DIGITAL IMAGE PROCESSING AND ENHANCEMENT
Lecture 7 — Session 1, 2009
Image Warping

DIP&E-7.1 Image Warping

In this lecture, we will look at image warping.

- *Warping is often needed when an image formed one way*
- *is to be matched to an image formed in another way.*

Examples include:

- *Medical images (X-rays, CAT scans, MRI scans, ultrasound, etc.)*
- *Aerial of satellite images relative to map data*
- *Other uses include morphing (as in special effects in movies).*

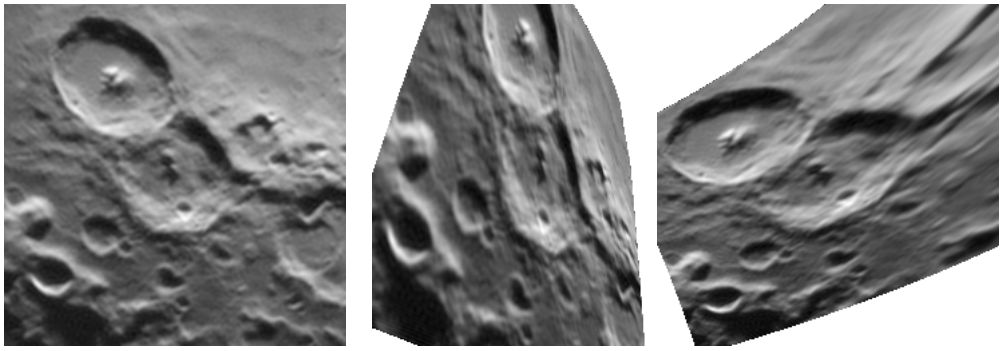
Rotation and scale changes are examples of linear warps.

Warps may also be non-linear in their mapping.

DIP&E-7.2 Image Warping

Image "warping" is a process whereby the geometry of a pixelated image is changed.

- *For example, consider the original and warped images below:*

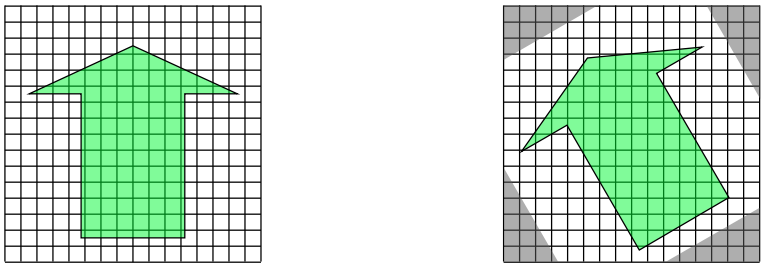


DIP&E-7.3 Image Warping

What is so special about pixelated images that we need to consider warping?

- *E.g., if a picture is crooked on the wall, it is easy to rotate it to be horizontal.*

However, a pixelated grid is not so easy to rotate, or otherwise warp; e,g,:

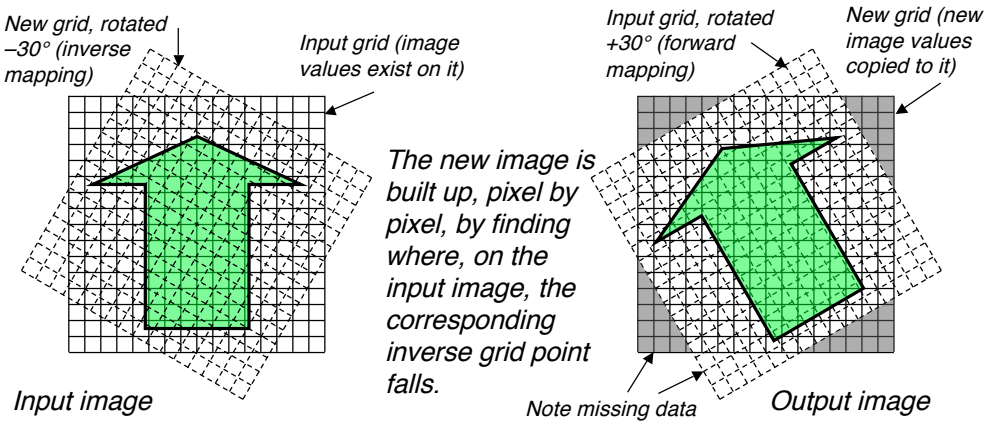


What needs to be done?

