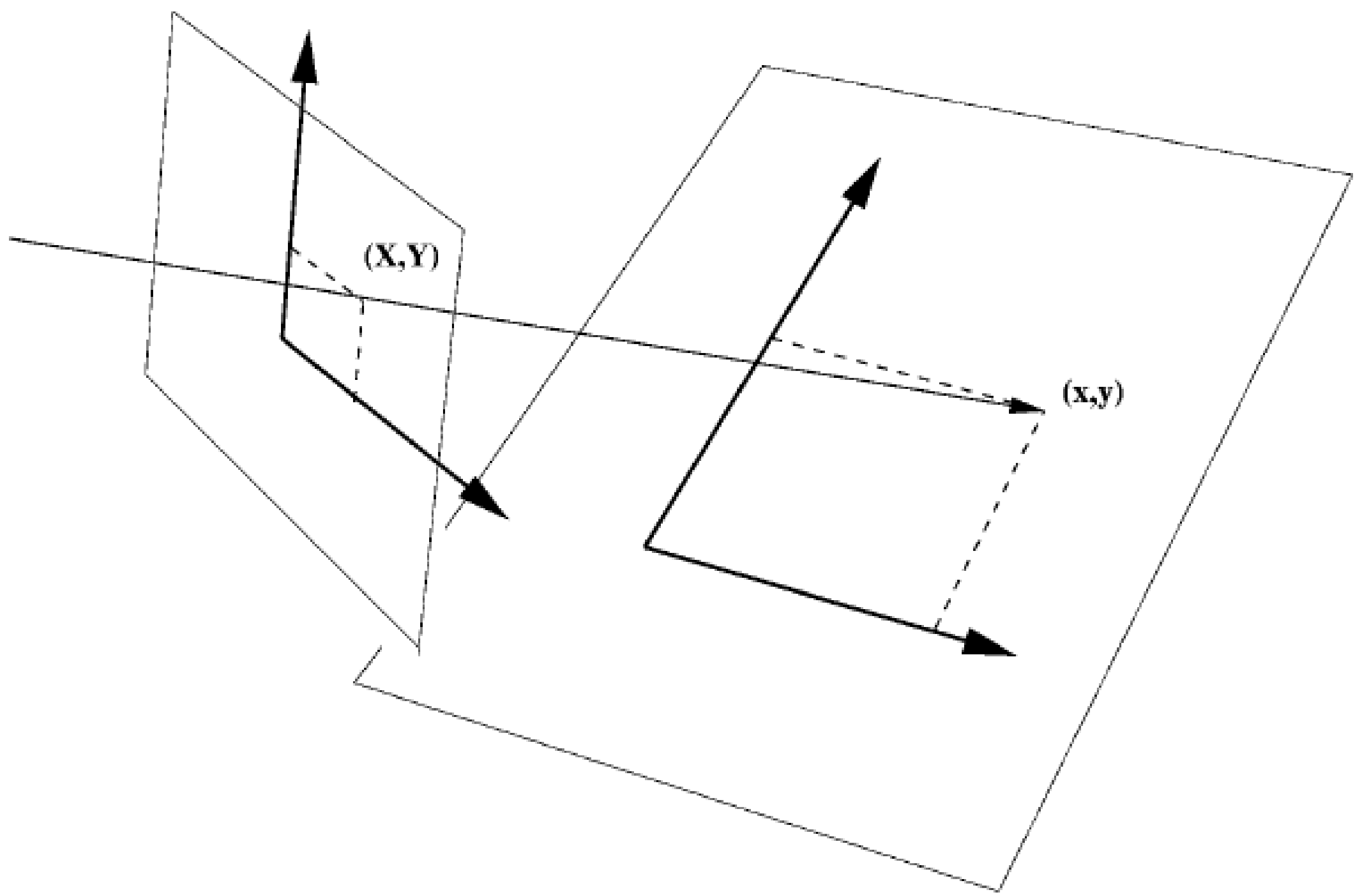
```
function affine_mapping_demo,A,T
;Map image A using the affine transformation defined by T.
;T is assumed to be in the homogeneous coordinate form.

sa=size(A,/dim)
ncols=sa[0]
nrows=sa[1]
B=fltarr(ncols,nrows)
Ti=invert(float(T)) ;Projects into the A domain

;Set up the B-domain grid to project into the A domain
grid=[[Lindgen(ncols*nrows) mod ncols], $ ; x coordinate row
      [Lindgen(ncols*nrows)/ncols], $ ; y coordinate row
      [replicate(1.,ncols*nrows)]]


q=Ti##grid ;Do the inverse projection

;Throw away the points of q that are outside of the A image grid
k=where(q[*,0] GE 0 AND q[*,0] LT ncols AND q[*,1] GE 0 AND q[*,1] LT nrows)
```

```
;Find the lower left corner of the pixel box that contains each point
xq=q[k,0]
yq=q[k,1]
x0=floor(xq)
y0=floor(yq)
kL=where(xq-x0 GE yq-y0) ;Points in the lower triangle
kU=where(xq-x0 LT yq-y0) ;Points in the upper triangle

;For the lower triangles
xL0=float(x0[kL]) & xL1=xL0+1 & xL2=xL0+1
yL0=float(y0[kL]) & yL1=yL0 & yL2=yL0+1
pL=xL0+yL0*ncols ;Location indexes
xqL=xq[kL] & yqL=yq[kL] ;Coordinates of projected grid
;Indexes of the grid points in the B image
qL=grid[k[kL],0]+grid[k[kL],1]*ncols

;Compute the triangular interpolation coefficients
d=float((xL1-xL2)*yL0 + (xL2-xL0)*yL1 + (xL0-xL1)*yL2)
c0=(xL2*yL1 - xL1*yL2 + xqL*(yL2 - yL1) - yqL*(xL2 - xL1))/d
c1=(xL0*yL2 - xL2*yL0 + xqL*(yL0 - yL2) - yqL*(xL0 - xL2))/d
c2=(xL1*yL0 - xL0*yL1 + xqL*(yL1 - yL0) - yqL*(xL1 - xL0))/d
;Interpolate and place into the image.
```

```
B[qL]=(c0*float(A[pL])+c1*float(A[pL+1])+c2*float(A[pL+1+ncols]))

;For the upper triangles
xU0=float(x0[kU]) & xU1=xU0 & xU2=xU0+1
yU0=float(y0[kU]) & yU1=yU0+1 & yU2=yU0+1
pU=xU0+yU0*ncols ;Location Indexes

xqU=xq[kU] & yqU=yq[kU] ;Coordinates of projected grid
;Indexes of grid points
qU=grid[k[kU],0]+grid[k[kU],1]*ncols

;Compute the triangular interpolation coefficients
d=float((xU1-xU2)*yU0 + (xU2-xU0)*yU1 + (xU0-xU1)*yU2)
c0=(xU2*yU1 - xU1*yU2 + xqU*(yU2 - yU1) - yqU*(xU2 - xU1))/d
c1=(xU0*yU2 - xU2*yU0 + xqU*(yU0 - yU2) - yqU*(xU0 - xU2))/d
c2=(xU1*yU0 - xU0*yU1 + xqU*(yU1 - yU0) - yqU*(xU1 - xU0))/d
;Interpolate and place into the image
B[qU]=(c0*float(A[pU])+c1*float(A[pU+ncols])+c2*float(A[pU+1+ncols]))

RETURN,B
END
```

