Geometric Transformation

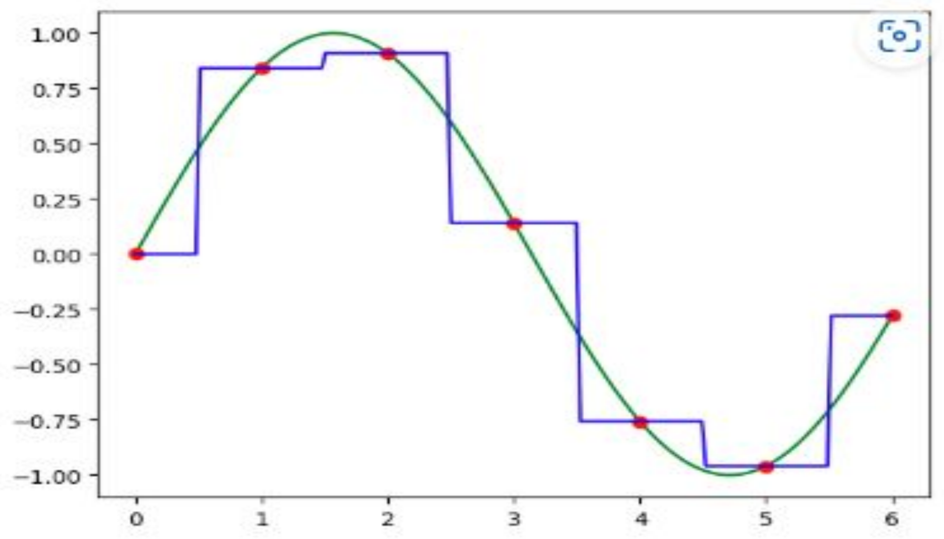
Image Interpolation

- **Interpolation in image processing** is a technique used to estimate pixel values at non-sampled points based on known pixel values. It is essential in various applications, such as resizing images, enhancing image quality, and remapping pixel grids. Common methods of interpolation include:

- **Nearest Neighbour**: Assigns the value of the nearest known pixel to the unknown pixel.
- **Bilinear**: Applies linear interpolation in two dimensions, considering the closest four pixels.
- **Bicubic**: Uses the values of the nearest 16 pixels to calculate the new pixel value, providing smoother results.
- Interpolation is crucial in digital photos, especially during processes like Bayer demosaicing and photo enlargement. For a more detailed understanding, you can refer to the foundational concepts of image interpolation.

Nearest Neighbour

```
In [1]: from scipy.interpolate import interp1d

In [2]: x = np.linspace(0, 6, 200); # fine sampling to represent continuous function

In [3]: xs = np.array([0, 1, 2, 3, 4, 5, 6]); # the sample points

In [4]: f = np.sin(x); F = np.sin(xs); # the 'continuous' function and its sampled version

In [5]: ifunc = interp1d(xs, F, kind='nearest'); hatf_nn = ifunc(x);

In [6]: plt.clf(); plt.plot(x, f, 'g-', xs, F, 'ro', x, hatf_nn, 'b-');
```

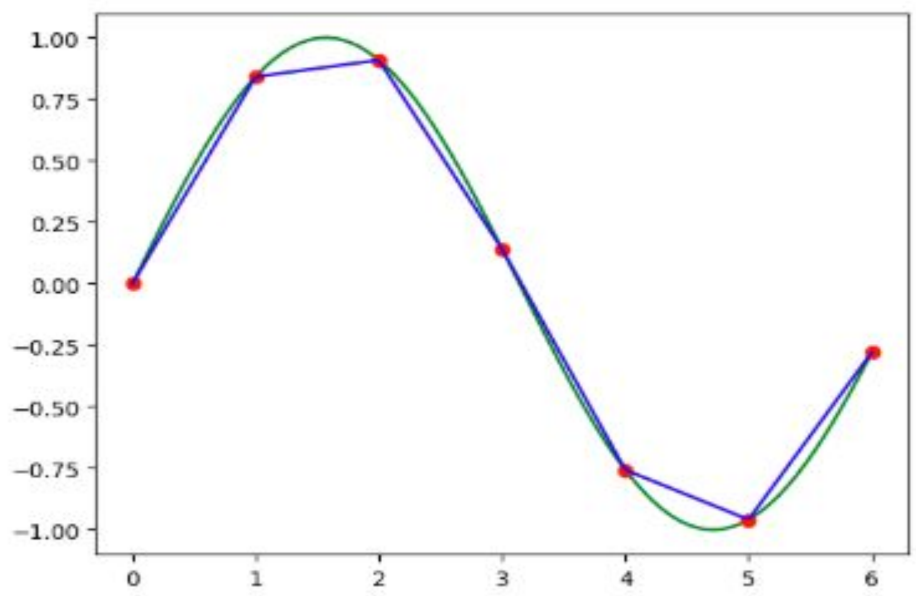


Linear Interpolation

Linear interpolation is the scientific equivalent of what you have already learned in kindergarten: connect the dots. Between to adjacent sample points k and $k+1$ we assume the function is a linear function and thus in this interval $[k, k+1]$ we have:

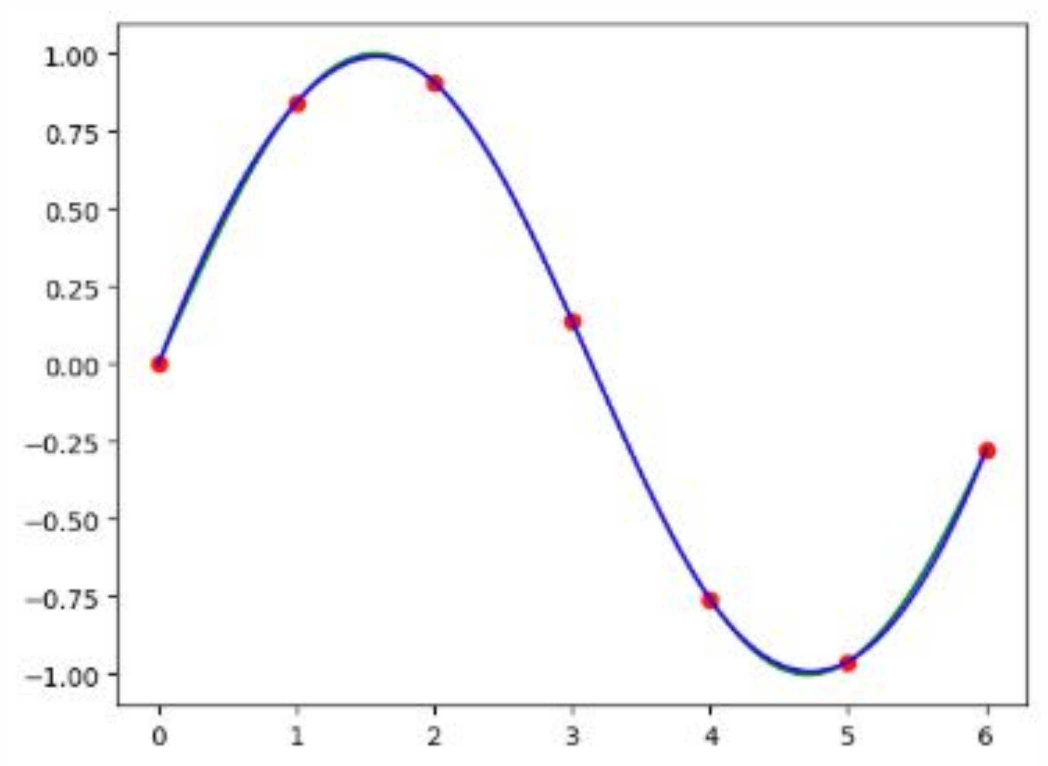
$$k \leq x \leq k+1 : \quad \hat{f}(x) = (1-(x-k))F(k) + (x-k)F(k+1)$$

```
In [7]: ifunc = interp1d(xs, F, kind='linear'); hatf_lin = ifunc(x);

In [8]: plt.clf(); plt.plot(x, f, 'g-', xs, F, 'ro', x, hatf_lin, 'b-');
```



