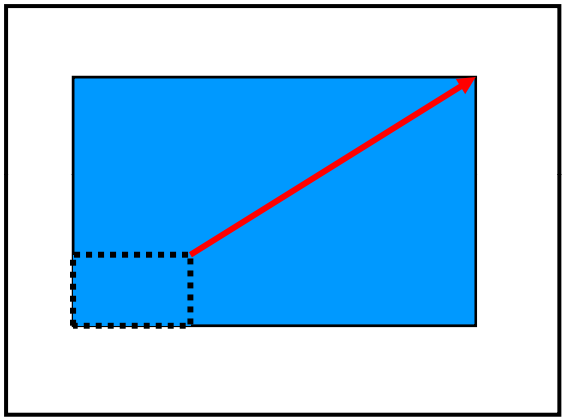
CHAPTER 4

GEOMETRIC PROCESSES

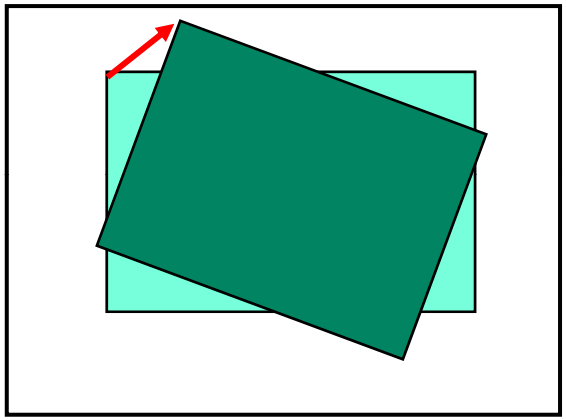
Modify the arrangement of pixels based on some geometric transformation.

