Chapter 6

Image Geometry

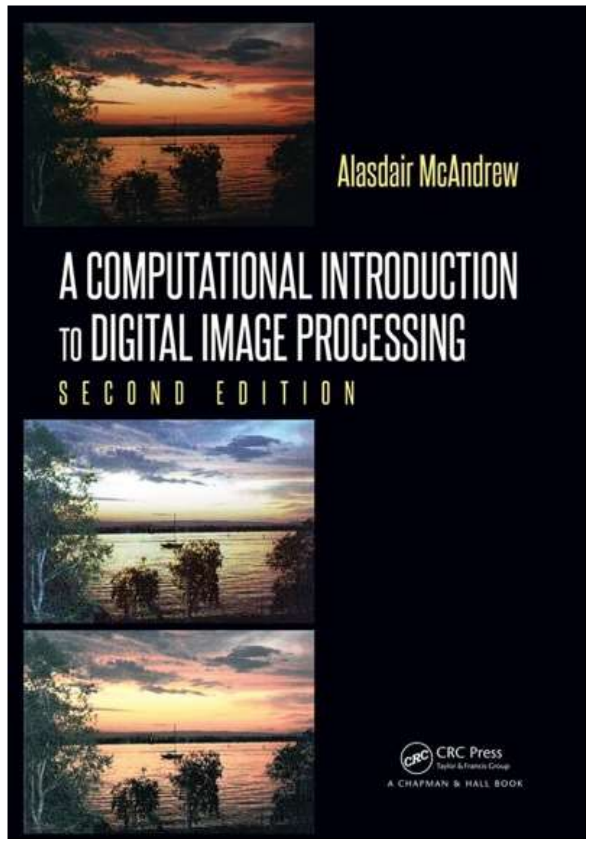


Image Processing Operations

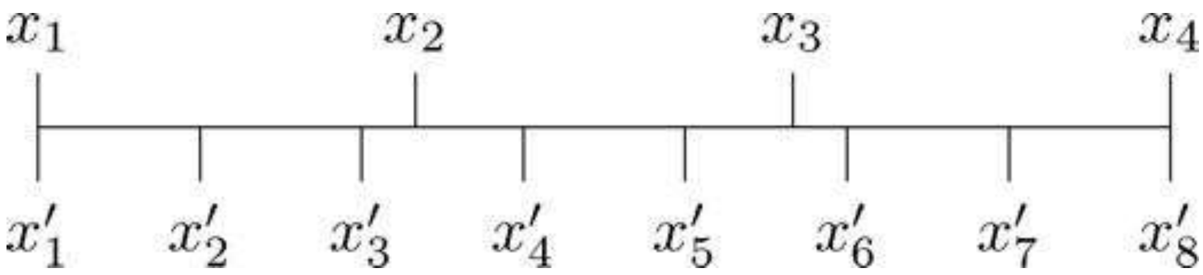
- 6.1 Interpolation of Data
- 6.2 Image Interpolation
- 6.3 General Interpolation
- 6.4 Enlargement by Spatial Filtering
- 6.5 Scaling Smaller
- 6.6 Rotation
- 6.7 Correcting Image Distortion

Interpolation of Data

Simple Problem

- Have 4 values
- Want to enlarge to 8

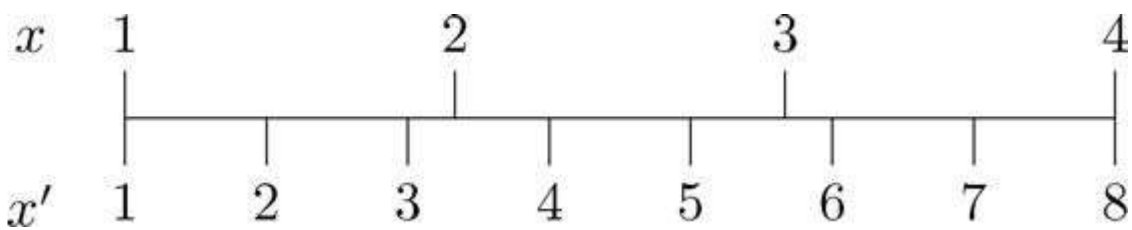
Figure 6.1: Replacing four points with eight



Interpolation of Data

MIDWESTERN STATE UNIVERSITY

Figure 6.2: Figure 6.1 slightly redrawn



Suppose that the distance between each of the x_i points is 1; thus, the length of the line is 3. Thus, since there are seven increments x_1 from to x_8, the distance between each two will be 3/7 ≈ 0.4286.

Formulas can be derived from finding eq. of a line given 2 points (4, 8) & (1, 1) going from *x* to *x*′; (8, 4) & (1, 1) going from *x*′ to *x*

$$x' = \frac{1}{3}(7x - 4),$$

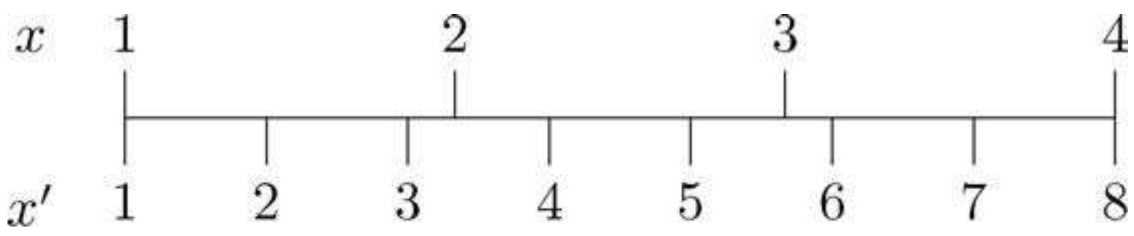
$$x = \frac{1}{7}(3x' + 4).$$

What value for x do you get if you plug in 4 for x'?

$$2\frac{2}{7}$$

Interpolation of Data

Figure 6.2: Figure 6.1 slightly redrawn



MatLab/Octave:

linspace(X1, X2, N) generates N points between X1 and X2.
For N = 1, linspace returns X2.

```
>> x2 = linspace(1,4,8)

x2 =

   1.0000   1.4286   1.8571   2.2857   2.7143   3.1429   3.5714   4.0000
```