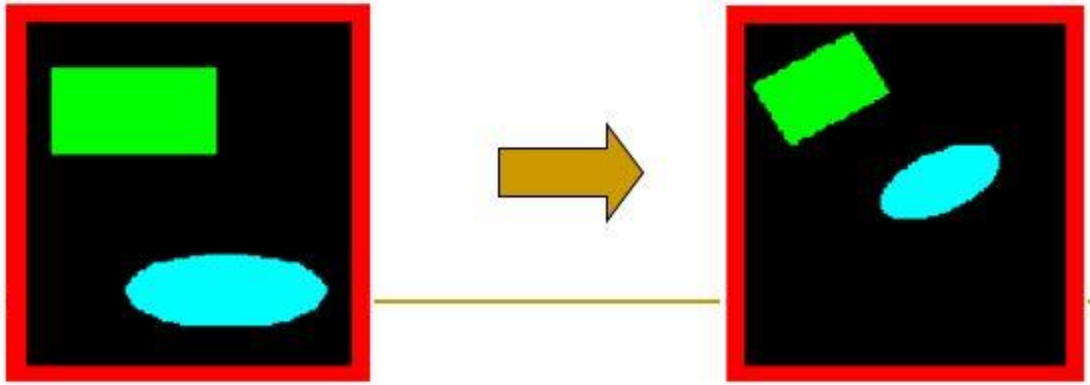
Cubic Interpolation

```
In [9]: ifunc = interp1d(xs, F, kind='cubic'); hatf_cubic = ifunc(x);

In [10]: plt.clf(); plt.plot(x, f, 'g-', xs, F, 'ro', x, hatf_cubic, 'b-');
```



Geometric Operations

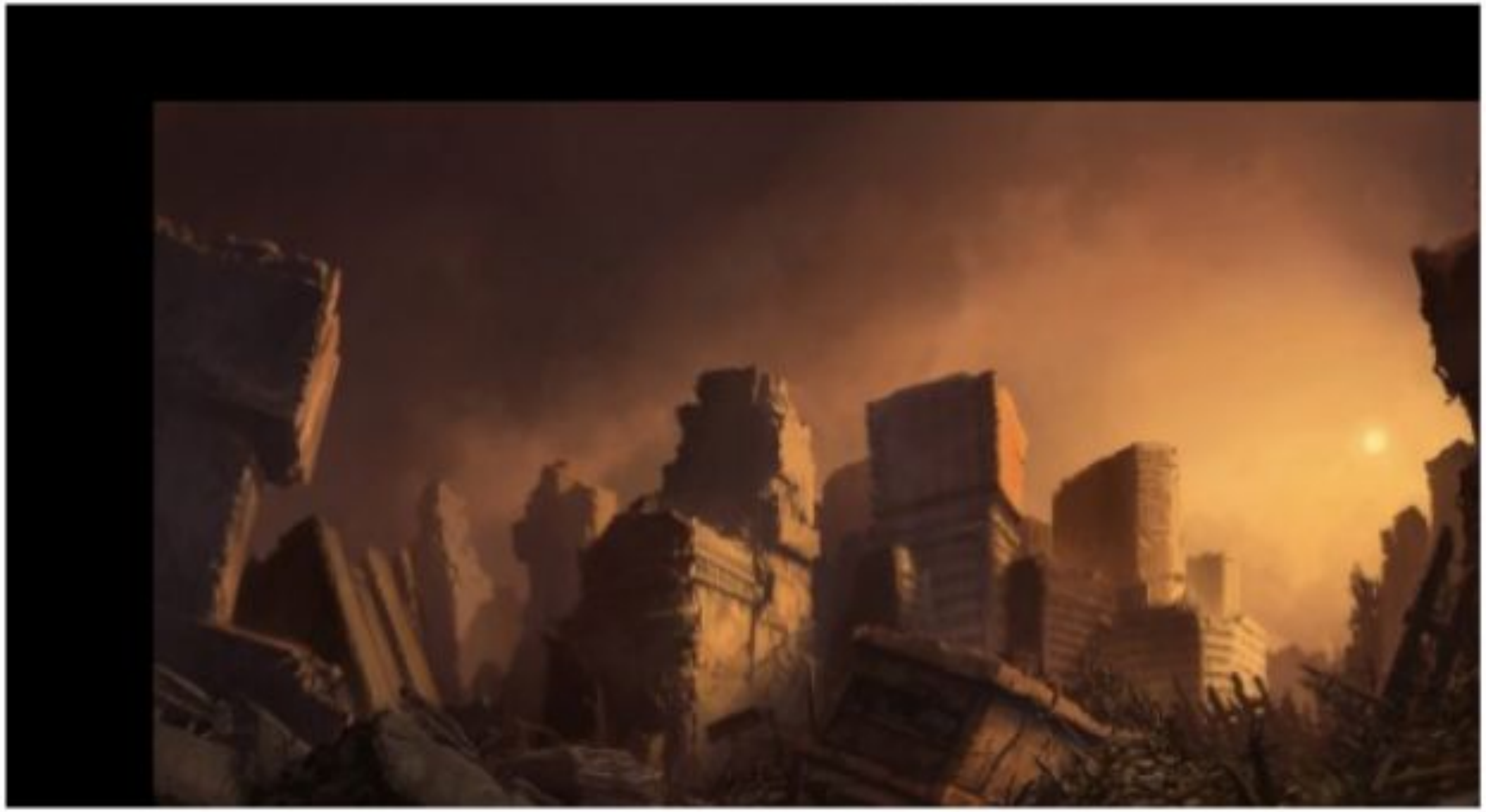
- **Scale** - change image content size
- **Rotate** - change image content orientation
- **Reflect** - flip over image contents
- **Translate** - change image content position
- **Affine Transformation**
  - general image content linear geometric transformation



Translation:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('D://SJC//Image Processing//images//image1.jpg')  # Provide the path to your image her
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
cv2.imshow('Image', image)  # 'Image' is the window name, and image is the image to be displayed
cv2.waitKey(0)  # Wait indefinitely for a key press
cv2.destroyAllWindows()



# Translation
def translate_image(image, tx, ty):
    rows, cols, _ = image.shape
    # Define the translation matrix
    translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])
    # Apply the transformation using cv2.warpAffine
    translated_image = cv2.warpAffine(image, translation_matrix, (cols, rows))
    return translated_image