Basic Geometric Processes

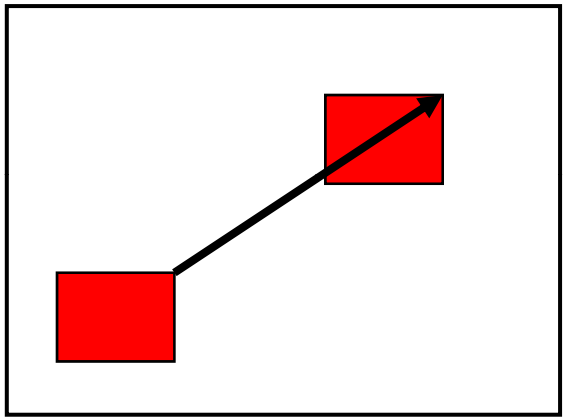
- Scale, Rotation, and Translation



Scale



Rotation

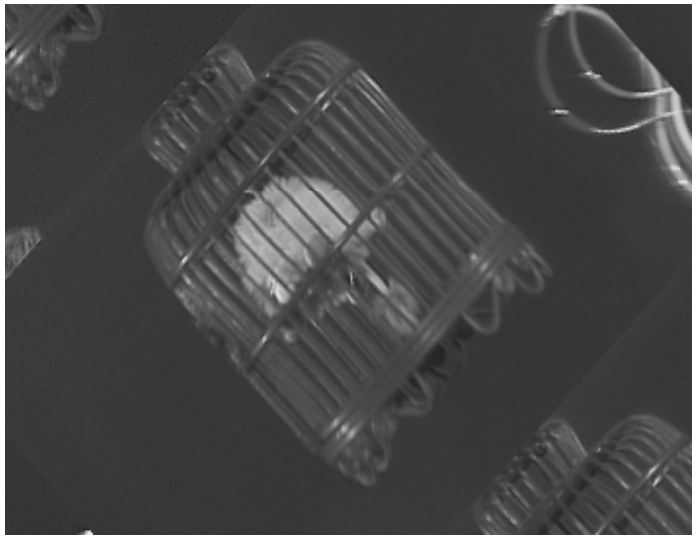


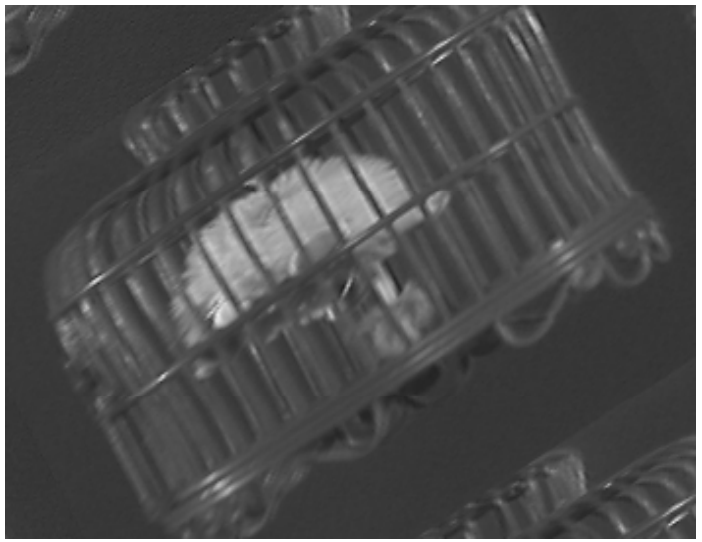
Translation

Geometric Processes

Original

Sx=2,
Sy=2

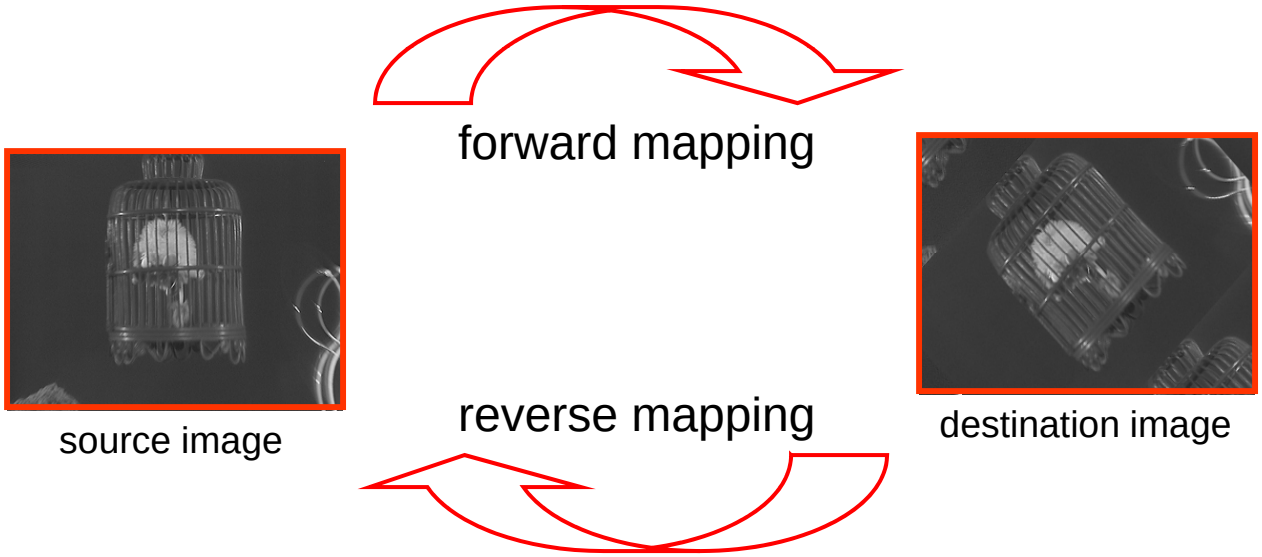




Sx=2,
Sy=1,

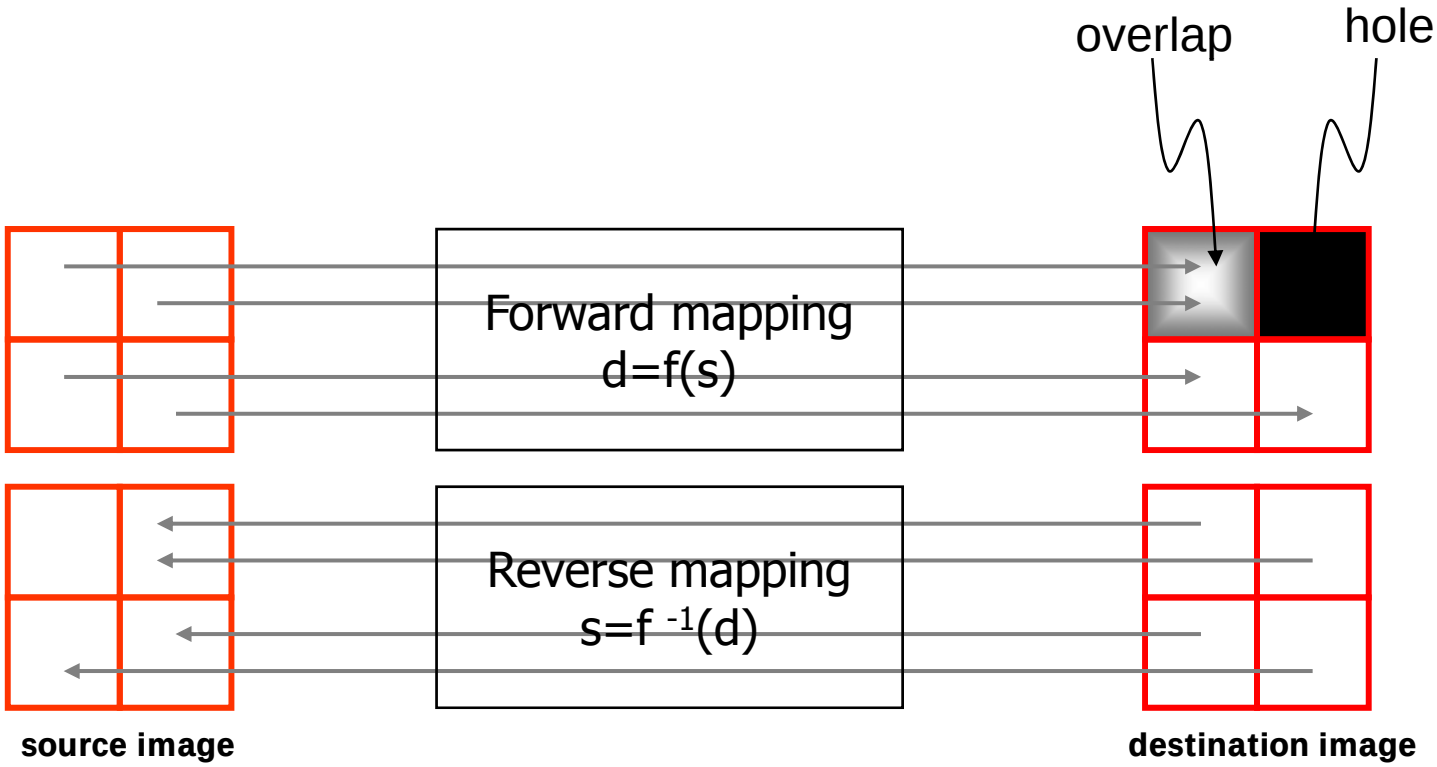
Forward vs. Reverse Mapping

- Forward Mapping
- Reverse Mapping



Forward vs. Reverse Mapping

- Two problems with forward mapping
  - Holes: undefined pixels (destination pixel has no corresponding source pixel)
  - Overlaps: two input pixels mapped to the same output pixel



Forward vs. Reverse Mapping

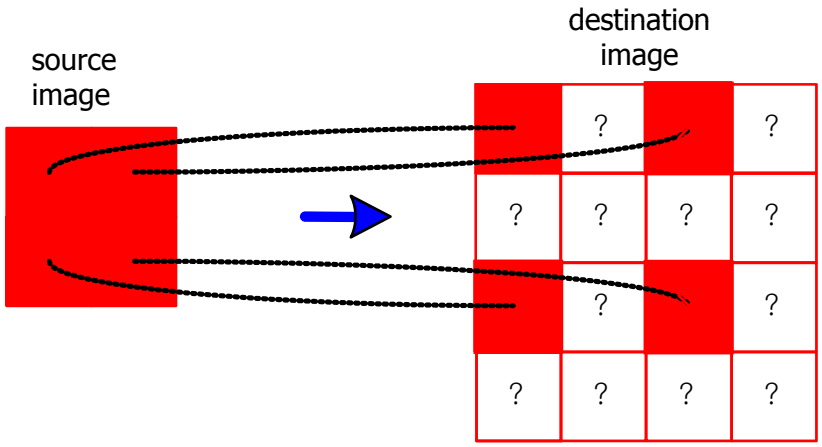
- Reverse Mapping
  - Calculate which pixel in the source image will be used to produce destination pixel via some inverse transformation.
  - Eliminate the problems of holes and overlaps.

original image | rotation by forward mapping | rotation by reverse mapping

Interpolation

- Example

$$x_{source} = \frac{x_{dest}}{2}, \quad y_{source} = \frac{y_{dest}}{2}$$



dest[0][0]← source[0][0]
dest[1][1] ← source[0.5][0.5] ???

Interpolation is the process of generating values for addresses that lie between pixels.