Projective Transform

The projective transform can handle changes caused by a tilt of the image plane relative to the object plane.



Projective Transform

The perspective transformation maps (X, Y, Z) points in 3D space to (x, y) points in the image plane.

$$x_i = \frac{-fX_0}{Z_0 - f} \quad \text{and} \quad y_i = \frac{-fY_0}{Z_0 - f}$$

Suppose that A and B are images taken at different camera angles. Projection of one image plane onto the other to correct the relative tilt requires a projective transform.

$$u = \frac{ax + by + c}{gx + hy + 1} \quad \text{and} \quad v = \frac{dx + ey + f}{gx + hy + 1}$$

This eight-parameter transform maps (x, y) points in A to (u, v) points in B.

Projective Transform

The coefficients can be computed if $n \geq 4$ matching points are known in A and B. Arrange the equations as

$$ax_i + by_i + c = gx_iu_i + hy_iu_i + u_i$$
$$dx_i + ey_i + f = gx_iv_i + hy_iv_i + v_i$$

$$\begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -x_0u_0 & -y_0u_0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -x_0v_0 & -y_0v_0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1u_1 & -y_1u_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1v_1 & -y_1v_1 \\ \vdots & & & & & & \vdots & \vdots \\ x_{n-1} & y_{n-1} & 1 & 0 & 0 & 0 & -x_{n-1}u_{n-1} & -y_{n-1}u_{n-1} \\ 0 & 0 & 0 & x_{n-1} & y_{n-1} & 1 & -x_{n-1}v_{n-1} & -y_{n-1}v_{n-1} \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} u_0 \\ v_0 \\ u_1 \\ v_1 \\ \vdots \\ u_{n-1} \\ v_{n-1} \end{bmatrix}$$

The parameters can be found by multiplying both sides with the pseudo-inverse of the big matrix of coordinate terms.