# Example: Translate the image 50 pixels right and 30 pixels down
translated_image = translate_image(image, 50, 30)
```

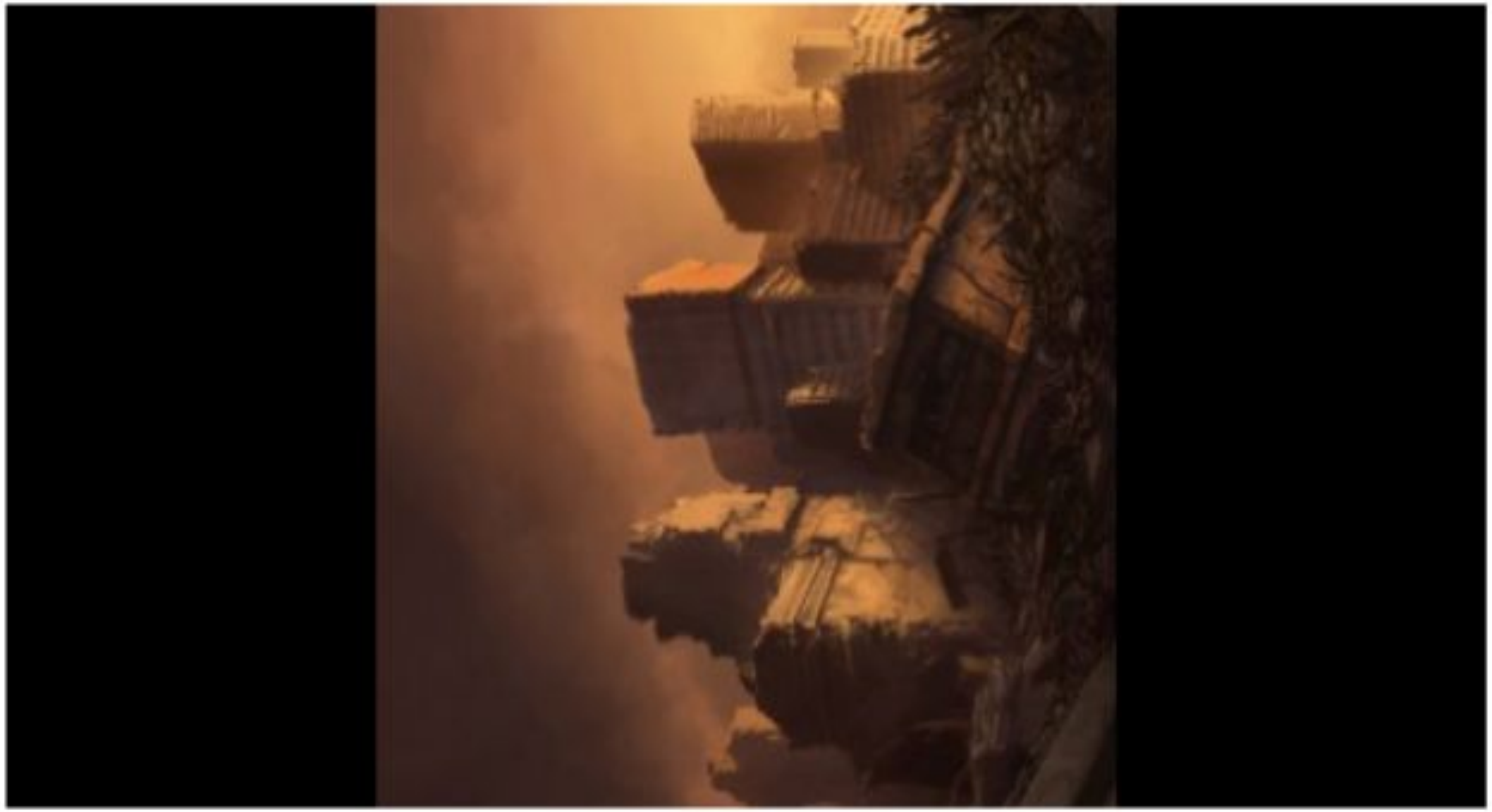
Translated Image



Rotation:

```python
#rotation
def rotate_image(image, angle):
    rows, cols, _ = image.shape
    # Get the rotation matrix
    rotation_matrix = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
    # Apply the rotation
    rotated_image = cv2.warpAffine(image, rotation_matrix, (cols, rows))
    return rotated_image

# Example: Rotate the image by 45 degrees
rotated_image = rotate_image(image, 90)

# Display the result
plt.imshow(rotated_image)
plt.title('Rotated Image')
plt.axis('off')
plt.show()
```

Rotated Image



Scaling:

```python
#scaling

def scale_image(image, fx, fy):
    # fx and fy are scaling factors along the x and y axes
    scaled_image = cv2.resize(image, None, fx=fx, fy=fy, interpolation=cv2.INTER_LINEAR)
    return scaled_image

# Example: Scale the image by a factor of 1.5 in both x and y directions
scaled_image = scale_image(image, 1.5, 1.5)

# Display the result
plt.imshow(scaled_image)
plt.title('Scaled Image')
plt.axis('off')
plt.show()
```

Scaled Image

Affine transform:

- An affine transformation is a type of geometric transformation which preserves collinearity (if a collection of points sits on a line before the transformation, they all sit on a line afterwards) and the ratios of distances between points on a line.

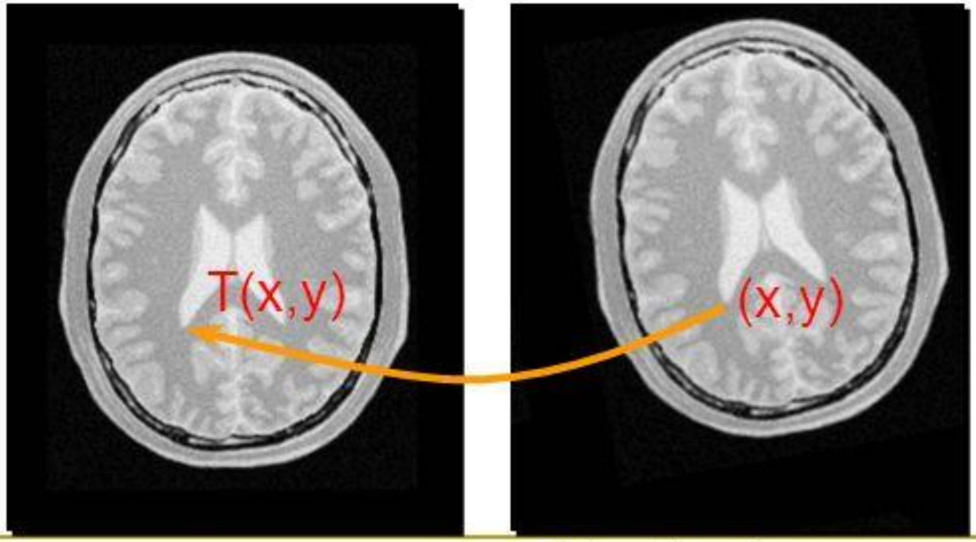
```python
# Affine Transformation Affine transformations preserve parallelism and ratios of distances.
#These can include translation, scaling, rotation, and shearing combined.
def affine_transform(image):
    rows, cols, _ = image.shape
    # Define three points for the original image
    pts1 = np.float32([[50, 50], [200, 50], [50, 200]])
    # Define the corresponding points in the transformed image
    pts2 = np.float32([[10, 100], [200, 50], [100, 250]])
    # Get the affine transformation matrix
    affine_matrix = cv2.getAffineTransform(pts1, pts2)
    # Apply the affine transformation
    affine_image = cv2.warpAffine(image, affine_matrix, (cols, rows))
    return affine_image