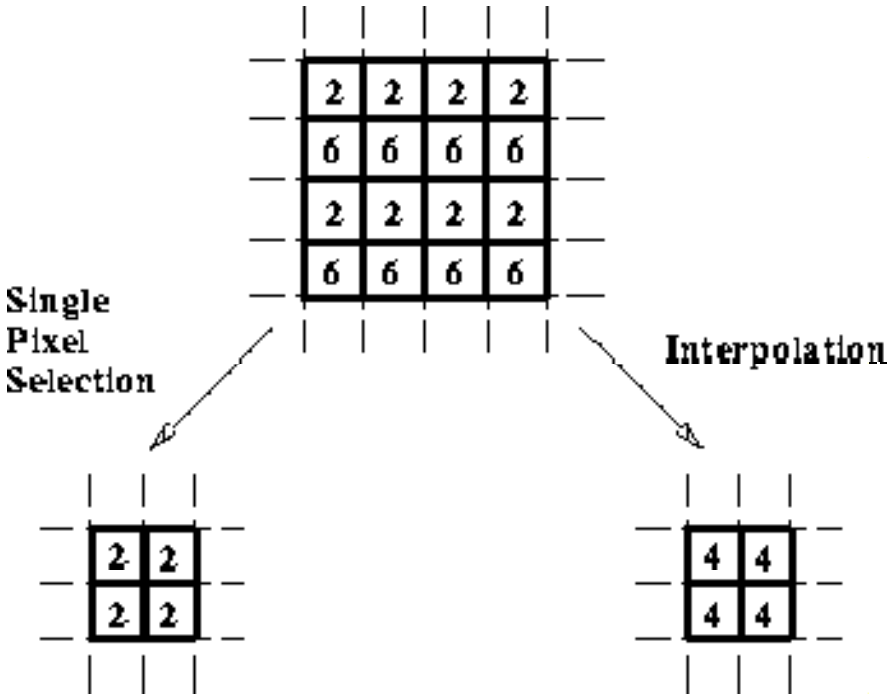
Interpolation Methods:
① Nearest Neighbor interpolation
② Bilinear interpolation
③ Cubic Convolution interpolation
④ B-Spline interpolation

Interpolation

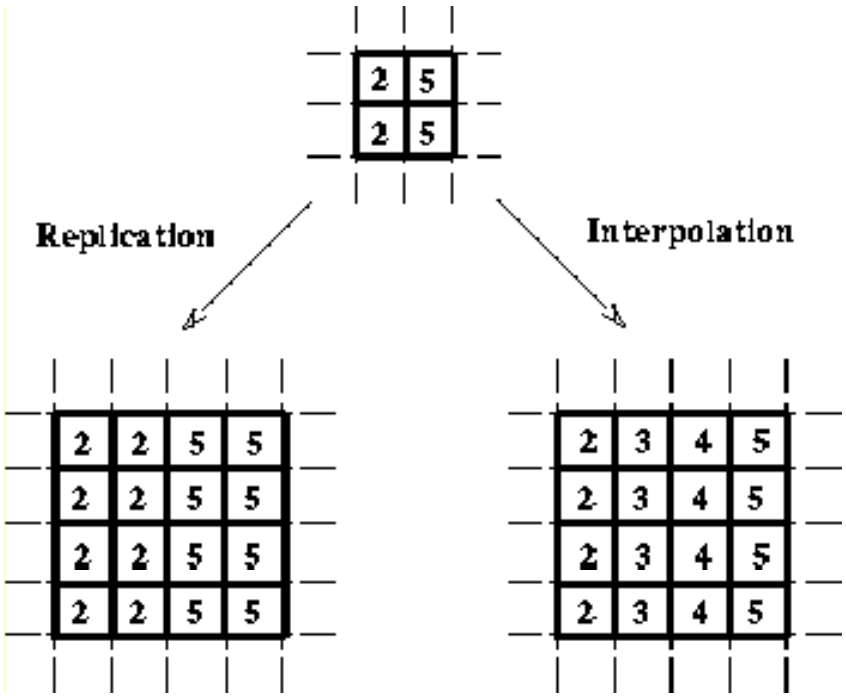
- What happens when a mapping function calculates a fractional pixel address?
- Interpolation generates a new pixel by analyzing the surrounding pixels.
- Right interpolation function is application dependent.
  - Trade off between quality and processing time
  - More sophisticated algorithms -> improve image quality -> more complex interpolation -> more processing time.

