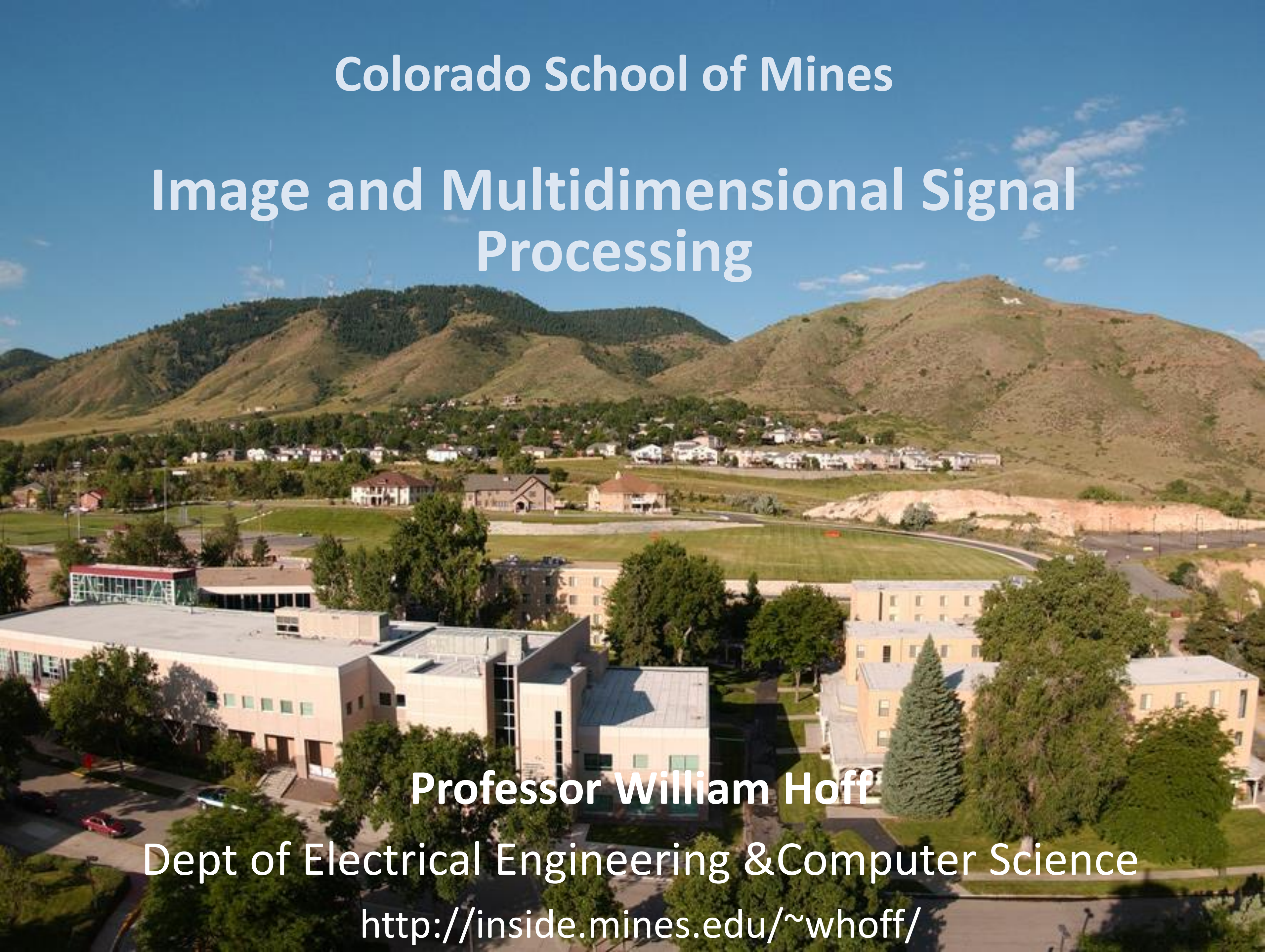


Colorado School of Mines

Image and Multidimensional Signal Processing

Professor William Hoff

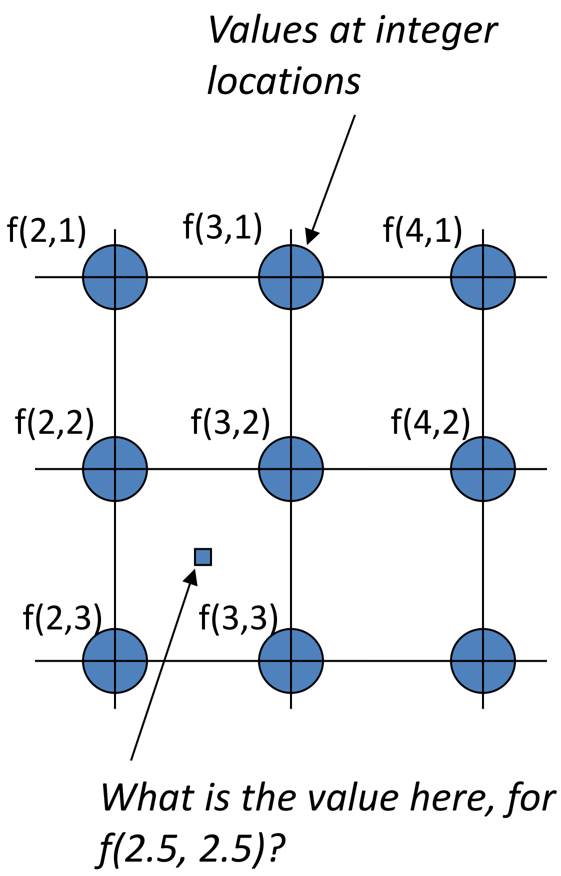