DIP&E-7.4 Image Warping

Suppose we wish to rotate the image on the left through 30° about its centre.

- *Best to work through an **inverse** rotation of an output grid through –30°, thus:*



DIP&E-7.5 Image Warping

So, the problem is one of re-gridding.

- *Re-gridding is also called re-sampling.*

This leads to the three main stages in warping:

- *Define or specify the required geometric transformation or "mapping".*
  - *e.g, rotation, scale change, some form of non-linear warp, etc.*
- *Use this definition to map new grid points over the input image grid.*
- *Resample on the (inverse) mapped grid to obtain discrete image data on the new grid.*

Why not use the forward mapping, and simply "move" input image data values to the forward mapped grid over the new grid?

DIP&E-7.6 Image Warping

For rotation, the mapping definition is simple.

- *In fact, we can write it in the well-known matrix form:*

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

which is similar to a change of axes from x_1, y_1 to x_2, y_2, after rotation through an angle θ. i.e., the matrix equation expands to:

$$x_2 = x_1 \cos\theta + y_1 \sin\theta$$
$$y_2 = -x_1 \sin\theta + y_1 \cos\theta$$

Rotation is a linear mapping, which, in general form is:

$$x_2 = a_1 + a_2 x_1 + a_3 y_1$$
$$y_2 = b_1 + b_2 x_1 + b_3 y_1$$

where the a's and b's are constants.

DIP&E-7.7 Image Warping

Linear mapping includes:

- *rotation (as we have seen),*
- *global scale change (which can be different in orthogonal directions), and*
- *shearing.*

The general linear mapping of the last slide is a first-order 2D polynomial. Higher order polynomials are useful for defining some non-linear mappings.

For example, a second order 2D polynomial mapping is:

$$x_2 = a_1 + a_2 x_1 + a_3 y_1 + a_4 x_1 y_1 + a_5 x_1^2 + a_6 y_1^2$$
$$y_2 = b_1 + b_2 x_1 + b_3 y_1 + b_4 x_1 y_1 + b_5 x_1^2 + b_6 y_1^2$$

Note that the linear 2D polynomial has three terms (for each of x an y), while the second order 2D polynomial has six terms.

DIP&E-7.8 Image Warping

We can define an n^{th} order 2D polynomial in a similar manner, which has k terms, given by $k = (n+1)(n+2)/2$.

Just as in one dimension, high order polynomials may turn out to be badly behaved —

- *So it is common to limit the order to $n \leq 3$.*
- *For a third order, n = 3, polynomial, there are k = 10 coefficients in each x and y equation.*

(In what way, and why, are high order polynomials badly behaved?)

DIP&E-7.9 Image Warping

How can we determine the mapping required in a warp?

- *Directly, by definition, such as "rotate through 30°".*
- *Indirectly, by comparing a given image with the desired result (which must be known).*

We will now discuss the second, indirect method, which uses "control points":

Two views of the same scene, taken from different angles.

DIP&E-7.10 Image Warping

Table of "Have" and "Find" coordinates obtained, e.g., by impixel in MATLAB.

Control Point	"Have" x_1	"Have" y_1	"Find" x_2	"Find" y_2
1	216	95	164	86
2	301	259	265	341
3	60	521	173	529
4	539	507	341	536
5	737	222	469	297
6	671	77	386	123

DIP&E-7.11 Image Warping

The table of measured control points is used to compute the coefficients of a 2D polynomial mapping, by solving simultaneous equations.

$$x_{21} = a_1 + a_2 x_{11} + a_3 y_{11}$$
$$x_{22} = a_1 + a_2 x_{12} + a_3 y_{12}$$
$$x_{23} = a_1 + a_2 x_{13} + a_3 y_{13}$$

and

$$y_{21} = b_1 + b_2 x_{11} + b_3 y_{11}$$
$$y_{22} = b_1 + b_2 x_{12} + b_3 y_{12}$$
$$y_{23} = b_1 + b_2 x_{13} + b_3 y_{13}$$

where the control point number is the second index in the measured coordinates.