Interpolation

- Example



reduced by a factor of 2

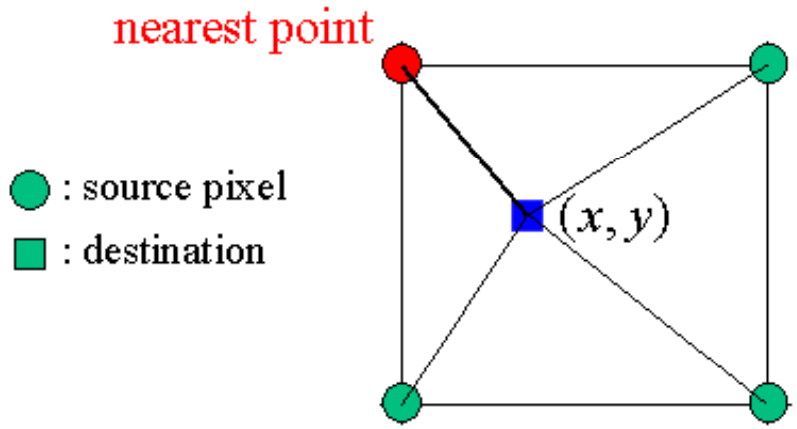


enlarged by a factor of 2

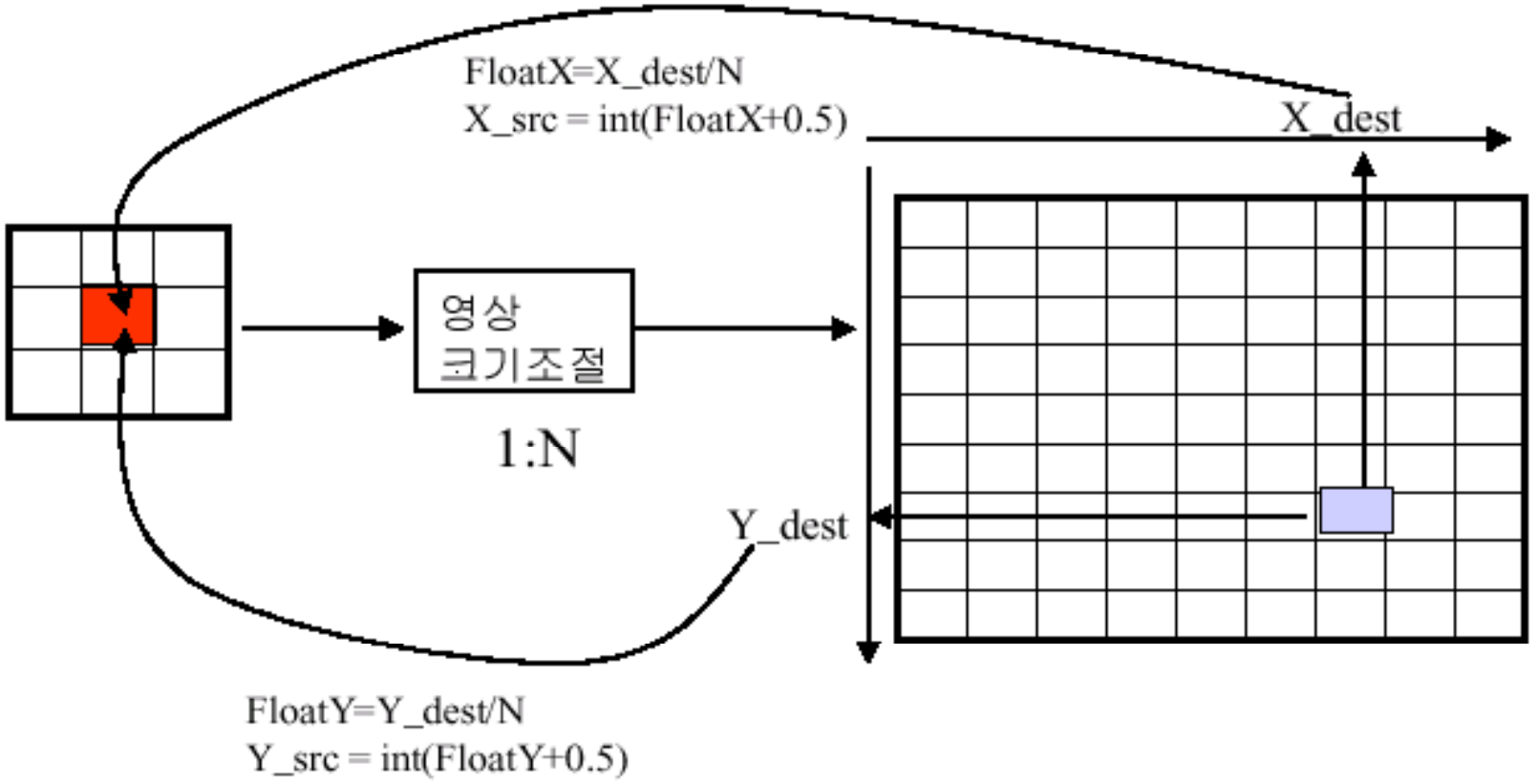
Nearest Neighbor Interpolation

- Assign the pixel closest to the newly generated address as output pixel.
  - The fractional address computed for the source pixel is rounded to the nearest valid pixel address.
- Code:

```
floatx = x_mapping_function(X_dest);
floaty = y_mapping_function(Y_dest);
X_source = (int)(floatx+0.5);
Y_source = (int)(floaty+0.5);
```

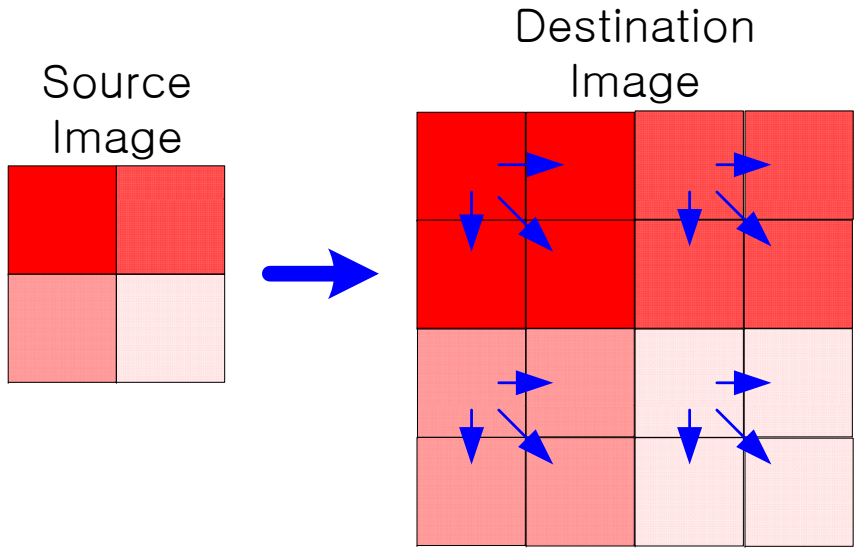


Nearest Neighbor Interpolation



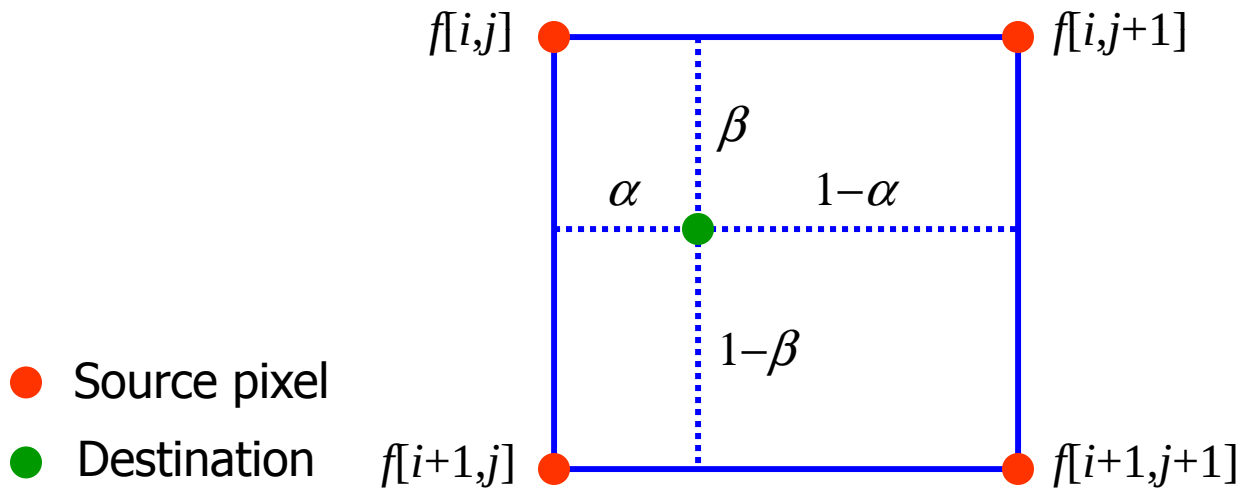
Nearest Neighbor Interpolation

- Fast, but yields greatly varying results.
  - The greater the number of output pixels tied to one input pixel, the worse the output looks.