# Apply affine transformation
affine_image = affine_transform(image)

# Display the result
plt.imshow(affine_image)
plt.title('Affine Transformed Image')
plt.axis('off')
plt.show()
```

Affine Transformed Image


Original Image
Translated Image
Rotated Image
Scaled Image
Affine Transformed Image

Geometric Transformations

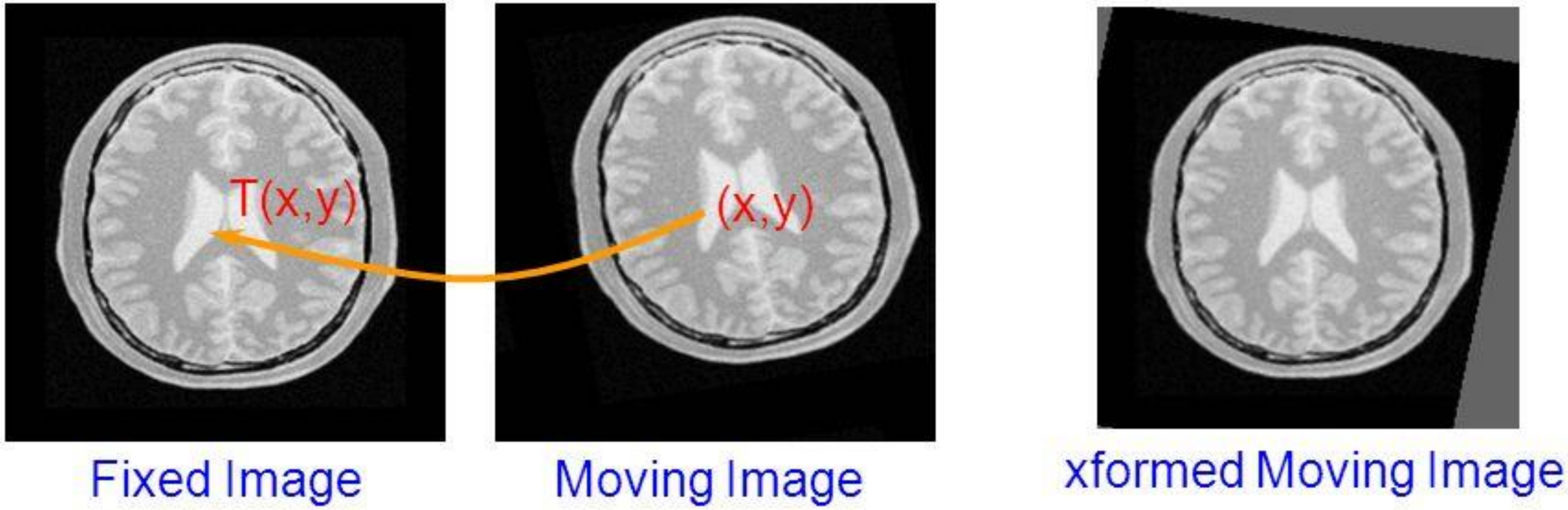
- A geometric transform consists of two basic steps ...
  - Step1: determining the pixel co-ordinate transformation
    - mapping of the co-ordinates of the moving image pixel to the point in the fixed image.



Fixed Image Moving Image

Geometric transformations

- Step2: determining the brightness of the points in the digital grid of the transformed image.
  - brightness is usually computed as an interpolation of the brightnesses of several points in the neighborhood.



We'll discuss step 2 first.

Affine Transformation

- An affine transformation maps variables (*e.g.* pixel intensity values located at position in an input image) into new variables (*e.g.* in an output image) by applying a linear combination of translation, rotation, scaling operations.
- **Significance:** In some imaging systems, images are subject to geometric distortions. Applying an affine transformation to a uniformly distorted image can correct for a range of perspective distortions.

Affine Transformation (con'd)

- By defining only the *B* matrix, this transformation can carry out pure **translation**:

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, B = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

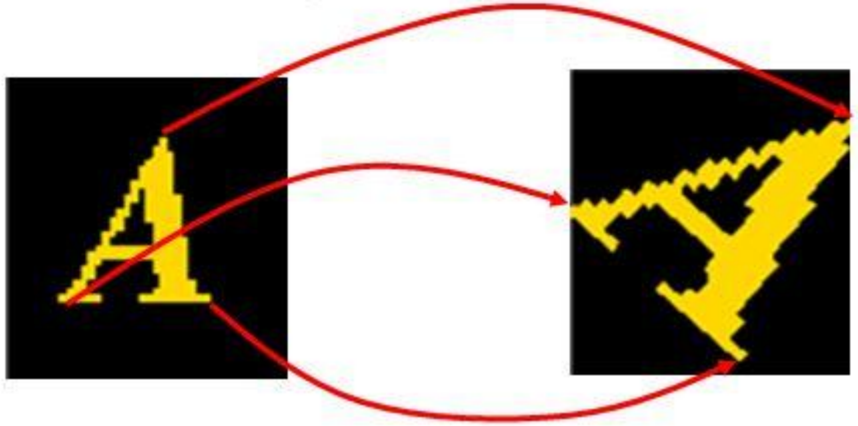
- Pure **rotation** uses the *A* matrix and is defined as (for positive angles being clockwise rotations):

$$A = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Affine Transformation (con'd)

- Pure **scaling** is defined as $A = \begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

- Since the general affine transformation is defined by 6 constants, it is possible to define this transformation by specifying 3 corresponding point pairs (more in next class).



Affine Transformation (con'd)

- An affine transformation is equivalent to the composed effects of translation, rotation and scaling, and shearing.
- The general affine transformation is commonly expressed as below:

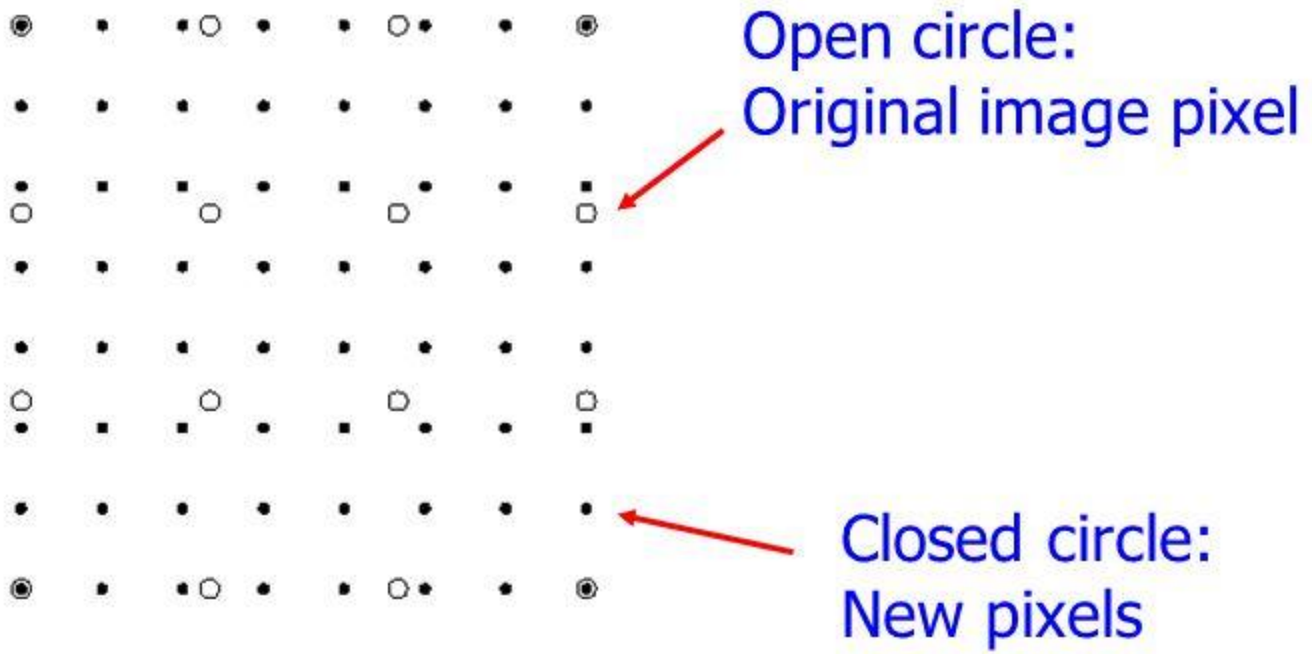
$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = A \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + B$$

0th order coefficients

1st order coefficients

Another Example

Interpolation on an image (4x4 -> 8x8) after scaling

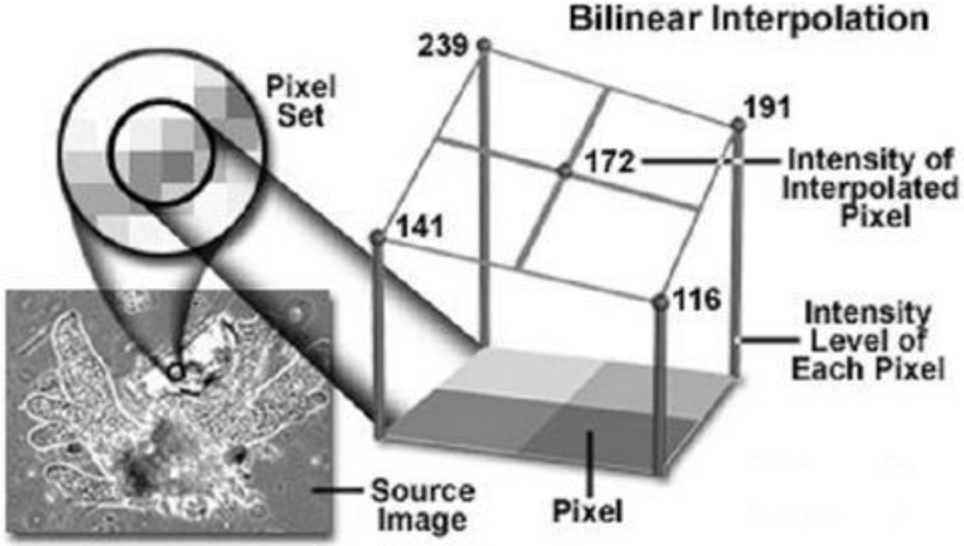


Bilinear Interpolation

- Substituting with the values just obtained:

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

- You can do the expansion as an exercise.
- This is the formulation for bilinear interpolation



Digital Image Processing Lecture 6: Image Geometry

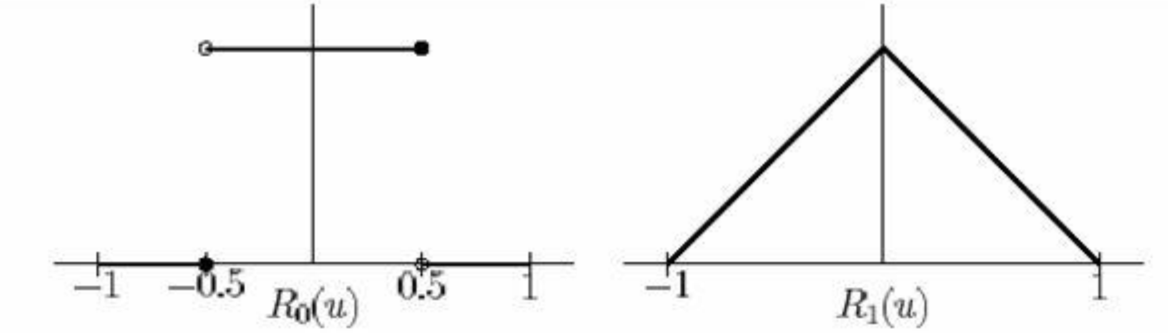
Prof. Charlene Tsai

Example