- *(Note that, as in 1D polynomials, the number of equations required is the same as the number of unknown coefficients; three, for a linear polynomial, in this case.)*

DIP&E-7.13 Image Warping

In matrix form we can write:

$$\begin{bmatrix} x_{21} \\ x_{22} \\ x_{23} \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & y_{11} \\ 1 & x_{12} & y_{12} \\ 1 & x_{13} & y_{13} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

and

$$\begin{bmatrix} y_{21} \\ y_{22} \\ y_{23} \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & y_{11} \\ 1 & x_{12} & y_{12} \\ 1 & x_{13} & y_{13} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

or, more simply:

$$\mathbf{x_2} = \mathbf{U} \cdot \mathbf{a}$$

and $$\mathbf{y_2} = \mathbf{U} \cdot \mathbf{b}$$

where $\mathbf{x}_2$ *and* $\mathbf{y}_2$ *are vectors of the "find" control points,* **a** *and* **b** *are vectors of the coefficients, and* **U** *is a square matrix of the "have" controls, to polynomial powers.*

DIP&E-7.14 Image Warping

To solve for the coefficients a and b we simply invert the U matrix, thus:

$$\mathbf{V} = \mathbf{U}^{-1}$$

then $$\mathbf{a} = \mathbf{V} \cdot \mathbf{x}_2$$

$$\mathbf{b} = \mathbf{V} \cdot \mathbf{y}_2 \quad .$$

The method can be generalised by increasing the number of control points and size of the matrices to any power polynomial.

(Recall that the number of unknowns is $k = (n+1)(n+2)/2$ *for order n.)*

- *So, we need three control points to specify a linear mapping,*
- *six to specify a second-order mapping, and*
- *ten to specify a third-order mapping.*

**** How do we decide where to place the control points? ****

DIP&E-7.14 Image Warping

Remember that an n^{th} order 2D polynomial has k terms, given by

$$k = (n+1)(n+2)/2 .$$

- *So, we need three control points to **exactly** specify a linear mapping,*
- *six to **exactly** specify a second-order mapping, and*
- *ten to **exactly** specify a third-order mapping.*

To solve for the coefficients a and b we simply inverted a **U** *matrix:*

$$\mathbf{V} = \mathbf{U}^{-1}$$

then $\mathbf{a} = \mathbf{V} \cdot \mathbf{x}_2$

$\mathbf{b} = \mathbf{V} \cdot \mathbf{y}_2$.

DIP&E-7.15 Image Warping

The exactness of fit is only between the polynomial and the control point coordinates.

- *But, these may be in error, either due to measurement or for some other reason.*

Intuitively, it seems reasonable to assume that the more points measured, the less the chance of a small number of errors dominating the result.

For this reason, it is usual to take:

- *more than three control points for a linear mapping,*
- *more than six for a second-order mapping, and*
- *more than ten for a third-order mapping.*

Let us consider extra control points for a linear mapping.

DIP&E-7.16 Image Warping

E.g., for k = 3 (linear), let us select 5 out of our 6 control points:

Control Point	"Have" x_1	"Have" y_1	"Find" x_2	"Find" y_2
1	216	95	164	86
2	301	259	265	341
3	60	521	173	529
4	539	507	341	536
5	737	222	469	297
6	671	77	386	123

DIP&E-7.17 Image Warping

For the same matrix equation as before:

- *we now have a rectangular matrix* **U** *with the number of columns given by k, and*
- *number of rows equal to the number of control points.*

$$\mathbf{x_2} = \mathbf{U} \cdot \mathbf{a}$$

and $\mathbf{y_2} = \mathbf{U} \cdot \mathbf{b}$

$$\begin{bmatrix} x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{25} \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & y_{11} \\ 1 & x_{12} & y_{12} \\ 1 & x_{13} & y_{13} \\ 1 & x_{14} & y_{14} \\ 1 & x_{15} & y_{15} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad , \quad \textit{and}$$

$$\begin{bmatrix} y_{21} \\ y_{22} \\ y_{23} \\ y_{24} \\ y_{25} \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & y_{11} \\ 1 & x_{12} & y_{12} \\ 1 & x_{13} & y_{13} \\ 1 & x_{14} & y_{14} \\ 1 & x_{15} & y_{15} \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad .$$

DIP&E-7.18 Image Warping

Since $\mathbf{U}$ *is rectangular, we cannot form a simple inverse.*

Instead, we use the "pseudo-inverse" method of "inverting" a rectangular matrix:

$$\mathbf{V} = (\mathbf{U}^T \cdot \mathbf{U})^{-1} \cdot \mathbf{U}^T$$

from which we obtain

$$\mathbf{a} = \mathbf{V} \cdot \mathbf{x}_2$$
$$\mathbf{b} = \mathbf{V} \cdot \mathbf{y}_2$$

as before.