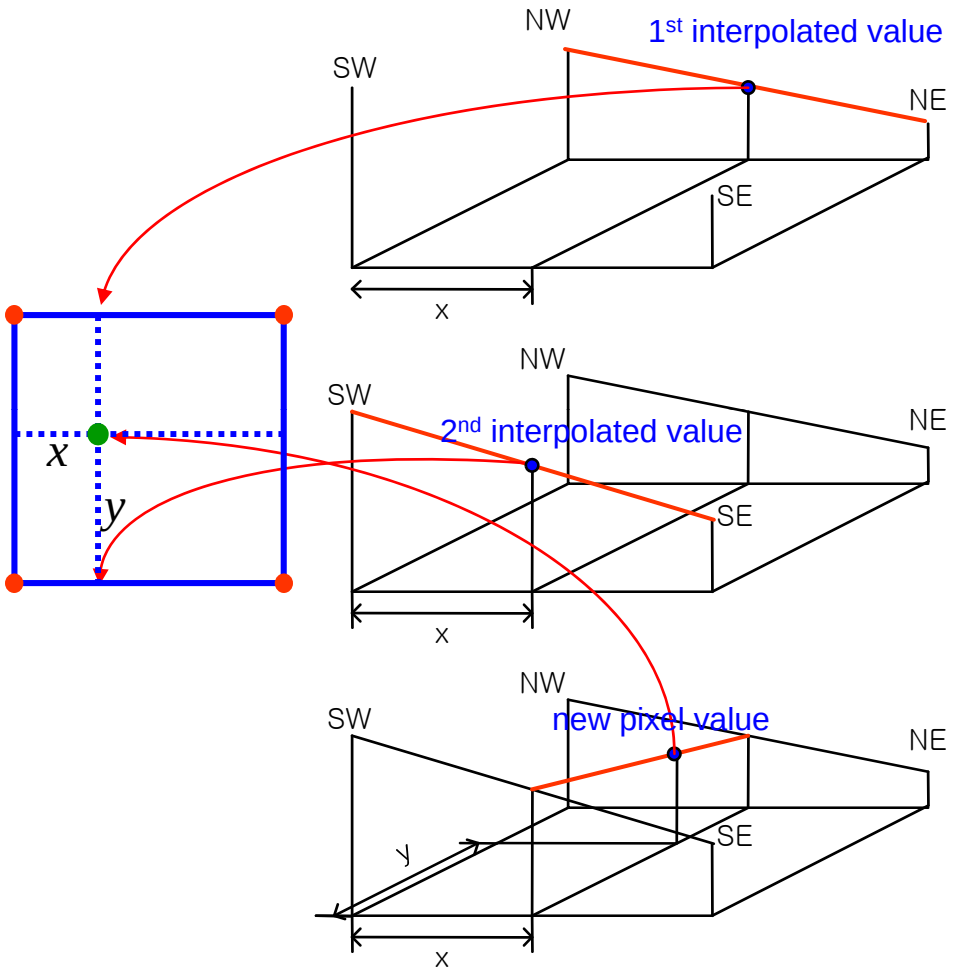
Bilinear Interpolation

- The newly generated pixel is a weighted sum of the four nearest pixels.
- The weights are determined linearly; each weight is directly proportional to the distance from each existing pixel.



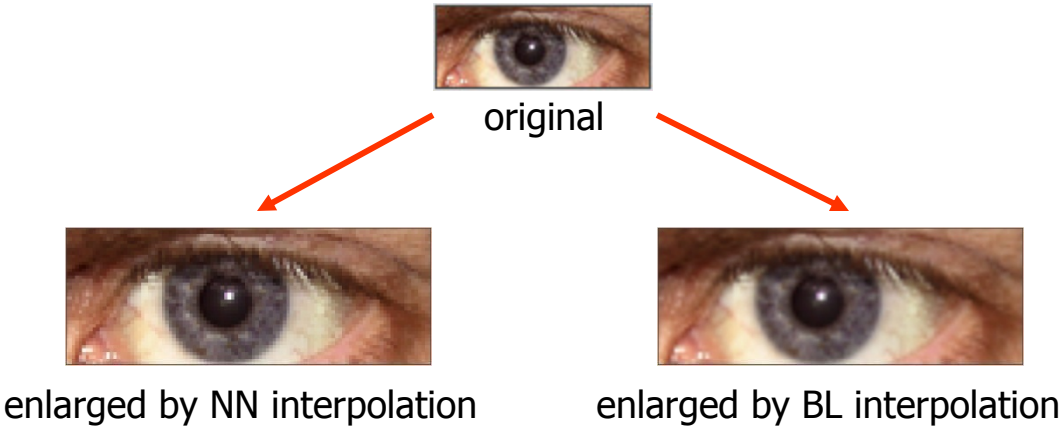
$$\text{new} = (1-\alpha)(1-\beta)\, f[i, j] + \alpha(1-\beta)\, f[i, j+1]$$
$$+ (1-\alpha)\,\beta\, f[i+1, j] + \alpha\beta\, f[i+1, j+1]$$

Bilinear Interpolation



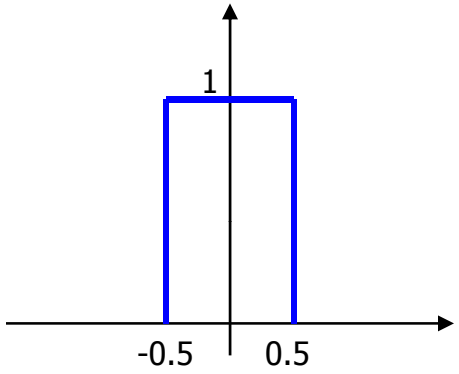
Bilinear Interpolation

- Bilinear (BL) interpolation yields a smoother image than nearest neighbor (NN) interpolation.
- Because of the three linear interpolations per pixel, BL interpolation requires significantly more computations than NN interpolation.

