# CSC317 Digital Image Processing

# Class Note One - Basics

## Roadmap

1. Administration – Syllabus, Schedule, Roster (Name)
2. Math Needed for CSC317
3. Basics about Images
4. Installation of Anaconda Python 3.7, OpenCV
5. (Next Week) Python Programming Basics

## I. Administration

We will use the sequence in the outline file to cover the first part of a computer vision course – digital image processing. There are three parts for this course. Part I covers the basics about images and neighborhood processing (pixel-level processing), and geometric transformations. Part II covers the local image processing methods including convolutions, spatial-domain filters, and Fourier transforms & frequency-domain filters. Part III covers image analysis with morphological transformations, gradient operators, Laplacian operators, Hough Line Detection, and Image Segmentations.

Our class includes about 12-13 lectures(modules), three quizzes/exams, three homework assignments with programming problems and computational problems. For each *lecture*, i.e., a *D2L module*, we will begin with about concepts in the area of digital image processing and analysis, followed by python programming method. For the lecture, we will use materials from several resources. The textbook is the main resource for the lectures. Programming will be done with Anaconda Python and OpenCV. If you prefer C++, you may use it to complete all programming problems in the assignment. However, the lecture will only focus on python and OpenCV. When we refer to OpenCV, we are not limited to learning OpenCV. We will also use some other packages such as Numpy, Skimage, etc.

## III. Basics about Images and Computer Vision

* Computer Vision integrates techniques from (EE) Digital Signal Processing, (MATH) Computer Geometry, (CS) Computer Graphics, and (CS) Machine Learning.
* In CSC317, we will focus on Digital Image Processing (Histogram, Convolution, and Gradient) as well as basics about Image Geometry (Scaling, Translation, Rotation, and Shearing), Morphology (Dilations, Erosions, Opening, and Closing), Segmentation and Line Detection.

(Section Numbers and contents are taken from the textbook.)

### 1.1 Images and Pictures

For our purpose, an image is a picture with foreground and background. Foreground is representing objects, when background represents the environment.

### 1.2 What is Image Processing?

## Image Processing involves changing the nature of an image for the following purposes:

1. Improve its pictorial information for human interpretation
2. Render it more suitable for autonomous machine perception

We will use a computer to change the nature of digital images. Here are some examples:

The left one is an original image of an alley with two pedestrian; the right one is a sharpened image.

Figure 1.1 Sharpening an image

The left one is a building with sale-and-pepper noises; the right one has dee-noised.

Figure 1.2 Remove noises

The left car has a fuzzy license plate; the right one has a clear license plate.

Figure 1.3 Removing the blur on the license plate

The left one has bricks and cylinders stacked together; the right image only shows the edges of these objects.

Figure 1.4 Display edges of bricks and cylinders

## THe left image is a buffalo; the right one uses blurring to remove the detail.

Figure 1.5 Blurring to remove details

### 1.3 Image Acquisition and Sampling - Issues

* CCD (Charged-Coupled Device) – Camera (will be discussed later)
* Flatbed Scanner – CCD Scanner
* X-Ray or MRI – Energy Sources

We will look into the CCD Camera model next week. We will discuss about Sampling today. Sampling refers to the process of digitizing a continuous function. For example, we sample the values of y for a range of given x values using the function:

y = sin(x) + (1/3)sin(3x)

If we take not enough samples, the shape of the function may be distorted. We call the situation as *under-sampling*. We can use R programming IDE R-Studio to draw the curve:

curve(sin(x)+(1/3)\*sin(3\*x), from =0, to=10, n=20)

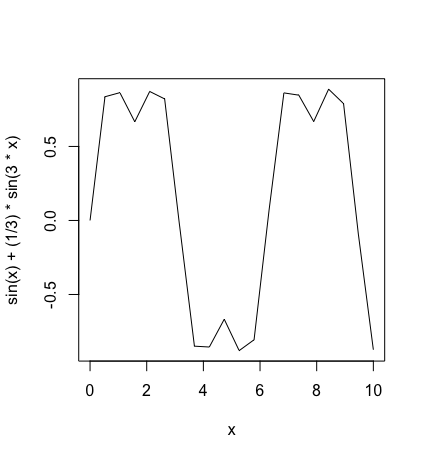


Fig. 1.6 Under Sampling (n=20)

Increased sample size (n=200)

Description automatically generated

Fig. 1.7 Increased sampling size

To improve the result of sampling, we need to increase the sampling size to 200 samples:

curve(sin(x)+(1/3)\*sin(3\*x), from =0, to=10, n=200)

Q: What is the difference between two sampled images?