The difference is that the pseudo-inverse takes note of the additional information,

- *causing the polynomial to be a least-square error fit to the control points.*

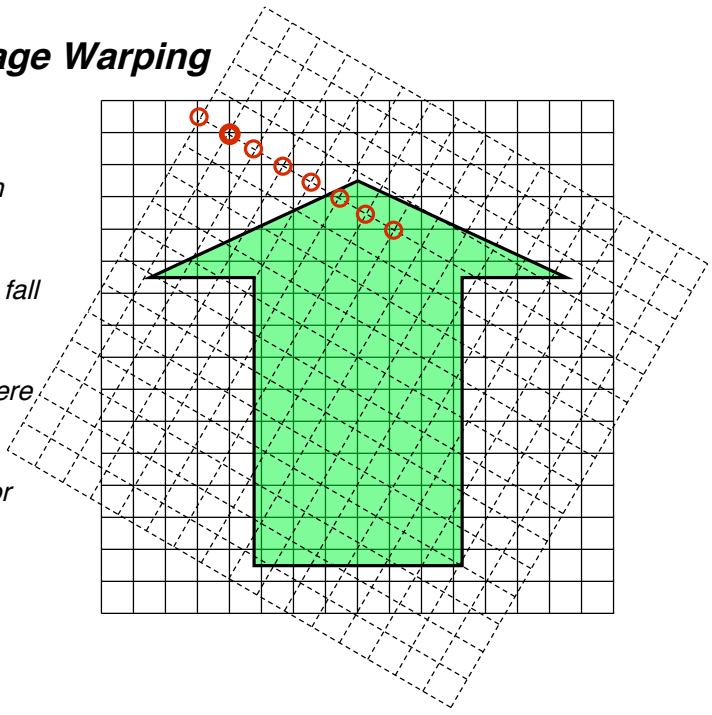
The polynomial is no longer an exact fit to the control points.

DIP&E-7.19 Image Warping

Looking at an enlarged version of a 30° rotation (inverse) mapping,

- *most points will not fall on old grid points,*
- *but will lie somewhere in between.*

What does this mean for carrying out an image warp?



DIP&E-7.20 Image Warping

The inverse mapping points back to the position in the input image space of a pixel in the output image.

- *That is, the output pixel coordinates are taken as the "have" x_1, y_1,*
- *and the position in the input image space is the "find" x_2, y_2.*

E.g., for a second order polynomial, we need a "findxy" function which computes:

$$x_2 = a_1 + a_2x_1 + a_3y_1 + a_4x_1y_1 + a_5x_1^2 + a_6y_1^2$$
$$y_2 = b_1 + b_2x_1 + b_3y_1 + b_4x_1y_1 + b_5x_1^2 + b_6y_1^2 .$$

Even though x_1, y_1 are integer (i.e., output pixel coordinate indices),

- *the found input space coordinates, x_2, y_2, are likely to be fractional.*

DIP&E-7.21 Image Warping

The simplest solution, if x_2, y_2, are real (non-integer) is to "round" them to the nearest integers.

- *This process is called "nearest neighbour" interpolation.*
- *E.g., MATLAB code might look like:*

```
for y1 = 1:ny
    for x1 = 1: nx
        x2 = round(findx(y1, x1));
        y2 = round(findy(y1, x1));
        image_out(y1, x1) = image_in(y2, x2);
    end
end
```

Before obtaining a pixel value so referenced, what else must be checked?

DIP&E-7.22 Image Warping

While nearest neighbour "interpolation" is fast, it is also crude, leading to visual and measurement errors at the fine scale of pixel separation.

- *E.g., if an image is rotated again and again by this method, say nine times 40° each time (9 x 40 = 360), the result will be quite badly degraded.*

Question: what are we really trying to do?