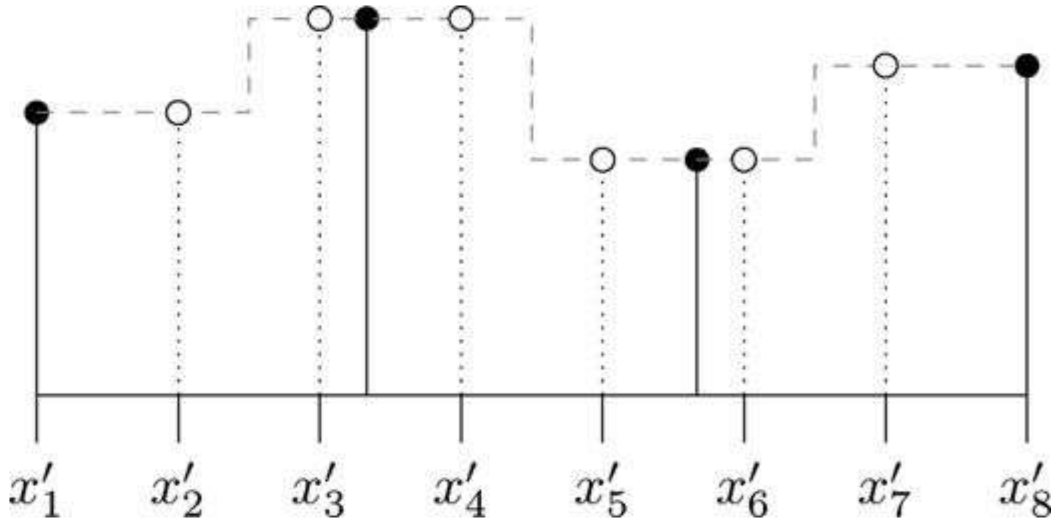
Nearest Neighbor Interpolation

- Interpolation: guessing at function
- Nearest neighbor interpolation: assign $f(x_i') = f(x_j)$ where x_j is the original point closest to x_i'

Figure 6.3: Nearest neighbor interpolation

Closed circles = original values

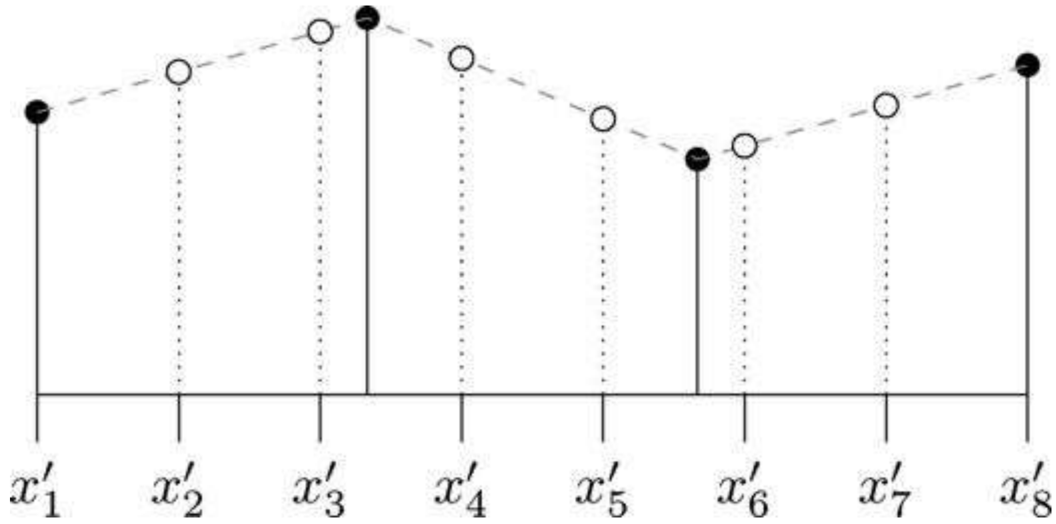
Open circles = interpolated values



Linear Interpolation

Linear interpolation: join the original function values by straight lines, and take interpolated values as the values at those lines.

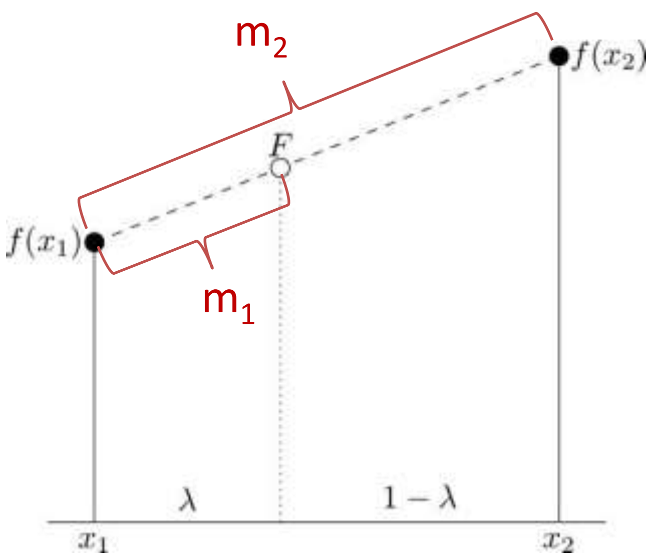
Figure 6.4: Linear interpolation



Linear Interpolation

To calculate values

- By considering slopes: $\overset{m_1}{\dfrac{F-f(x_1)}{\lambda}} = \overset{m_2}{\dfrac{f(x_2)-f(x_1)}{1}}$
- Solve for F: $F = \lambda f(x_2) + (1-\lambda) f(x_1)$



Linear Interpolation

$$F = \lambda f(x_3) + (1 - \lambda) f(x_2)$$

Example:

$f(x_1) = 2,$ $f(x_2) = 3,$
$f(x_3) = 1.5,$ $f(x_4) = 2.5$

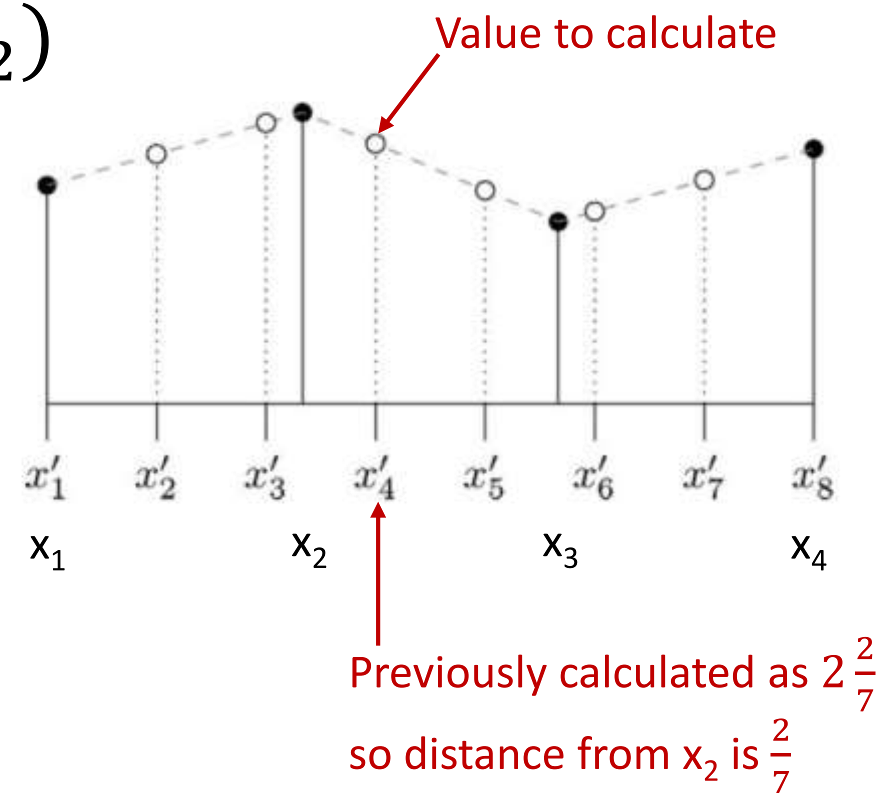
Consider x'_4

Between x_2 and x_3

Corresponding value for λ is $^2/_7$

Thus:

$$f(x'_4) = \frac{2}{7}(1.5) + \frac{5}{7}(3) \approx 2.5714$$

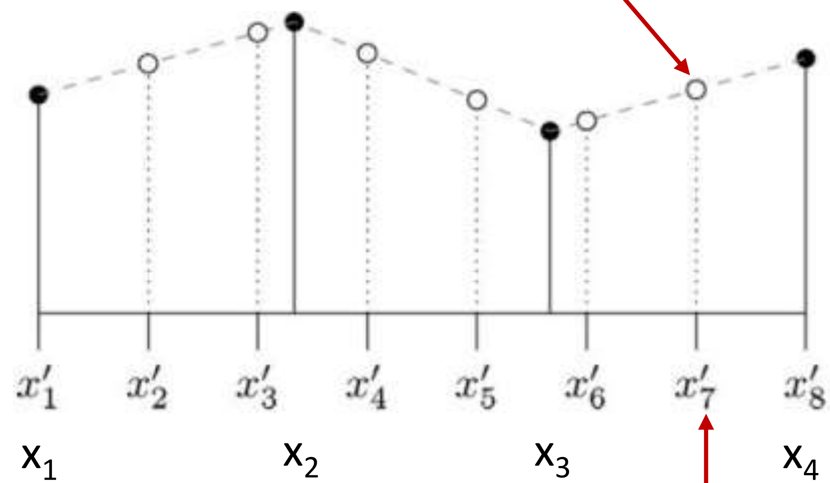


Linear Interpolation

MIDWESTERN STATE UNIVERSITY

$$F = \lambda f(x_4) + (1 - \lambda) f(x_3)$$

Value to calculate



Previously calculated as $3\frac{4}{7}$

so distance from x_3 is $\frac{4}{7}$

Example:

$f(x_1) = 2,\qquad f(x_2) = 3,$

$f(x_3) = 1.5,\qquad f(x_4) = 2.5$

Consider x'_7

Between x_3 and x_4

Corresponding value for λ is $^4/_7$

Thus:

$$f(x'_7) = \frac{4}{7}(2.5) + \frac{3}{7}(1.5) \approx 2.0714$$

Interpolation Example

MatLab/Octave Example:

```
>> orig = [3 5 8 4 2];
>> sp = linspace(1,5,7)

sp =

   1.0000    1.6667    2.3333    3.0000    3.6667    4.3333    5.0000

>> new = uint8(interp1(orig, sp, 'nearest'))

new =

   3    5    5    8    4    4    2

>> new = uint8(interp1(orig, sp, 'linear'))

new =

   3    4    6    8    5    3    2
```

Nearest: Notice that all numbers are the same as original, but spread out so there are 7 instead of 5.

Linear: Notice that the 7 numbers are NOT all the same as original 5.

MatLab/Octave Example:

```
           1 2 3 4 5
>> orig = [3 5 8 4 2];
>> sp = linspace(1,5,7)

sp =

   1.0000   1.6667   2.3333   3.0000   3.6667   4.3333   5.0000

>> new = uint8(interp1(orig, sp, 'linear'))

new =

   3   4   6   8   5   3   2
```

$= round(.3333 * 8 + (1 - .3333) * 5)$

- By considering slopes: $\frac{F-f(x_1)}{\lambda} = \frac{f(x_2)-f(x_1)}{1}$
- Solve for F: $F = \lambda f(x_2) + (1-\lambda) f(x_1)$

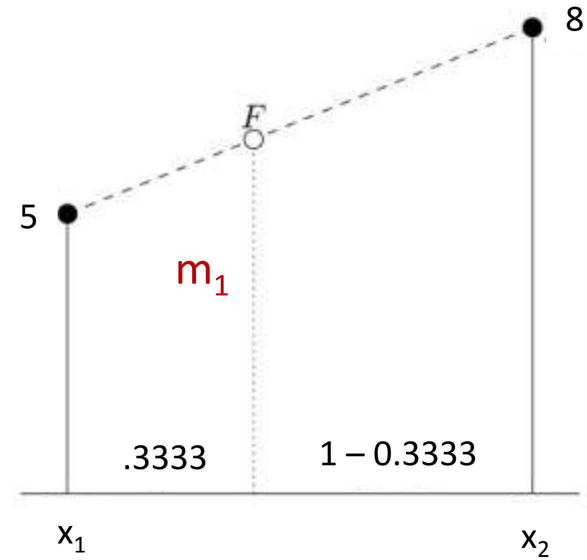


Image Interpolation

Figure 6.6: Interpolation on an image: Large open circles are original points, filled circles are interpolated