Dept of Electrical Engineering &Computer Science

http://inside.mines.edu/~whoff/

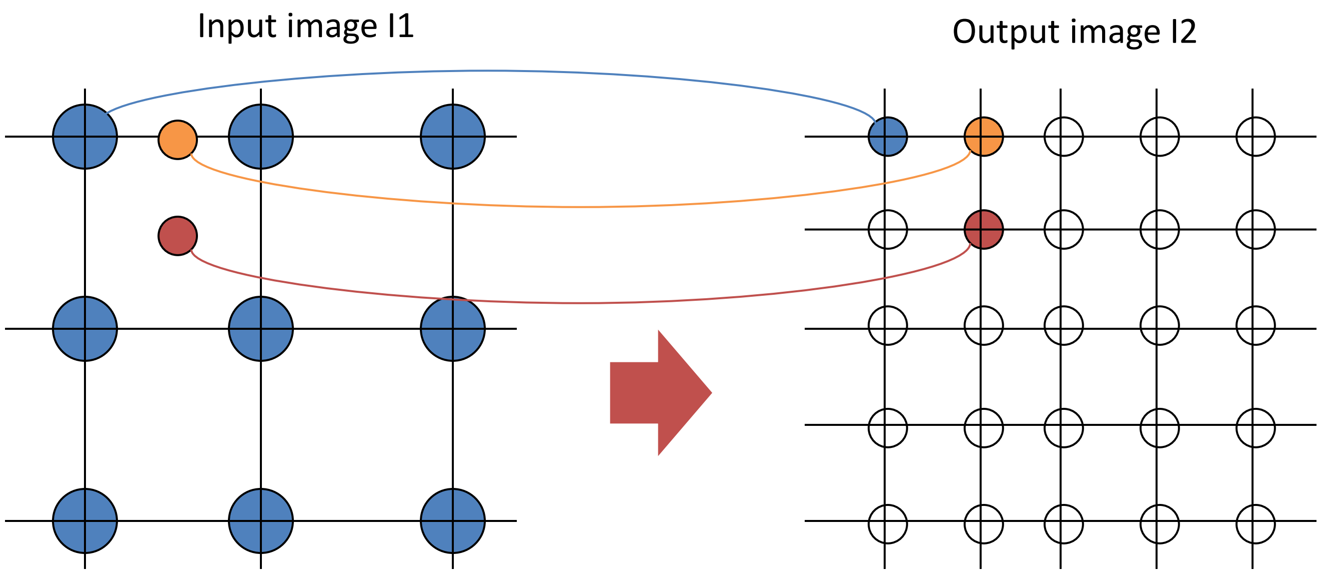
Interpolation and Spatial Transformations

Image Interpolation (Sect 2.4.4)