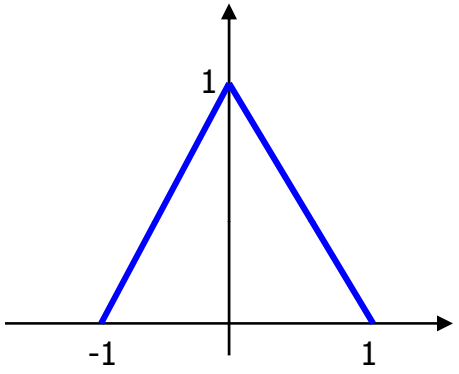


Nearest Neighbor vs. Linear Interpolation

- 1D Case: $g(x) = \sum_{n=-\infty}^{\infty} h(x-n)\, f[n]$



NN interpolation kernel



BL interpolation kernel

- 2D Case: $g(x,y) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} h(x-n, y-m)\, f[n,m]$

$$h_{NN}(x,y) = h_{NN}(x)h_{NN}(y)$$

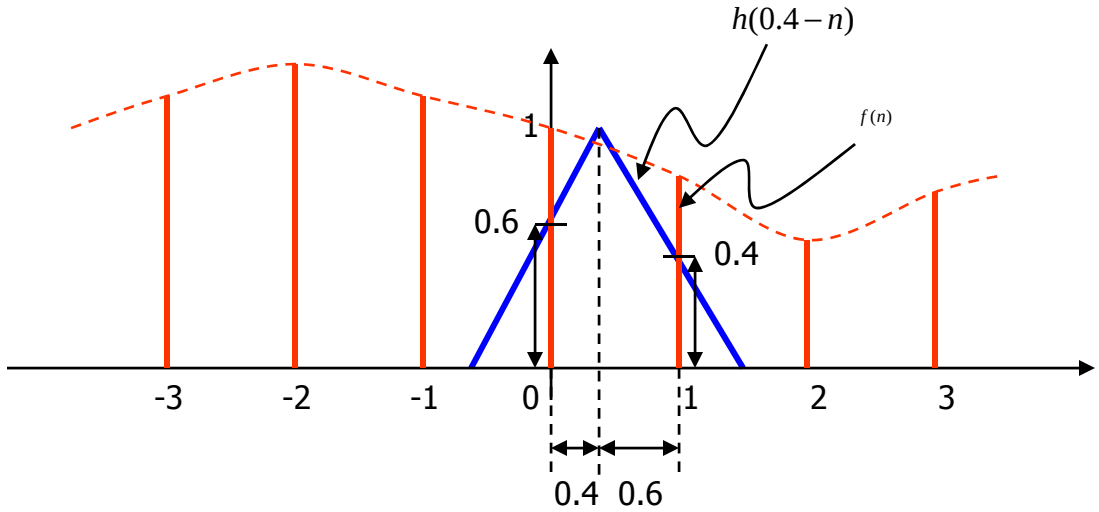
$$h_{BL}(x,y) = h_{BL}(x)h_{BL}(y)$$

How to Perform Interpolations

- Recall:

$$g(x,y)=\sum_{n=-\infty}^{\infty}\sum_{m=-\infty}^{\infty}h(x-n,y-m)\,f[n,m]$$

- 1-D Example: $g(0.4)=\sum_{n=-\infty}^{\infty}h(0.4-n)\,f(n)$



Higher Order Interpolation

- Cubic Interpolation
  - Higher order interpolation functions span larger areas to generate output pixels. (16 nearest pixels for cubic interpolation)
  - The 1-D cubic convolution function is defined as

$$h(x)=\begin{cases}(a+2)|x|^3-(a+3)|x|^2+1 & 0\le|x|<1\\ a|x|^3-5a|x|^2+8a|x|-4a & 0\le|x|<2\\ 0 & 2\le|x|\end{cases}$$

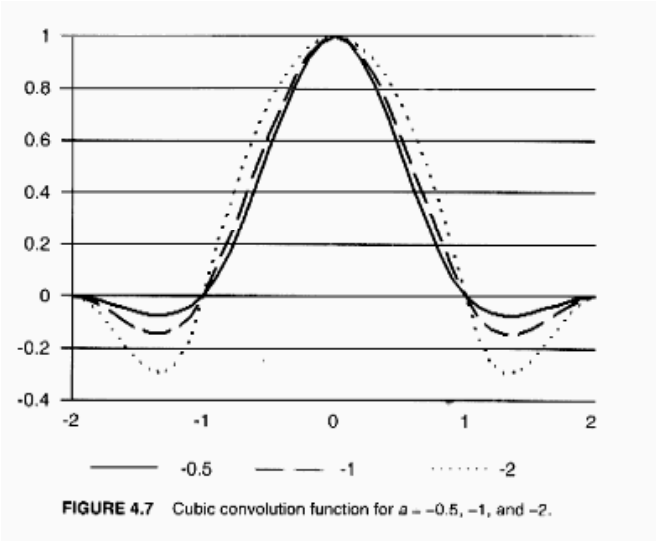
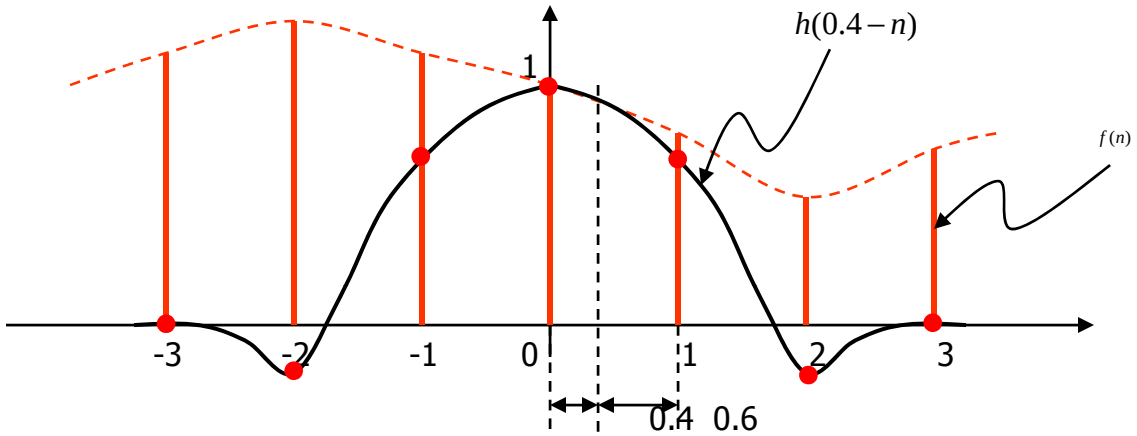


FIGURE 4.7 Cubic convolution function for $a = -0.5, -1$, and -2.