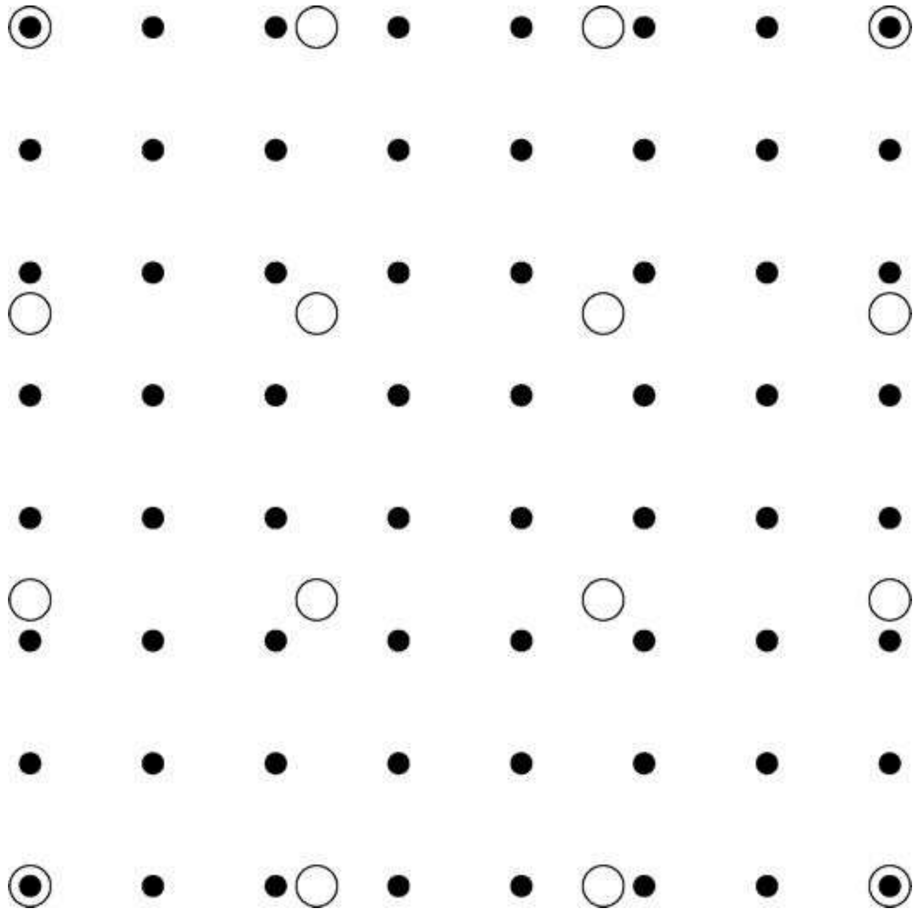


Image Interpolation

Figure 6.7: Interpolation between four image points

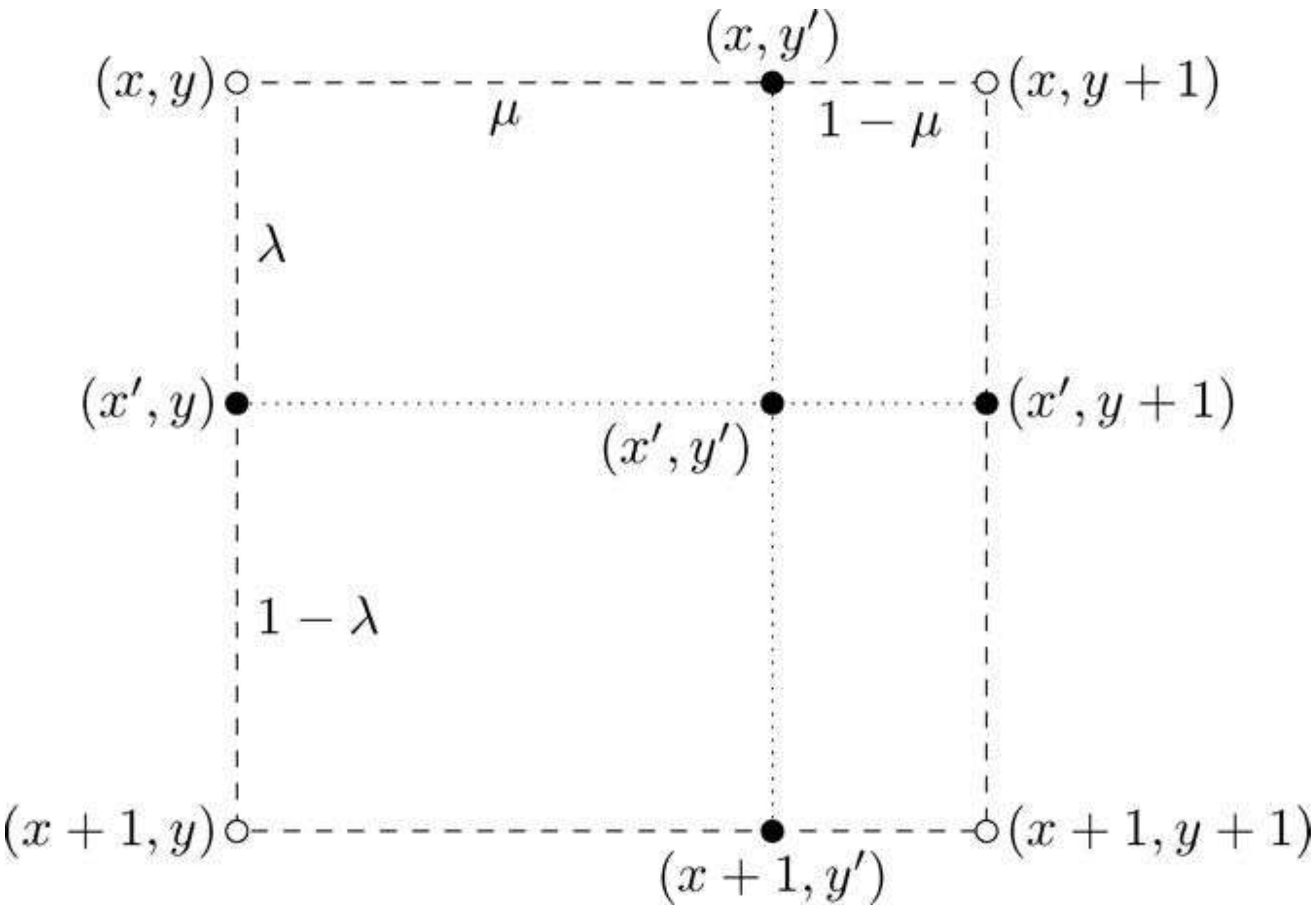


Image Interpolation

Equation 6.1

- $f(x, y') = \mu f(x, y+1) + (1-\mu) f(x, y)$

  and

- $f(x+1, y') = \mu f(x+1, y+1) + (1-\mu) f(x+1, y)$

  along the y' column

- $f(x', y') = \lambda f(x+1, y') + (1-\lambda) f(x, y')$

and substituting in the values just obtained produces

$$\begin{aligned} f(x', y') &= \lambda(\mu f(x+1, y+1) + (1-\mu) f(x+1, y)) + (1-\lambda)(\mu f(x, y+1) \\ &\quad + (1-\mu) f(x, y)) \\ &= \lambda\mu f(x+1, y+1) + \lambda(1-\mu) f(x+1, y) + (1-\lambda)\mu f(x, y+1) \\ &\quad + (1-\lambda)(1-\mu) f(x, y) \end{aligned}$$

This last equation is the formula for *bilinear interpolation*.