- We often need to estimate (interpolate) the value of an image at a non-integer location
- Example: expanding a 500x500 image to 1000x1000 – pixel spacing is different
- We can do this by looking at the values nearby. Methods include:
  - Nearest neighbor – just take the value of the closest neighbor
  - Bilinear – take a combination of the four closest neighbors
  - Bicubic – use the closest 16 neighbors (most computationally expensive, but best results)



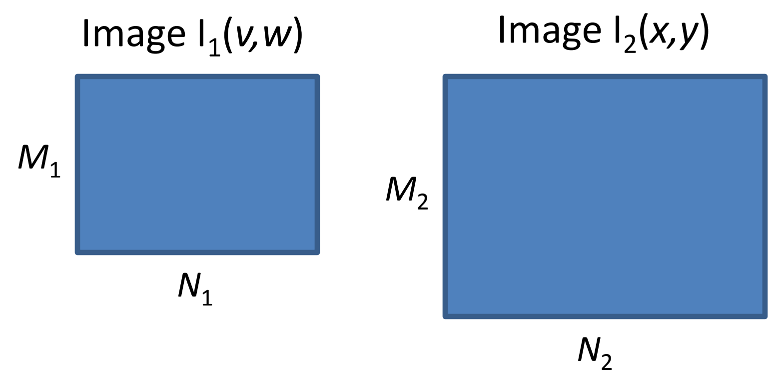
Application: Resizing an Image



- We need to assign values to each pixel in the output image
- So scan through the output image; at each pixel calculate the value from the input image at the corresponding location
- If not at an integer position in the input image, we need to interpolate

Mapping function

- Assume
  - Image I_1 has size $M_1 x N_1$
  - Image I_2 has size $M_2 x N_2$



- Then

  $x = c_x v$

  $y = c_y w$

  - where cx,cy are the scaling factors:

    $c_x = N2/N1$

    $c_y = M2/N1$

- We actually want the inverse function
  - $v = x/c_x$
  - $w = y/c_y$

Matlab example – Nearest Neighbor Interpolation

```
clear all
close all

I1 = imread('cameraman.tif');
M1 = size(I1,1);       % Number of rows in I
N1 = size(I1,2);       % Number of columns in I

% Pick size of output image
M2 = 300;
N2 = 100;
I2 = zeros(M2,N2);  % Allocate output image

cx = N2/N1;      % Scale in x
cy = M2/M1;      % Scale in y
for x=1:N2
    for y=1:M2
        % Calculate position in input image
        v = x/cx;
        w = y/cy;

        % We'll just pick the nearest neighbor to (v,w)
        v = round(v);
        w = round(w);

        I2(y,x) = I1(w,v);
    end
end
```

Bilinear interpolation

- The value at (*x*,*y*) is

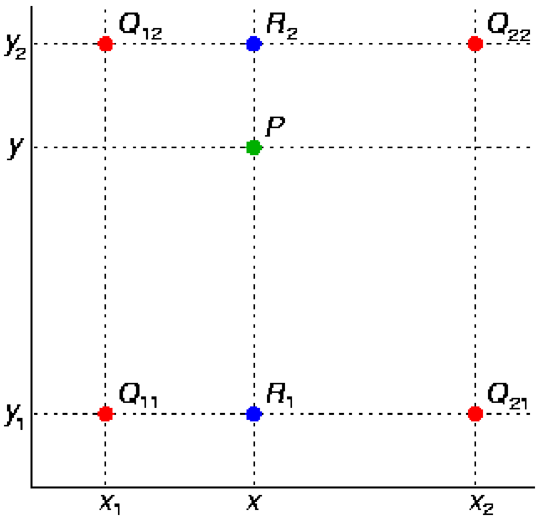
  $$f(x,y) = ax + by + cxy + d$$

  where *a*,*b*,*c*,*d* are coefficients determined by the four closest neighbors
- Equivalent to doing linear interpolation in one dimension, then the other
- To find the value at P, if we know values at the Q's
  - First interpolate horizontally to find values at R1,R2

    $$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

    $$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$
  - Then interpolate vertically

    $$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2).$$



- If corners are at (0,0), (0,1), (1,0), (1,1)

  $$f(x,y) \approx f(0,0)\,(1-x)(1-y) + f(1,0)\,x(1-y) + f(0,1)\,(1-x)y + f(1,1)xy.$$

http://en.wikipedia.org/wiki/Bilinear_interpolation

Geometric Transformations (2.6.5)