NOTE: The function has negative values.
- Will sharpen more than any other function.
- Can output negative numbers. (Clipping needed)
- Can generate pixel values greater than the maximum value. (Clipping needed)

Higher Order Interpolation



- Saving Computations by Horner's Rule:

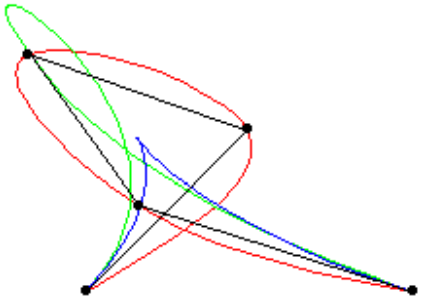
$$x^2 + x = (x+1)x$$

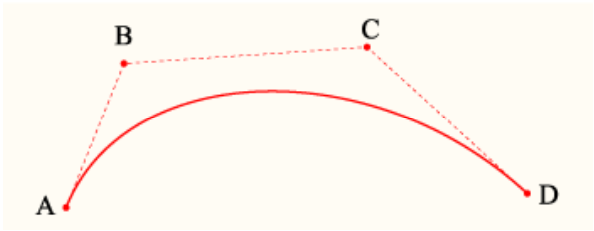
$$x^3 + 2x^2 + 3x + 4 = \Big(\big((x+2)x+3\big)x+4\Big)$$

B-Spline Interpolation

- What is a "spline"?

  A spline is a long strip of wood (lath) that is fixed in a number of points. In older days splines were often used in shipbuilding to mark the curve of the hull. The lath will then take the shape which minimizes the energy required for bending it between the fixed points, and thus adopt the smoothest possible shape. Later splines have been made from rubber, steel, and other elastomeric materials.





B-Spline Interpolation

$$f(x)=\begin{cases} 1/2\,|x|^3-|x|^2+2/3 & 0\le|x|<1 \\ -1/6\,|x|^3+|x|^2-2|x|+3/4 & 0\le|x|<2 \\ 0 & 2\le|x| \end{cases}$$

- The B-spline function makes a good low-pass filter.
- The sum of the sample points is 1.
- No need to clip the output.
- Strictly positive.

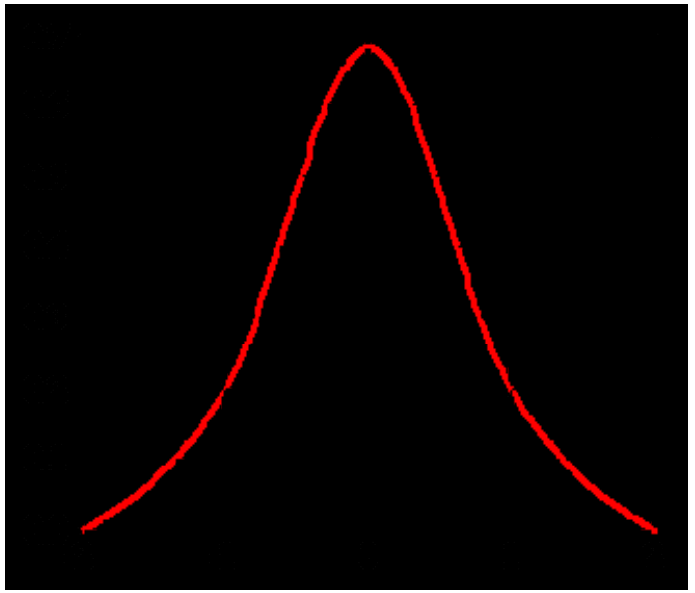
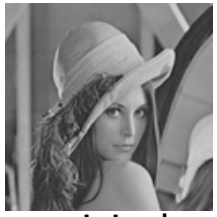


Image Scaling

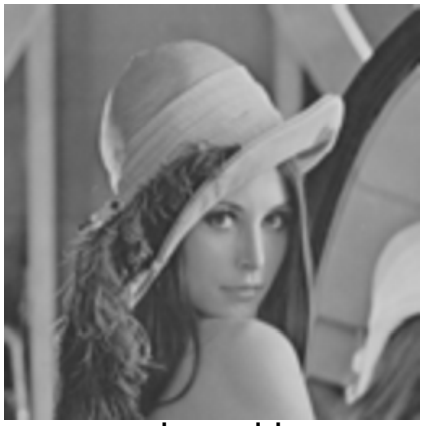
- Image Scaling
  - Enarging image: magnification, scaling up, zooming, upsampling, stretching.
  - Reducing image: shrinking, scaling down, decimation, downsampling, minification.
- Two important things:
  - Never improve upon the original resolution.
  - The resulting image degrades with every operation.
- Methods:
  - Compute source addresses by dividing destination by scaling factors.
    - x_source = x_dest / x_scale
    - y_source = y_dest / y_scale
  - Determine the pixel value by using one of the interpolation functions.

Image Scaling



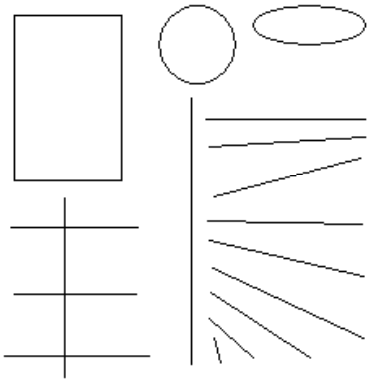
original
(128x128)

enlarged (256x256) by
NN interpolation



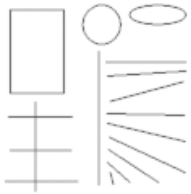
enlarged by
BL interpolation

enlarged by
BC interpolation

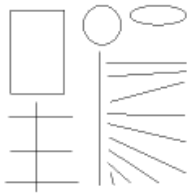


original
(256x256)