```
I = imread('cman.tif');
tform = maketform('affine',[1 0 0; .5 1 0; 0 0 1]);
J = imtransform(I,tform);
imshow(I), figure, imshow(J)
```

General Interloplation: 0^{th} and 1^{st} orders

- Consider 2 functions $R_0(u)$ and $R_1(u)$



$$R_0(u) = \begin{cases} 0 & \text{if } u \le -0.5 \\ 1 & \text{if } -0.5 < u \le 0.5 \\ 0 & \text{if } u > 0.5 \end{cases}$$

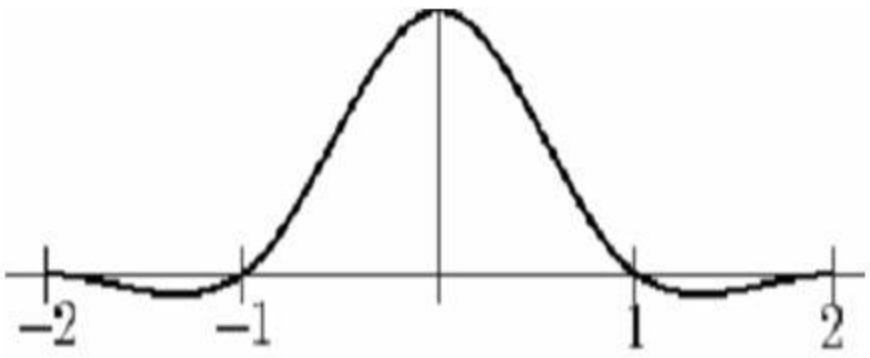
$$R_1(u) = \begin{cases} 1+u & \text{if } u \le 0 \\ 1-u & \text{if } u \ge 0 \end{cases}$$

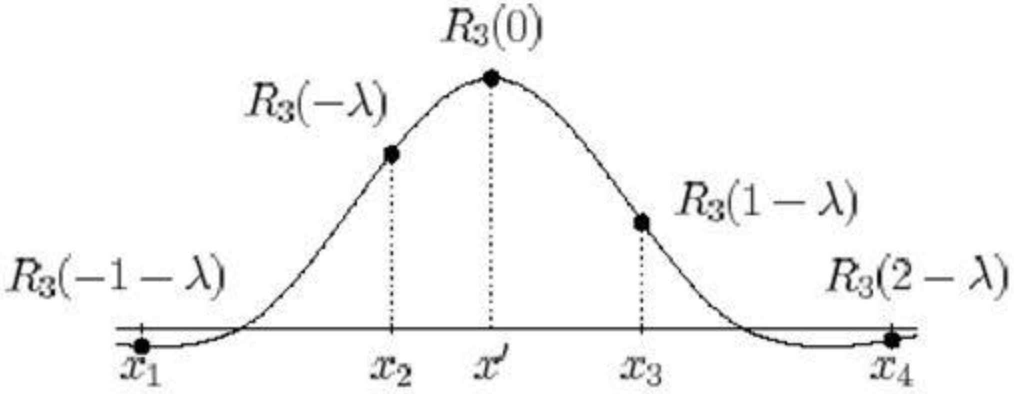
Substitute $R_0(u)$ for $R(u)$ ➡ nearest neighbour interpolation.

Substitute $R_1(u)$ for $R(u)$ ➡ linear interpolation.

General Interloplation: 3rd order (Cubic)

$$R_3(u) = \begin{cases} 1.5|u|^3 - 2.5|u|^2 + 1 & \text{if } |u| \le 1 \\ -0.5|u|^3 + 2.5|u|^2 - 4|u| + 2 & \text{if } 1 < |u| \le 2 \end{cases}$$



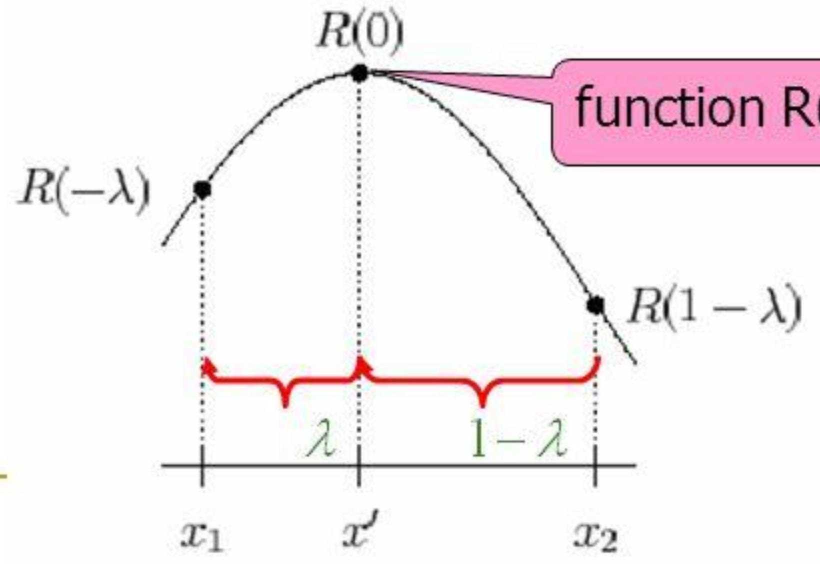


$$f(x') = R_3(-1-\lambda)f(x_1) + R_3(-\lambda)f(x_2) + R_3(1-\lambda)f(x_3) + R_3(2-\lambda)f(x_4)$$

General Interpolation

- We wish to interpolate a value f(x') for $x_1 \le x' \le x_2$ and suppose $x' - x_1 = \lambda$ ← $\boxed{0 \le \lambda \le 1}$
- We define an interpolated value R(u) and set

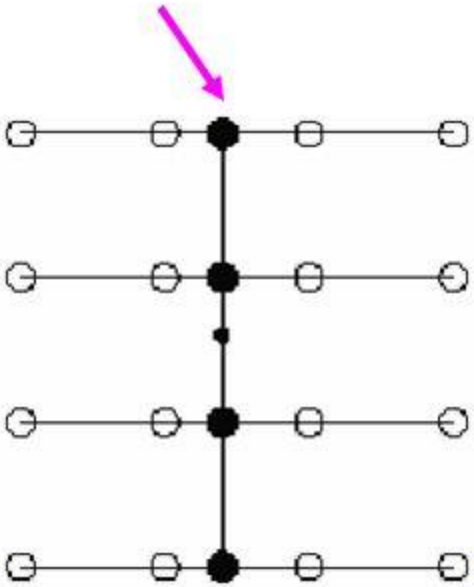
$$f(x') = R(-\lambda) f(x_1) + R(1-\lambda) f(x_2)$$



x_1 corresponds with u= - λ, and x_2 with u= 1- λ

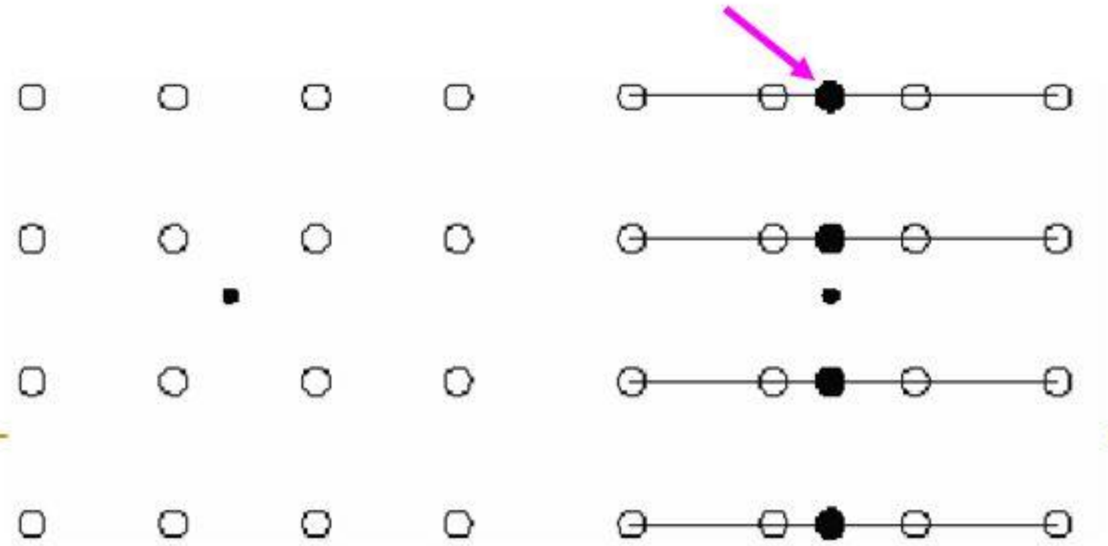
General Interpolation: Bicubic

- Step 2: the fractional part of the calculated pixel's address in the y-direction is used to fit another cubic polynomial down the column, based on the interpolated brightness values that lie on the curves F(i), i = 0, ..., 3.



General Interpolation: Bicubic (2D)

- Bicubic interpolation fits a series of cubic polynomials to the brightness values contained in the 4 x 4 array of pixels surrounding the calculated address.
  - Step 1: four cubic polynomials F(i), i = 0, 1, 2, 3 are fit to the control points along the rows. The fractional part of the calculated pixel's address in the x-direction is used.



General Interpolation: Example

- Original detailed part of flower image (8bit,75×75)
- Detailed part of super-resolution image (8bit,300×300) :

NN Interpolation

Bilinear Interpolation

Bicubic Interpolation