Image Interpolation

- Nearest neighbor gives blocky effect…YUCK!
- Bilinear interpolation is smoother…but blurry ☹
- Interpolation CAN'T predict values
- CAN'T create something from NOTHING!

Try It:

```
>> c = imread('cameraman.png');
>> head = c(33:96,90:153);
>> imshow(head)
>> head4n = imresize(head,4,'nearest');imshow(head4n)
>> head4b = imresize(head,4,'bilinear');imshow(head4b)
```

MATLAB/Octave

Python has a `rescale` function in the `transform` module of `skimage`:

```
In :  c = io.imread('cameraman.png')
In :  head = c[32:96,89:153]
In :  io.imshow(head)
In :  head4n = tr.rescale(head,2,order=0)     #order=0, nearest neighbor
In :  head4b = tr.rescale(head,2,order=1)     #order=1, bilinear interpolation
```

Python

General Interpolation

Figure 6.9: Scaling by interpolation

(a) Nearest neighbor scaling

(b) Bilinear interpolation

General Approach

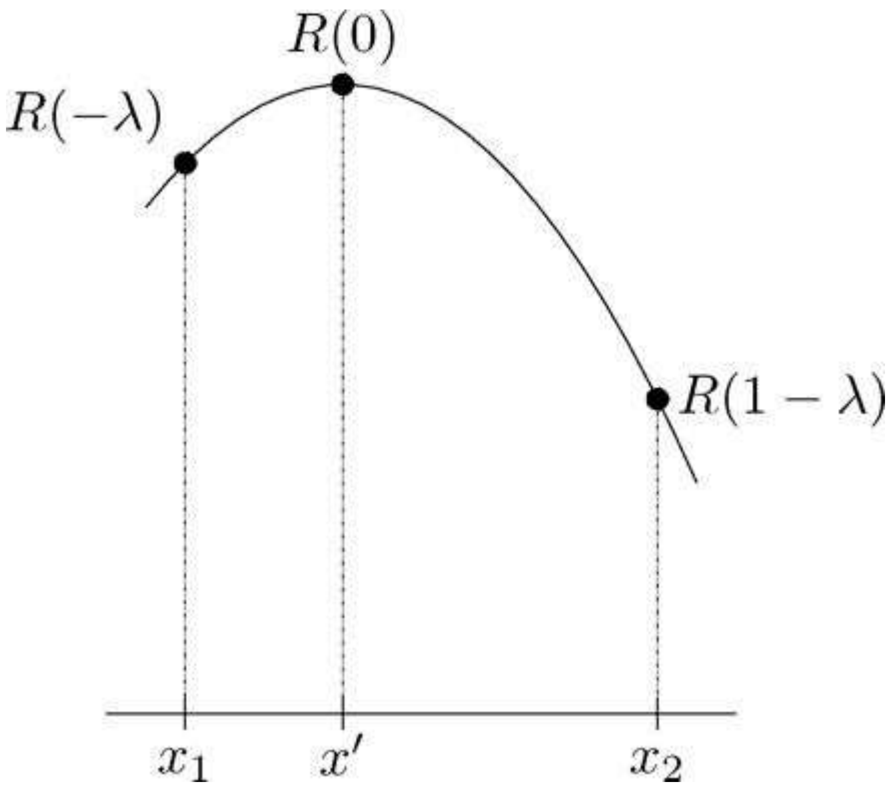
Nearest neighbor and bilinear interpolation are two special cases of a more general approach.

The idea is this: we wish to interpolate a value f(x') for $x_1 \leq x' \leq x_2$, and suppose $x' - x_1 = \lambda$. We define an interpolation function R(u), and set

$$f(x') = R(-\lambda)f(x_1) + R(1-\lambda)f(x_2).$$

General Interpolation

Figure 6.10: Using a general interpolation function



The function R(u) is centered at x', so x1 corresponds with u = −λ, and x2 with u = 1 − λ.