- Example applications
  - Rotation
  - De-warp to undo lens distortion
  - Register two images (e.g., satellite images of the same patch of ground)
  - Also called “rubber sheet” transformations

- A transformation is a function that maps pixel coordinates

$$(x,y) = T(v,w)$$

  - where
    - (v,w) are pixel coordinates in the original image
    - (x,y) are the corresponding pixel coordinates in the transformed image
  - Example: $T(x,y) = (v/2, w/2)$ shrinks the image by a factor of 2

- Remember that we actually need $(v,w) = T^{-1}(x,y)$

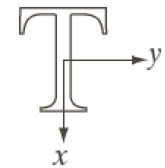
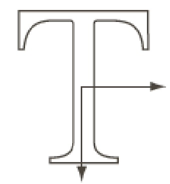
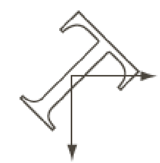
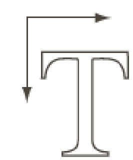
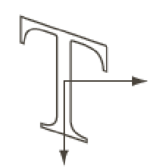
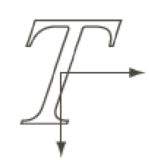
Matrix Representation

- We can often represent the transformation T as a matrix
- It is convenient to append a "1" to the pixel coordinates

- Example: A common transformation is the "affine transform"

$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} u & v & 1 \end{bmatrix} \mathbf{T} = \begin{bmatrix} u & v & 1 \end{bmatrix} \begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$

TABLE 2.2
Affine transformations based on Eq. (2.6.–23).

Transformation Name	Affine Matrix, T	Coordinate Equations	Example
Identity	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = w$	
Scaling	$\begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = c_x v$ $y = c_y w$	
Rotation	$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v\cos\theta - w\sin\theta$ $y = v\cos\theta + w\sin\theta$	
Translation	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$	$x = v + t_x$ $y = w + t_y$	
Shear (vertical)	$\begin{bmatrix} 1 & 0 & 0 \\ s_v & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v + s_v w$ $y = w$	
Shear (horizontal)	$\begin{bmatrix} 1 & s_h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$x = v$ $y = s_h v + w$	

Estimating parameters of the transformation

- If you want to register one image to another, you need to estimate the parameters of the transformation function
- You can do this if you know the correspondences for a few control points ("tiepoints") in each image
  - I.e., (x_1,y_1) corresponds to (v_1,w_1), (x_2,y_2) corresponds to (v_2,w_2), etc

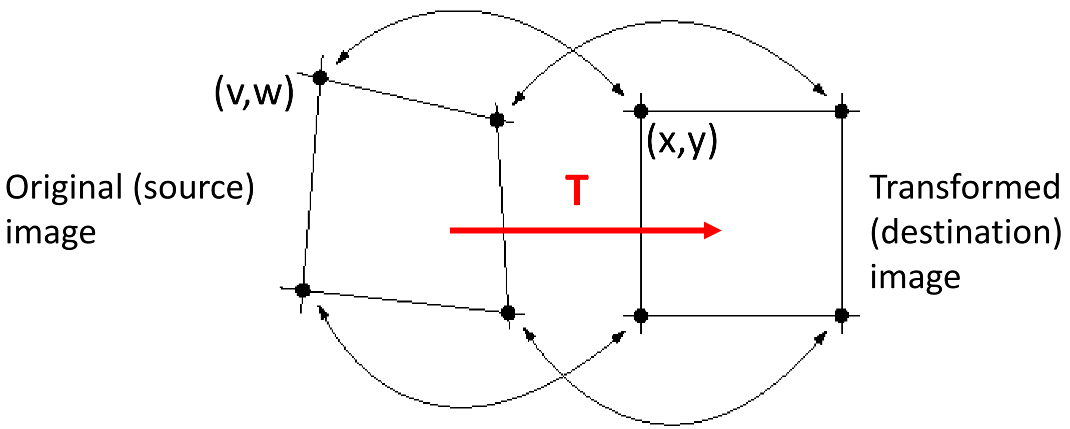


FIGURE 5.32 Corresponding tiepoints in two image segments.