reduced by
subsampling



blurred then
reduced

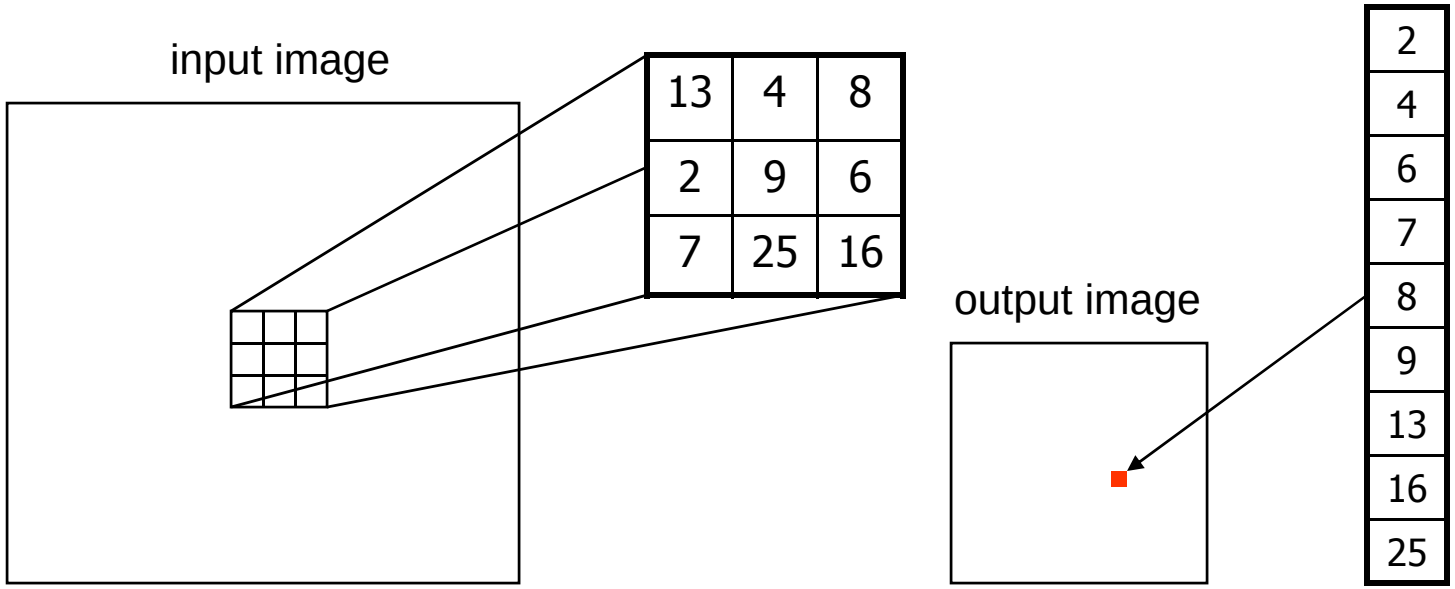


reduced by
BL interpolation

reduced by
BC interpolation

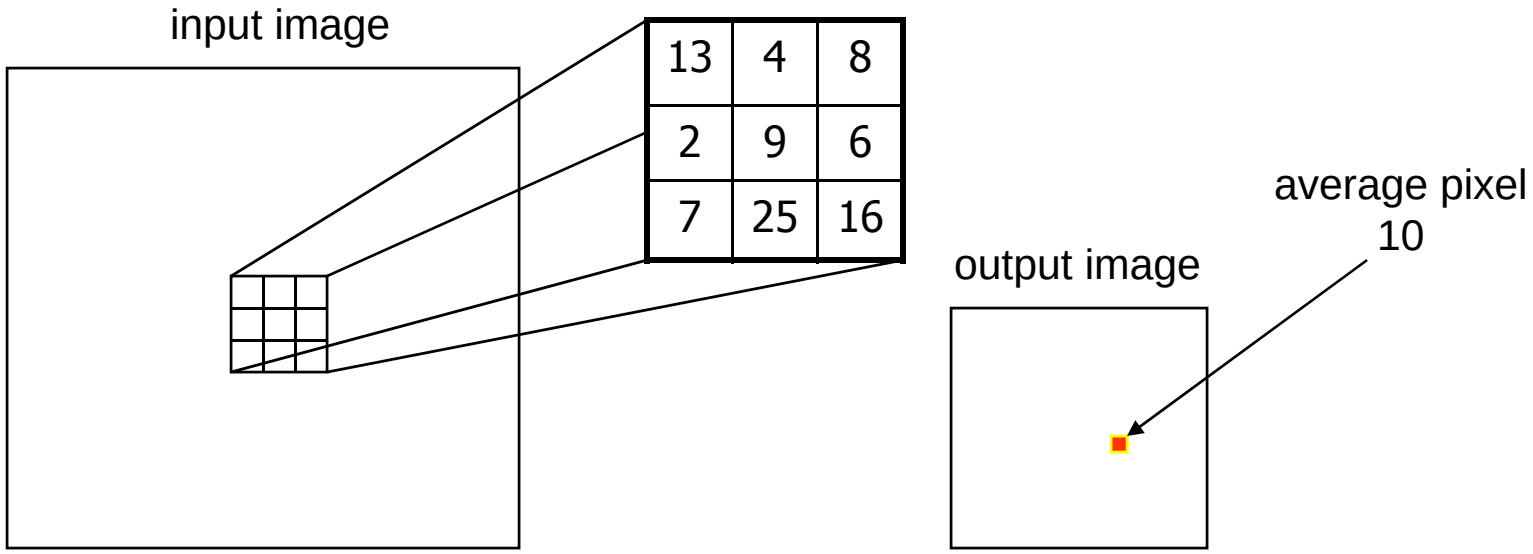
Minification Only Techniques

- Median Representation
  - Basic idea: represent a block of pixels with one pixel.
  - Pass an nxn window over the image.
  - Replace a block of pixels with its median value.



Minification Only Techniques

- Average Representation
  - Replace a block of pixels with an average value of all of the pixels.
  - Faster than median representation.



Rotation

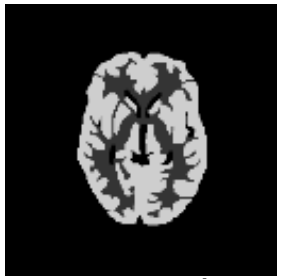
- Rotates an image about its center pixel through any given angle.
- Rotation about (0,0):

$$\begin{pmatrix} x_{\text{src}} \\ y_{\text{src}} \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ \cos\theta & -\sin\theta \end{pmatrix} \begin{pmatrix} x_{\text{dst}} \\ y_{\text{dst}} \end{pmatrix}$$

- Rotation about image center $(x_{\text{ctr}}, y_{\text{ctr}})$:

$$\begin{pmatrix} x_{\text{src}} \\ y_{\text{src}} \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ \cos\theta & -\sin\theta \end{pmatrix} \begin{pmatrix} x_{\text{dst}} - x_{\text{ctr}} \\ y_{\text{dst}} - y_{\text{ctr}} \end{pmatrix}$$

- Rotation by multiples of 90° can be done simply by transposing the image matrix. (No interpolation is needed.)



original image

rotation by forward mapping

rotation by reverse mapping

Translation

- Move a section of the image to another location in the image.

$$x_{\text{dst}} = x_{\text{src}} + \Delta_x$$
$$y_{\text{dst}} = y_{\text{src}} + \Delta_y$$

$$\begin{bmatrix} x_{\text{dst}} \\ y_{\text{dst}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta_x \\ 0 & 1 & \Delta_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{src} \\ y_{\text{src}} \\ 1 \end{bmatrix}$$

- Interpolation is not needed.
- Needs two buffers to avoid possible overlaps between source and destination windows.

Mirroring

- Horizontal mirroring: flips an image about the y-axis.
- Vertical mirroring: flips an image about the x-axis.
- Flipping is a pure geometric process. (No interpolation involved)
- Requires two buffers (a source and destination image buffers).

original

horizontal mirroring

vertical mirroring