Bicubic Interpolation

Figure 6.15: Enlargement using bicubic interpolation

Spatial Filtering

Figure 6.16: Enlargement by spatial filtering

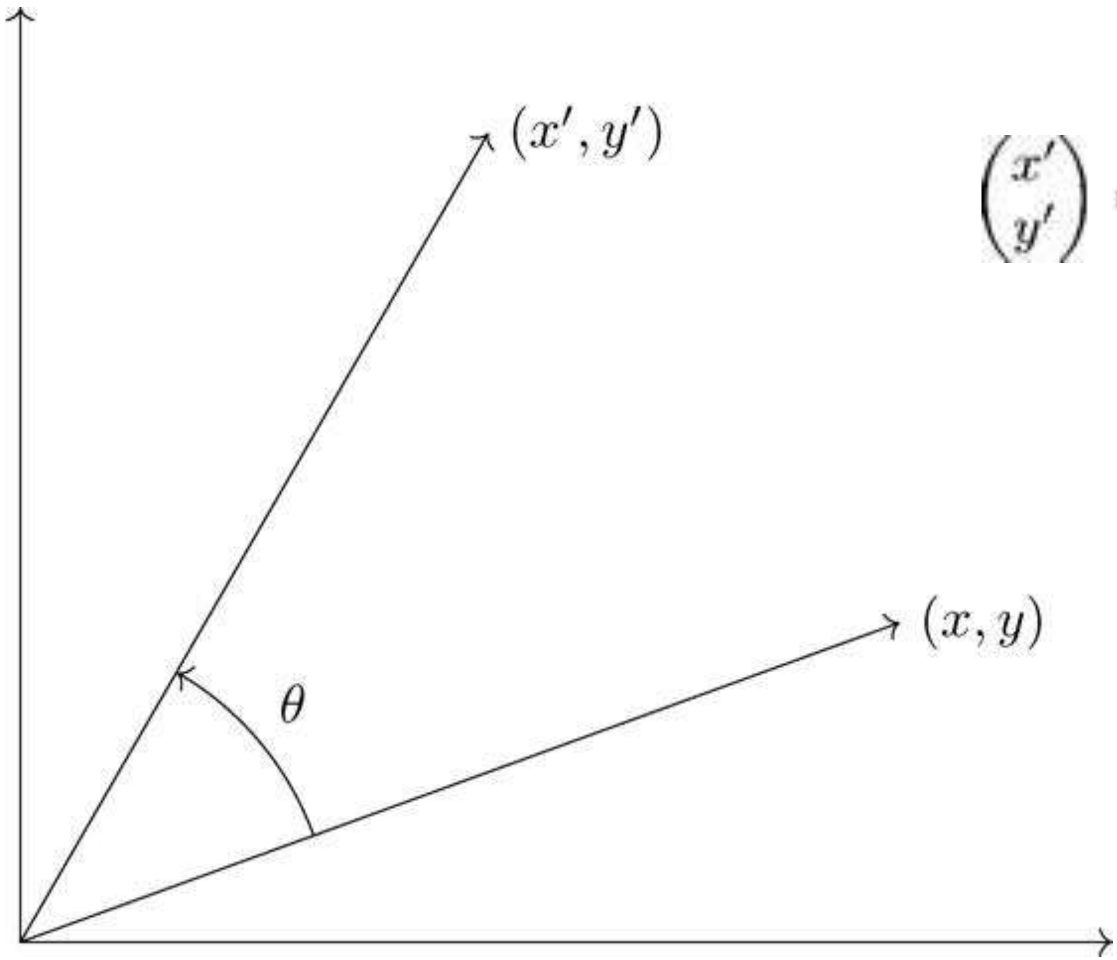
Zero interleaving | Nearest neighbor | Bilinear | Bicubic

Scaling Smaller

- Image minimization: making an image smaller
- How?
  - Subsampling: taking out alternate pixels
    - Can create gaps
  - Apply a low pass filter first

Rotation

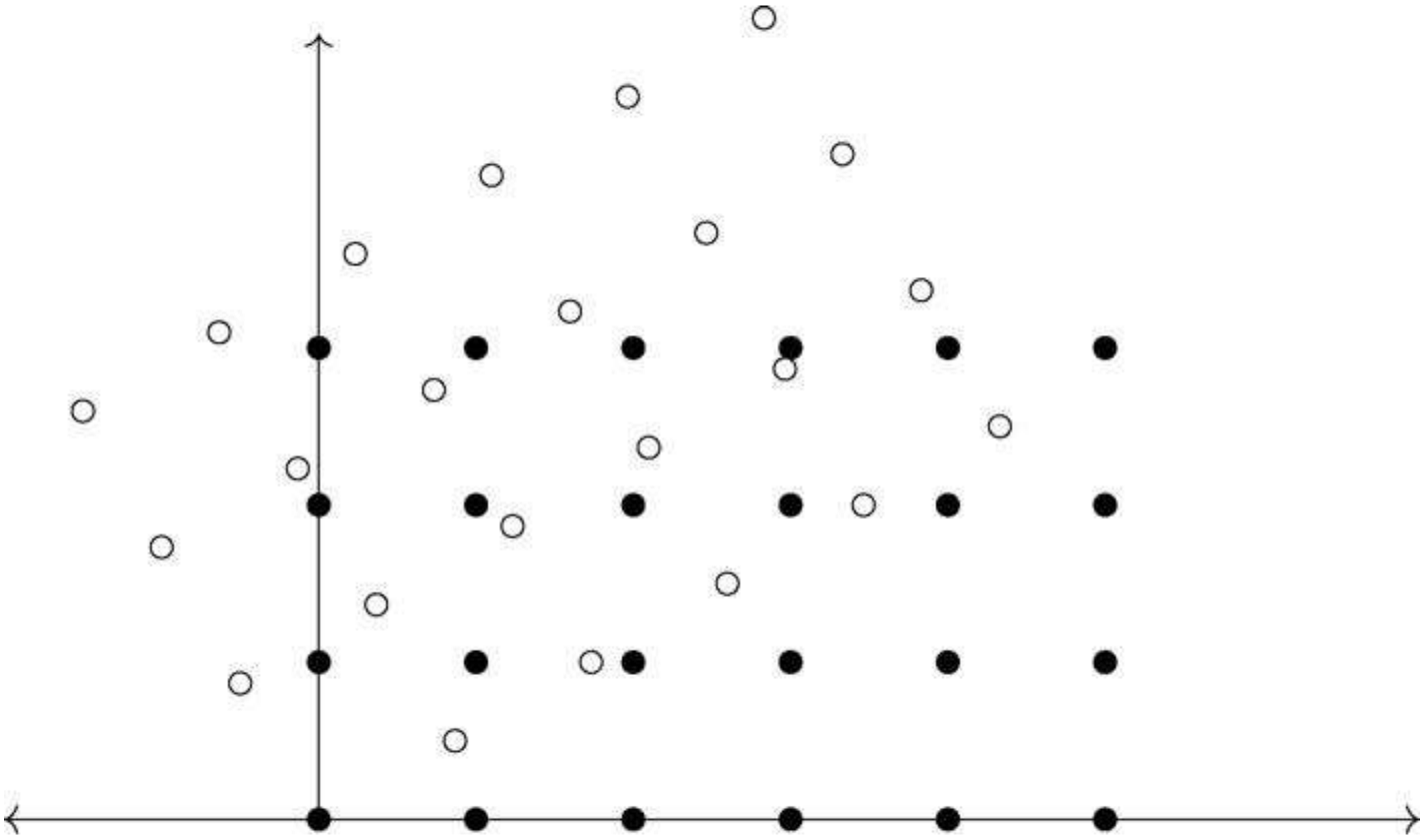
Figure 6.18: Rotating a point through angle θ