```
im = imread('flower.jpg');
im2= imresize(im,[800,800],method);
```

General Interpolation: Summary

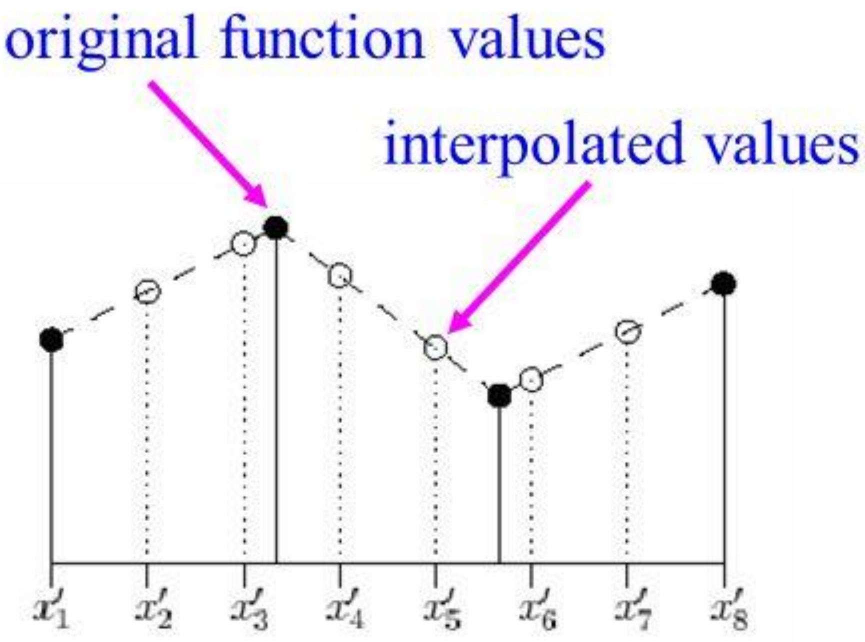
- For NN interpolation, the output pixel is assigned the value of the pixel that the point falls within. No other pixels are considered.
- For bilinear interpolation, the output pixel value is a weighted average of pixels in the nearest 2-by-2 neighborhood.
- For bicubic interpolation, the output pixel value is a weighted average of pixels in the nearest 4-by-4 neighborhood.
- Bilinear method takes longer than nearest neighbor interpolation, and the bicubic method takes longer than bilinear.
- The greater the number of pixels considered, the more accurate the computation is, so there is a trade-off between processing time and quality.

Geometric transformations

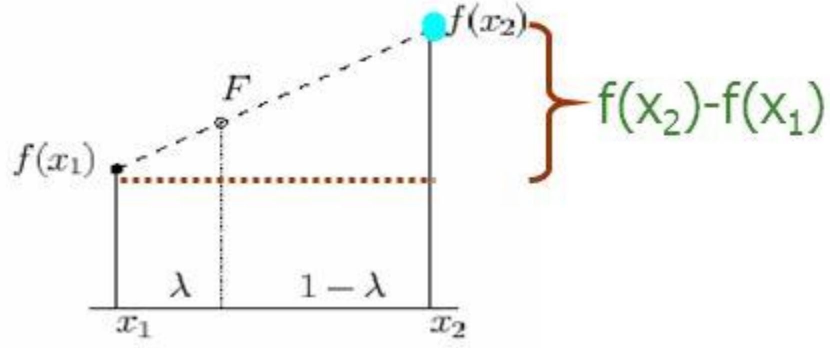
- Geometric transformations are common in computer graphics, and are often used in image analysis.
- Geometric transforms permit the elimination of geometric distortion that occurs when an image is captured.
- If one attempts to match two different images of the same object, a geometric transformation may be needed.
- Examples?

Interpolation: Linear (1D)

- General idea:



To calculate the interpolated values

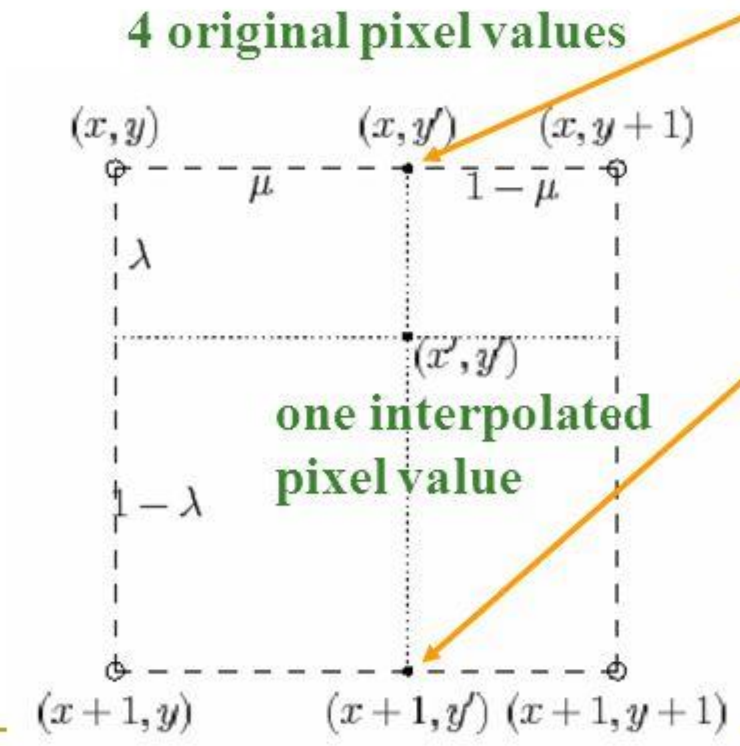


$$\frac{F - f(x_1)}{\lambda} = \frac{f(x_2) - f(x_1)}{1}.$$

Interpolation: Linear (2D)

- How a 4x4 image would be interpolated to produce an 8x8 image?