Projective Transform

The inverse projective transform can be found by exchanging the (x, y) and (u, v) coordinates in the above equation. One can also solve directly for the parameters of the inverse projective transform if one knows the forward transform.

A program to compute the inversion is:

```
Function ProjectiveTransformInvert,c
;+
;d=ProjectiveTransformInvert(c) computes the coefficent vector
;d that will do the inverse projective transform defined by
;the coefficients in c.
;    c[0]u+c[1]v+c[2]          c[3]u+c[4]v+c[5]
; x=-----------------       y=----------------
;     c[6]u+c[7]v+1             c[6]u+c[7]v+1
;
;    d[0]x+d[1]y+d[2]          d[3]x+d[4]y+d[5]
; u=-----------------       v=----------------
;     d[6]x+d[7]y+1            d[6]x+d[7]y+1
;-
```

```
d=[c[4]-c[5]*c[7],$
   c[2]*c[7]-c[1],$
   c[1]*c[5]-c[2]*c[4],$
   c[5]*c[6]-c[3],$
   c[0]-c[2]*c[6],$
   c[3]*c[2]-c[0]*c[5],$
   c[3]*c[7]-c[4]*c[6],$
   c[1]*c[6]-c[0]*c[7],$
   c[0]*c[4]-c[1]*c[3]]

d=d[0:7]/d[8]
Return,d
END
```

Lens Distortion Correction

The interior orientation of a camera is the relationship of the image that is actually collected on the focal plane to an undistorted image. The distortion correction is described by a set of equations:

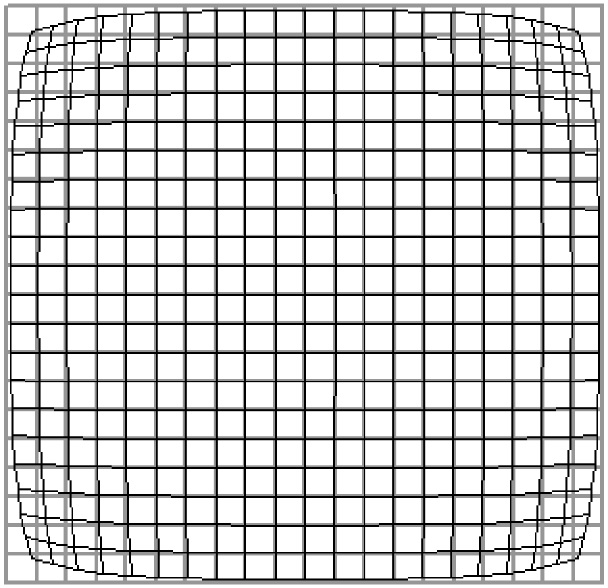
$$r^2 = (x - x_p)^2 + (y - y_p)^2$$

$$\delta_r = ((K_3 r^2 + K_2) r^2 + K_1) r^2$$

$$x_1 = x(1 + \delta_r)$$

$$y_1 = y(1 + \delta_r)$$

where (x_p, y_p) are the coordinates of the lens axis and K_1, K_2, K_3 are distortion parameters.



Effect of the radial distortion projection

Lens Distortion Correction

Direct calculation of the image on the corrected grid requires an inverse mapping of the radial distortion. The inverse mapping equations

$$x = f(u, v)$$
$$y = g(u, v)$$

can be found by

1. Construct a grid (x, y) in domain A.
2. Project the (x, y) grid into B using the available radial distortion correction equations to find matching points (u, v). These are on a distorted grid in B.
3. Use regression techniques to find the parameters of functions f and g to map (u, v) onto x and y.
4. Functions f and g can now be used to project a regular grid from B to A for interpolation.

WASP Example

The false color image on the next page contains an original LWIR image from WASP in the red layer and the distortion-corrected image in the green layer.

The correction can be seen by finding matching red and green objects. Yellow is where both layers are bright.