$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

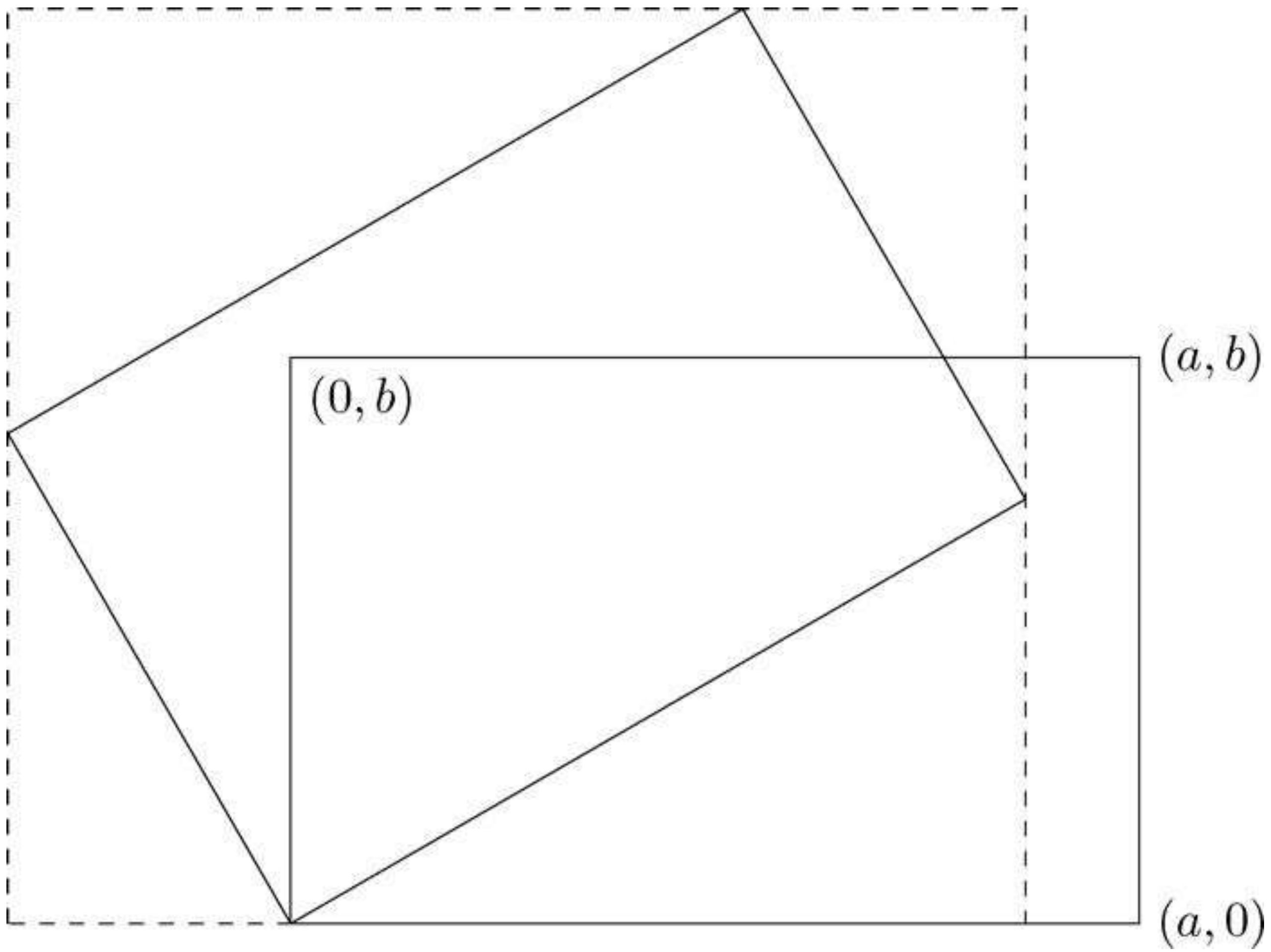
Rotation

Figure 6.19: Rotating a rectangle



Rotation

Figure 6.20: A rectangle surrounding a rotated image



Try It:

c = imread('cameraman.png')

```
>> cr = imrotate(c,60);
>> imshow(cr)
>> crc = imrotate(c,60,'bicubic');
>> figure,imshow(crc)
```

% default is nearest neighbor

MATLAB/Octave

and in Python the commands are

```
In :   cr = tr.rotate(c,60,order=0)
In :   io.imshow(cr)
In :   crc = tr.rotate(c,60,order=3)
In :   f = figure(), f.show(io.imshow(crc))
```

Python

Rotation

Figure 6.23: Rotation with interpolation

(a) Nearest neighbor

(b) Bicubic interpolation

Correcting Image Distortion

Perspective Distortion:

- Because of the position of the camera lens relative to the building, the towers appear to be leaning inward.
- Fixing this requires a little algebra.