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



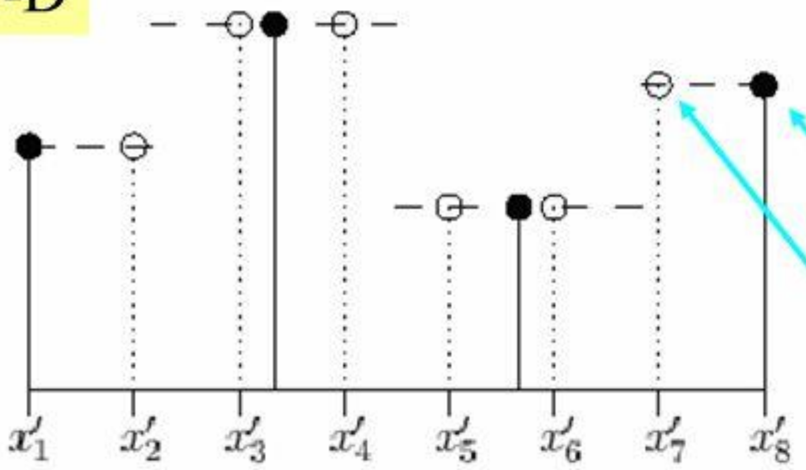
$$f(x+1, y') = \mu f(x+1, y+1) + (1-\mu) f(x+1, y)$$

Along the y' column we have

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

Interpolation: Nearest Neighbor

1-D



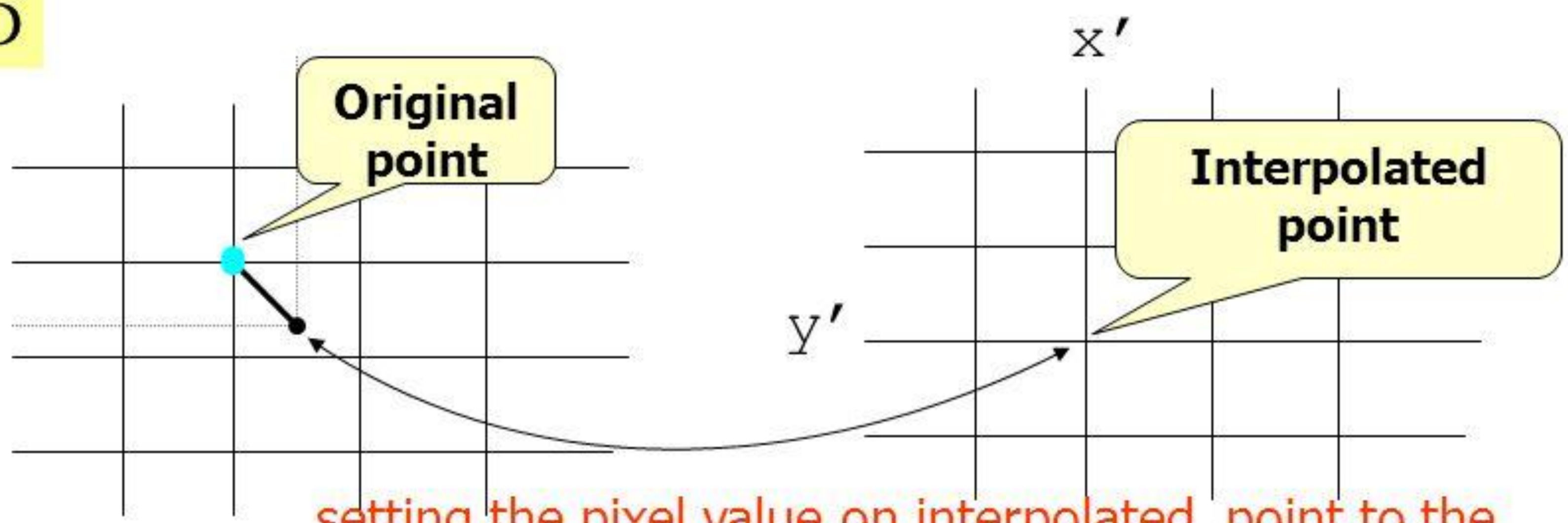
We assign $f(x_i') = f(x_j)$

x_j is the original point closest to x_i'

The original function values

The interpolated values

2-D



setting the pixel value on interpolated point to the pixel of closet image point

Matlab ***imtransform***

- The `imtransform` function accepts two primary arguments:
  - The image to be transformed
  - A spatial transformation structure, called a TFORM, that specifies the type of transformation you want to perform
- Specify the type of transformation in a TFORM structure.

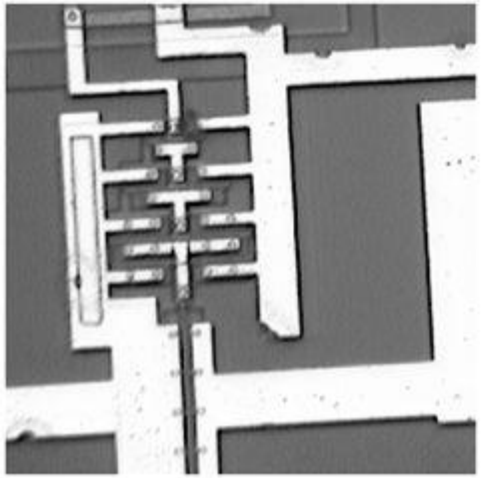
  Two ways to create a TFORM struct:
  - Using the maketform function
  - Using the cp2tform function

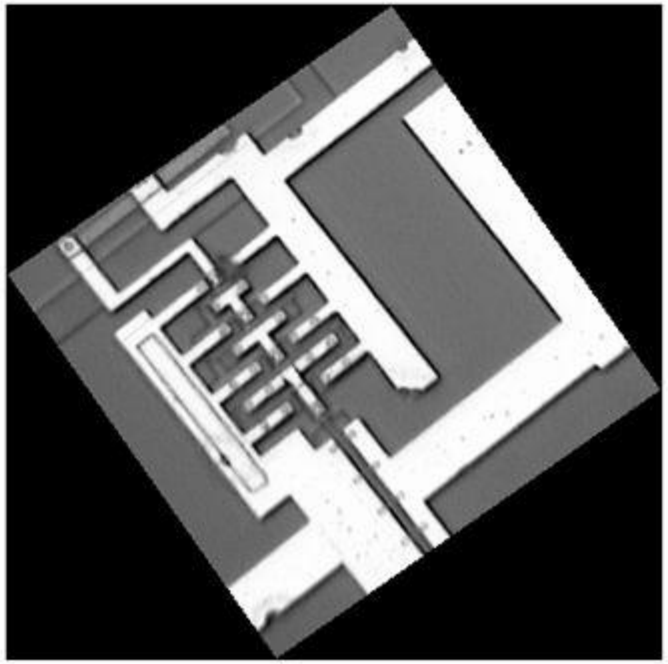
Rotation Operation

- A geometric transform which maps the position of a picture element in an input image onto a position in an output image by rotating it through an angle about an origin.
- Commonly used to improve the visual appearance of an image.
- Can also be useful as a pre-processor in applications where directional operators are involved.
- Rotation is a special case of affine transformation.

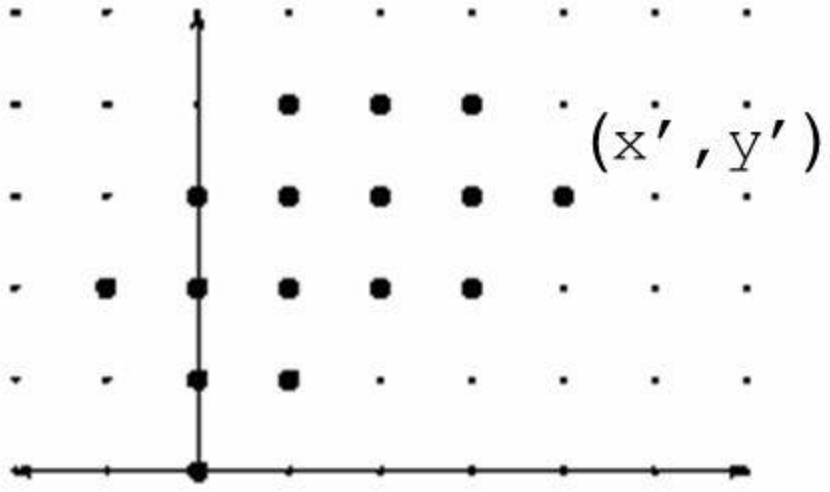
Rotation Operation: Example