- *Interpolation means obtaining a (meaningful) value in the spaces between given values.*
- *We are really defining a "continuous intensity surface", that "clothes" the discrete data set which is our (input) image.*

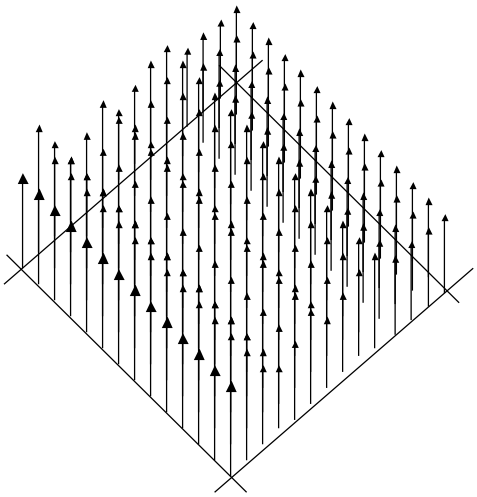
DIP&E-7.23 Image Warping

Image intensity surface.

- *We wish to "clothe" the Fakir's "bed-of-nails":*
- *The discrete (pixel) data is simply a code for a continuous intensity surface.*

Nearest neighbour interpolation makes a "city block" surface:

- *The surface is assumed like the flat rooves of city buildings, abutting one another centred on the pixel sampling points.*

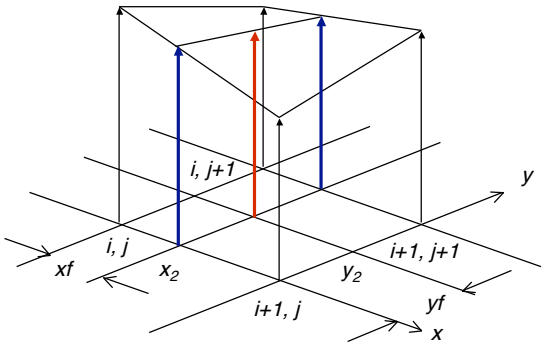


DIP&E-7.24 Image Warping

The next, obvious form of interpolation is linear interpolation.

- *For a 2D geometry, this is known as "bilinear" interpolation (linear in x and y).*

The geometry is:



What sort of image intensity surface is this?

DIP&E-7.25 Image Warping

For bilinear interpolation, we can write the following MATLAB code:

```
for y1 = 1:ny
    for x1 = 1: nx
        x2 = findx(y1, x1);
        y2 = findy(y1, x1);
        i = trunc(x2);
        j = trunc(y2);
        xf = x – i;
        yf = y – j;
        value_lo = (1–xf) . image_in(i, j) + xf . image_in(i+1, j);
        value_hi = (1–xf) . image_in(i, j+1) + xf . image_in(i+1, j+1);
        value = (1–yf) . value_lo + yf . value_hi;
        image_out(y1, x1) = (1–yf) . value_lo + yf . value_hi;
    end
end
```

Note that xf and yf are the fractional offsets from (truncated) x2 and y2 indices.

DIP&E-7.26 Image Warping

Another, popular method in image warping is so-called "Cubic Convolution" interpolation.

In this method, the surrounding 16 grid points are used, with a weighting function depending on the distance of the x2, y2 point from each of the 16 grid points.

The weight is:

$$r(xf) = \begin{cases} (\alpha+2)|xf|^3 - (\alpha+3)|xf|^2 + 1, & |xf| < 1 \\ \alpha|xf|^3 - 5\alpha|xf|^2 + 8\alpha|xf| - 4\alpha, & 1 \le |xf| < 2 \\ 0, & \text{otherwise} \end{cases}$$

where xf is the distance of x2 from each grid point, in the x direction (similarly, we could compute weights in the y direction); and α is a parameter which controls the detailed shape of the function.

Typically, $\alpha = -1$ (traditional), $\alpha = -0.5$ (better), $\alpha = 0$ (approximately linear).

DIP&E-7.27 Image Warping

Is there an "ideal" interpolator?

If we consider the sampling theorem, which says that we can reconstruct an original, continuous function from its sampled data, if:

- *the sampling frequency is more than twice the highest frequency present in the data,*
- *then, we have a clue to an ideal method.*

We can call this a (near) perfect low-pass filter method.