Figure 6.24: Perspective distortion

Correcting Image Distortion

Figure 6.25: The corners of the building

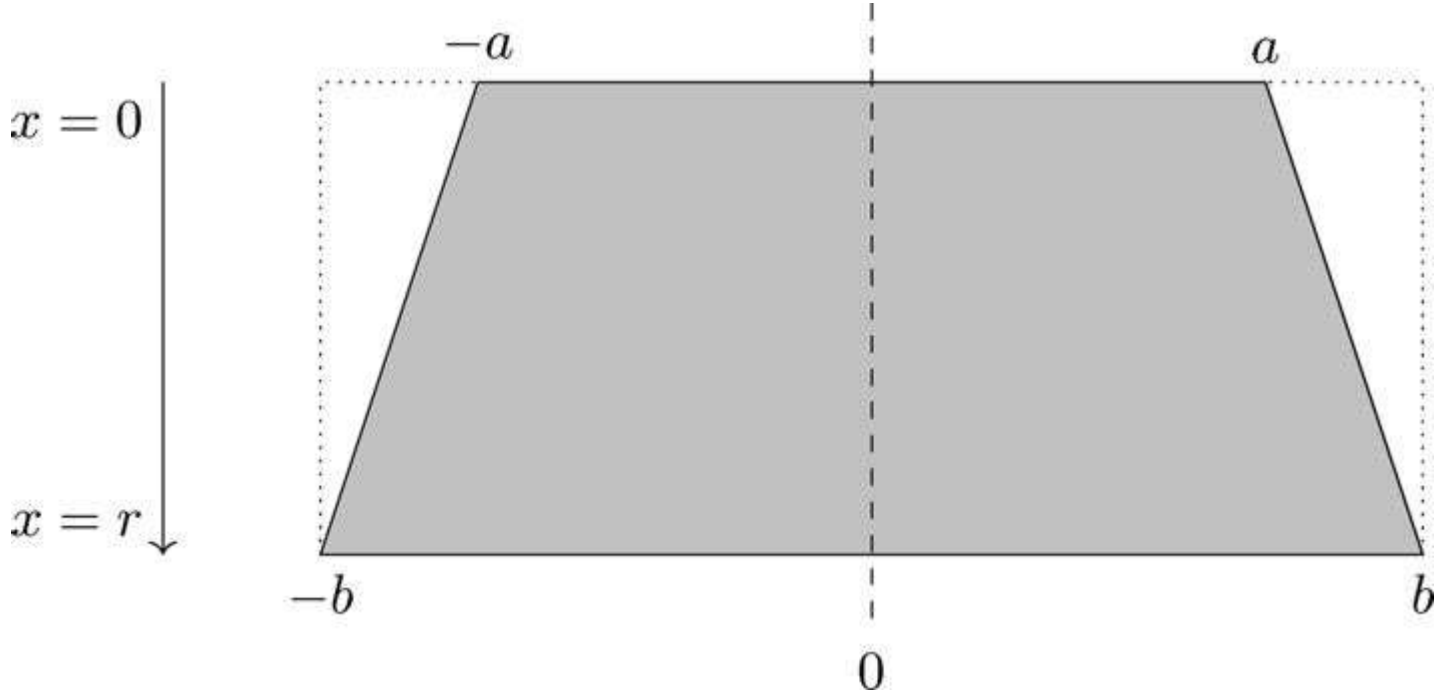
$(129, 69)$ $(129, 635)$

$(478, 29)$ $(478, 675)$

$y = 352$

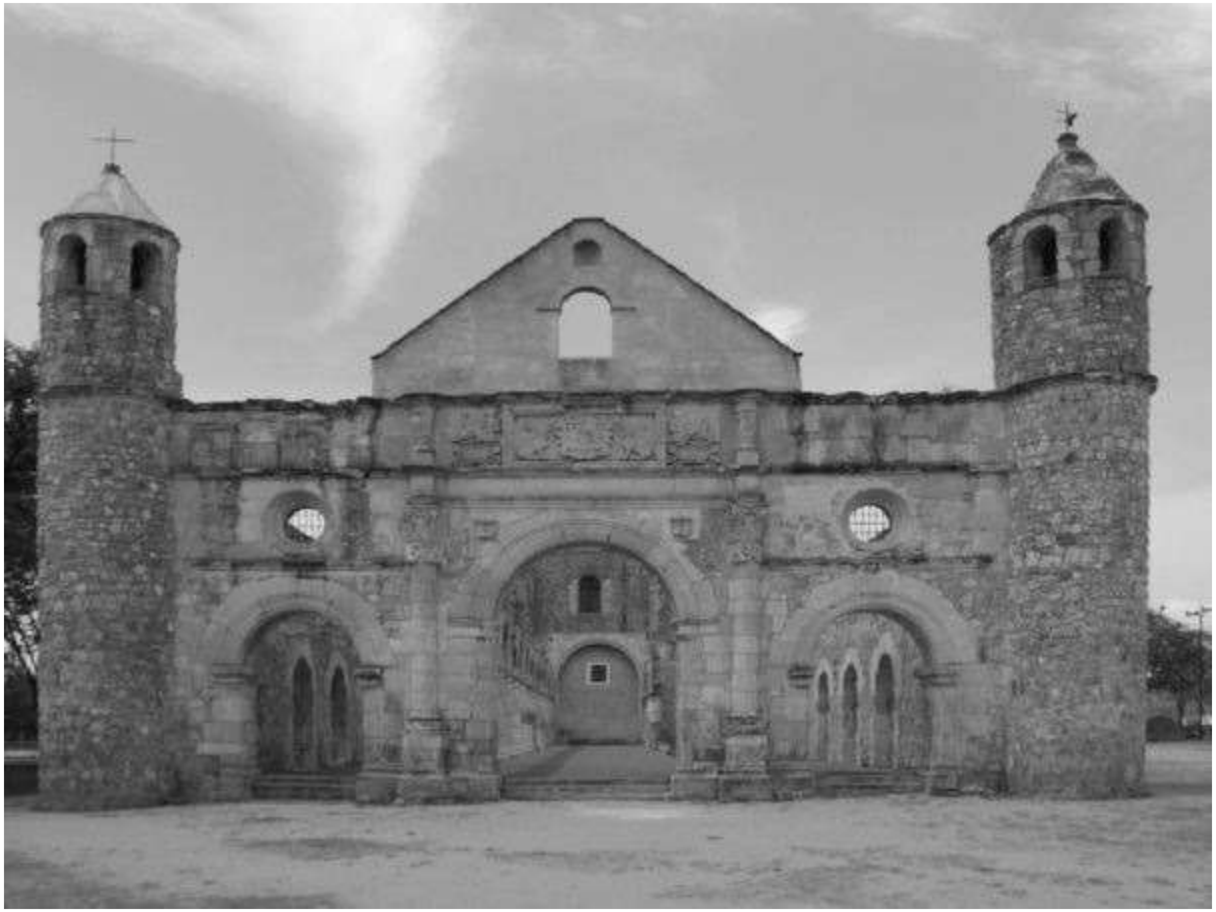
Correcting Image Distortion

Figure 6.26: A general symmetric trapezoid



Correcting Image Distortion

Figure 6.28: The image corrected for perspective distortion



im = imread('monastery.png');

```
>> [r,c] = size(im);
>> z = zeros(r,c);
>> x1 = 129; y1=283; x2=478; y2=323;
>> a = y1-x1*(y2-y1)/(x2-x1)
>> b = y1+(r-x1)*(y2-y1)/(x2-x1)
>> [y,x] = meshgrid(1:c,1:r);
>> sq = floor((y-352).*((b-a)/(b*r)*x+a/b)+352);
>> for i = 1:r,...
>    for j = 1:c,...
>      z(i,j) = im(i,sq(i,j));
>    end;...
>  end;
>> im2 = uint8(z)
```

MATLAB/Octave

```
In :  r,c = im.shape
In :  x1, y1, x2, y2 = 129.0, 283.0, 478.0, 323.0
In :  a = y1-x1*(y2-y1)/(x2-x1)
In :  b = y1+(r-x1)*(y2-y1)/(x2-x1)
In :  z = np.zeros_like(im)
In :  x,y = np.mgrid[0:r,0:c]
In :  sq = np.floor((y-352)*((b-a)/(b*r)*x+a/b)+352)
In :  for i in range(r):
...:      for j in range(c):
...:          z[i,j] = im[i,sq[i,j]]
...:
In :  im2 = uint8(z)
```

Python