```
I = imread('ic.tif');
J = imrotate(I,35,'bilinear');
imshow(I)
figure, imshow(J)
```



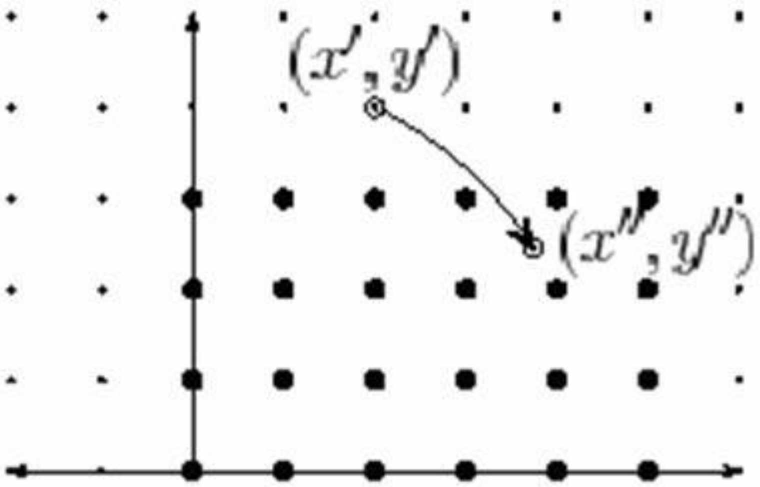


Rotation Operation: Remedies (con'd)



(x′,y′) in rotated image

(x",y") is the rotated (x′,y′) back into the original image



- The grey value at (x",y") can be found by interpolation.
- This value is the grey value for the pixel at (x′,y′) in the rotated image.

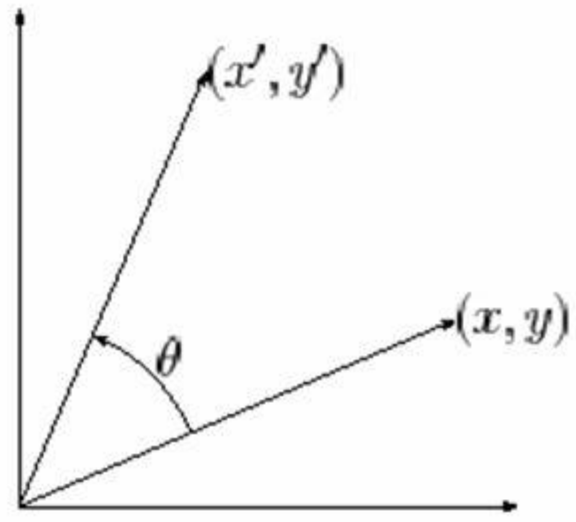
Rotation Operation (cont)

- Mapping of a point (x,y) to another (x',y') through a counter-clockwise rotation of θ

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

$$\Downarrow$$

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



Scaling Operation

- To shrink or zoom the size of an image (or part of an image).
- To change the visual appearance of an image;
- To alter the quantity of information
- To use as a low-level pre-processor in multi-stage image processing chain which operates on features of a particular scale.
- Scaling is a special case of *affine transformation.*
- The matlab command is simply “imresize”.

Summary

- Interpolation of intensity values on non-grid points:
  - Nearest Neibhgor (NN)
  - Bilinear
  - Bicubic
- Image transformation
  - Computation of intensity values of the transformed image
  - Discussed some instances of affine transformation
    - Translation
    - Rotation
    - Scaling

Using *maketform*

- When using the maketform function, you can specify the type of transformation, e.g
  - 'affine'
  - 'projective'
  - 'composite', et al
  - 'custom' and 'composite' capabilities of maketform allow a virtually limitless variety of spatial transformations to be used
- Once you define the transformation in a TFORM struct, you canperform the transformation by calling imtransform.