- For the affine transformation: $$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} v & w & 1 \end{bmatrix} \mathbf{T} = \begin{bmatrix} v & w & 1 \end{bmatrix} \begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$
  - There are six unknowns – we need at least three pairs of tiepoints (each yields two equations)

Geometric Transformations

- Recall that we actually need the inverse mapping, from transformed image to source image

  $(v,w) = T^{-1}(x,y)$

- This is because what we really need is the values of the pixels at the integer locations in the transformed image

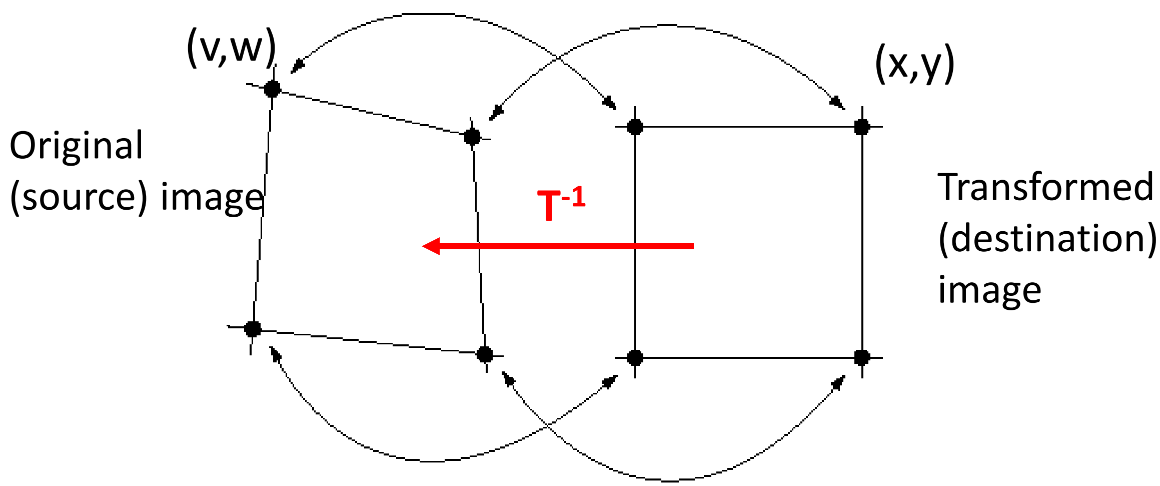


FIGURE 5.32 Corresponding tiepoints in two image segments.

- We need to do gray level interpolation to find the values in the original image

Estimating parameters of the transformation

- If we have three tie points

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} = \begin{bmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ u_3 & v_3 & 1 \end{bmatrix} \mathbf{T} \quad \text{or} \quad \mathbf{X} = \mathbf{UT}. \quad \text{Then } \mathbf{T} = \mathbf{U}^{-1}\mathbf{X}$$

- If we have more than three tie points (more is better for accuracy)

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ & \vdots & \\ x_n & y_n & 1 \end{bmatrix} = \begin{bmatrix} u_1 & v_1 & 1 \\ u_2 & v_2 & 1 \\ u_3 & v_3 & 1 \\ & \vdots & \\ u_n & v_n & 1 \end{bmatrix} \mathbf{T}$$