- *In fact, the cubic convolution interpolator tries to approximate to this ideal;*

  but turns out to be far from ideal.

(The shape of r(xf) is meant to approximate the lobes of a sinc(fx) function.)

DIP&E-7.28 Image Warping

If a sinc(xf) function is convolved right across the image (i.e., large kernel convolution), a near perfect low pass filter interpolation would result.

- *Very expensive in computation time!*

Instead, we can use the 2D DFT (FFT) algorithm, as follows:

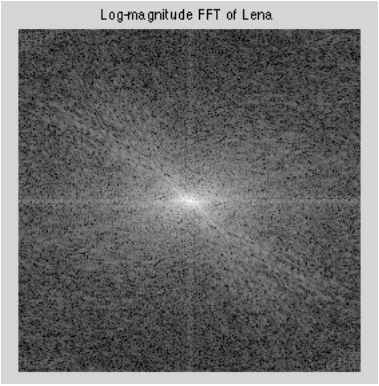
- *Fourier transform the input image;*
- *Use the (shifted) Fourier transform as the central part of a large, zeroed frequency domain image (perhaps 4 times each dimension);*
- *Inverse Fourier transform the result (after shifting zero frequency to origin);*
- *The result is a near perfect low-pass filtered interpolated version of the original, with sixteen times as many pixels.*

Finally, use linear interpolation between the much close pixels of the big image.

DIP&E-7.29 Image Warping

Example of a near-perfect interpolation by FFT expansion.

Begin with the FFT of an image:



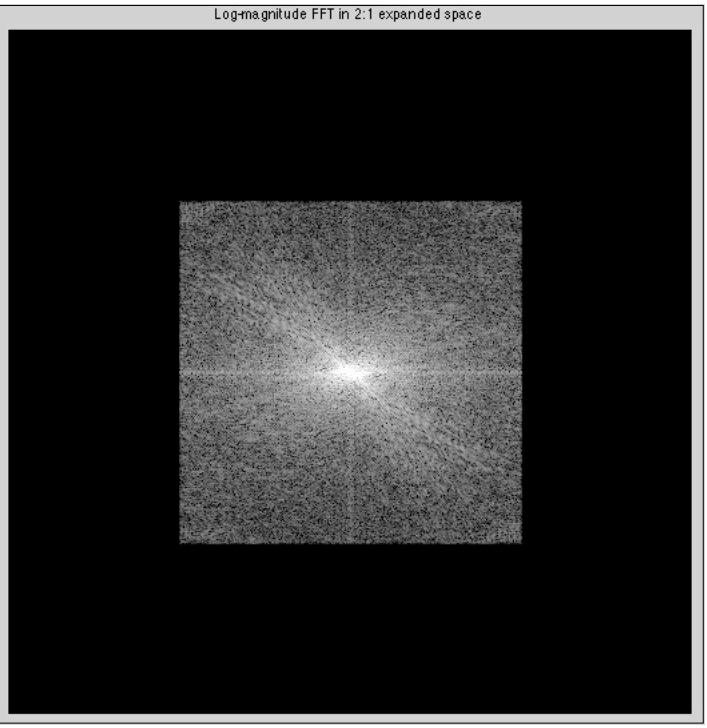
DIP&E-7.30

Image Warping

Near-perfect interpolation by FFT expansion:

Take the FFT of the original

"Drop" it in to an enlarged zero image space (e.g., 2:1 here).