$$\mathbf{X} = \mathbf{UT}$$

$$\mathbf{U}^{\mathbf{T}}\mathbf{X} = \mathbf{U}^{\mathbf{T}}\mathbf{UT}$$

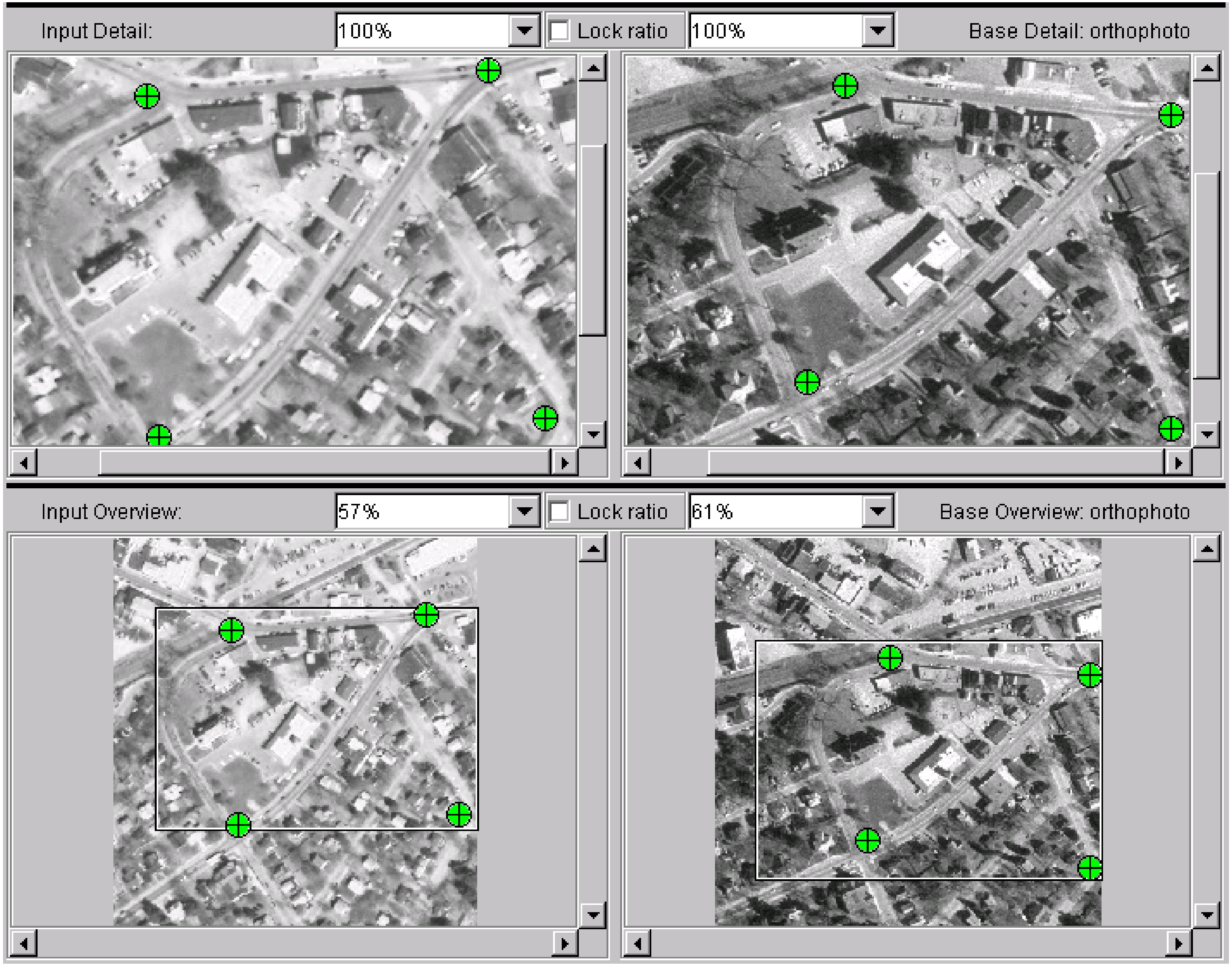
$$\left(\mathbf{U}^{\mathbf{T}}\mathbf{U}\right)^{-1}\mathbf{U}^{\mathbf{T}}\mathbf{X} = \mathbf{T}$$

Example from Matlab

- We want to transform the new image on the left to register (align) it to the old image on the right (to compare them for changes, for example)
- See Image Processing Toolbox>Examples>Image registration

westconcordaerial.png

westconcordorthophoto.png



See Matlab functions cpselect, cp2tform, imtransform

Registered Image

Orthophoto Image

```
diff = imabsdiff(registeredOriginal,
registered(:,:,1));
```

Can look at the 3x3 affine transformation matrix

Summary / Questions

- To do a spatial transformation, we need to (a) specify a function that maps points from one image to the other, and (b) interpolate intensity values at non-integer positions.
- Affine transformations can perform rotation, translation, scaling, and shear.
- When performing the transformation to generate the new image, why is the inverse mapping useful?