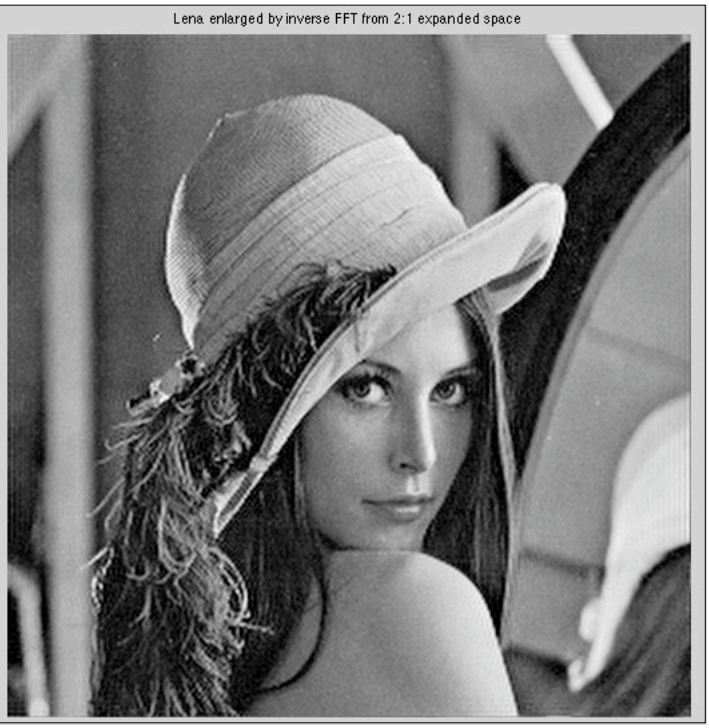
DIP&E-7.31

Image Warping

Near-perfect interpolation by FFT expansion:

Inverse FFT the enlarged Fourier space.

Obtain near-perfect low-pass filtered enlargement (2:1 in this case).



DIP&E-7.32 Image Warping

How can the near perfect expansion interpolation be used in general warping?

- *First, expand the source image (2x, 4x, 8x, etc.) by the FFT*
- *Then, use the expanded image as the new source in a general warp*
- *To do so, the "find" functions must return expanded coordinates*
- *e.g, multiply the x2, y2 by 2, 4, or 8, etc.*

In doing so, the found points will be much closer to perfectly interpolated values.

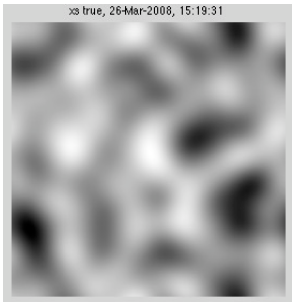
- *Thus, bilinear interpolation, say, does not have to "work so hard" to obtain an accurate value for the grey level.*

DIP&E-7.33 Image Warping

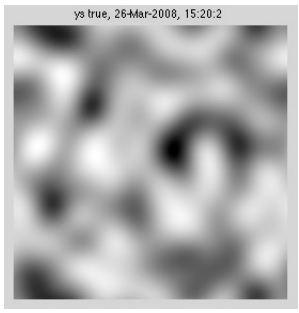
Finally, note that the warp mapping function does not have to be an analytical expression.

- *Instead, it might be "stored" as a discrete function in a look-up table.*
- *e.g., if we have two image matrices the same size as the image to be warped,*

Then the findx and findy "functions" become those matrices.



findx image matrix

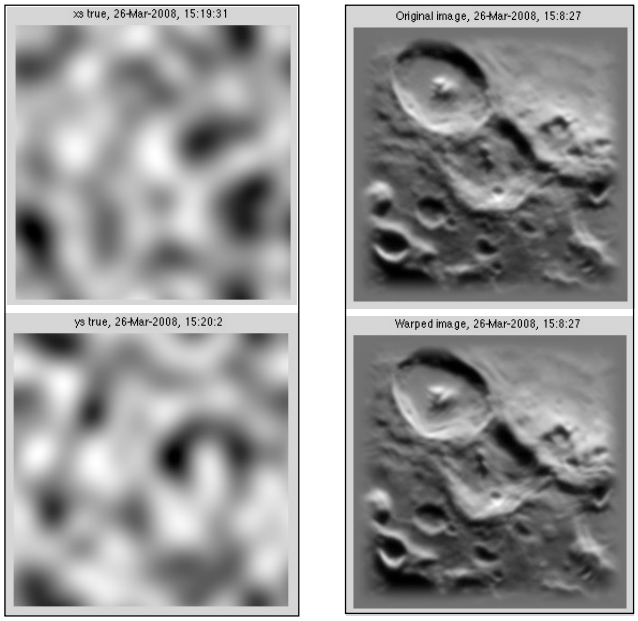


findy image matrix

DIP&E-7.34 Image Warping

Using the findx and findy matrices,

the original lunar image is warped as shown.



DIP&E-7.35 Image Warping

The previous warping was done as a simulation

To test an automatic registration algorithm.

The right-hand shift images are derived from the original lunar and its warped version.