A: The file size is enlarged. We will talk about CCD Camera next week.

## 1.4 Images and Digital Images

### Terms about Image Basics

* **Image *I***: A (digital) image is defined by integrating and sampling continuous (analog) data in a spatial domain. Each image consists of a rectangular array of pixels that are the “atomic elements” of an image.
* **Pixel**: A pixel is a triplet *(x, y, u)* in a 2D domain, each combining a *location* *(x, y)* ∈ Z2 and a value *u*, the *sample* at the pixel location *(x, y)*. We can represent the sampling at a pixel location as a function *p* *that maps the pixel location (x,y) to an integer u,* *p(x,y) = u.*
* **Carrier *Ω***: The rectangular set of pixel locations

***Ω*****= {*(x, y)* : 1 ≤ *x* ≤ *Ncols* ∧ 1 ≤ *y* ≤ *Nrows* } ⊂ Z2**

of *I* containing the *grid points or pixel locations for Ncols ≥ 1 and Nrows ≥ 1.*

* **Window *Wm,n\_p*(*I*):** A Window*Wm,n\_p (I)* is a sub-image I of size m x n positioned with respect to a reference point p (i.e., a pixel location). The default is that m = n is an odd number, and p is the central location in the window.
* **Channel:** one dimension of the pixel values. A *vector-valued image* has more than one channel or band, as it is the case for scalar images. Image values *(u1, . . . , uNchannels* ) are vectors of length *Nchannels*. For example, color images in the common RGB color

model have three channels, one for the red component, one for the green, and one for

the blue component. The values *ui* in each channel are in the set {*0, 1, . . . , Gmax*};

each channel is just a gray-level image.

A *digital image* differs from a *photo* in that the x, y, and f(x, y) (for the grayscale value) are all discrete. *Black is represented by 255, and white is represented by 0*. Also, in OpenCV, *white* objects are always considered *foreground*, and *black* pixels are referred to as *background*. In other words, a digital image in OpenCV consists of white objects on a black background.

### 1.7 Tasks or Design Issues about Image Processing/Computer Vision

1. Image Acquisition – Part I of this course
2. Image Preprocessing(Pixel, Local, and Global) – Part II of this course
3. Image Analysis (Segmentation) – Part III of this course
4. Recognition and Interpretation – future Computer Vision course.

### 1.8 Types of Images

* **Scalar image:** For each pixel, p(x,y) = v, where v: 0 to 255. Note, **white = 255, black = 0.**
* **Binary image:** For each pixel, p(x,y) = black or white. Traditionally, black is used for the foreground and white is used for the background. So, p(x, y) = 1 if black; 0 if white. However, *OpenCV always uses white for foreground, and black for background.* The concept will be critical when we discuss about mathematical morphology. Some textbook still uses 1 for white and 0 for black.
* **Vector-Valued or RGB image:** For each pixel, there are three channels, i.e., Red, Green, and Blue. A vector-valued image has more than one *channel* or *band*, as it is the case for scalar images. For example, color images in the common RGB color model has three channels, i.e., Red, Green, and Blue channels.

## 1.9 Image File Size

* Binary Image: 512 x 512 x 1 = 262,144 bits = 32,768 bytes = 0.033 MB
* Gray Scale: 512 x 512 x 1 = 262,144 bytes = 0.262 MB
* Color: 512 x 512 x 3 = 0.786 MB

# CSC317 Digital Image Processing

# Lecture 4 Point Process

## 4.1 Introduction

A screenshot of a cell phone

Description automatically generated

Fig. 4.1 Schema for Transform Processing

There are *three* classes of image processing:

1. Point Processes – A pixel’s grayscale value is changed without the knowledge of the surrounding. A histogram stretching or equalization is a typical example.
2. Local Processes – Neighborhood processing using a kernel/operator/filter. Linear or nonlinear kernels are used.
3. Global Transformation – A “transform” represents the pixel values in some other, but equivalent, form. Examples include Geometric Transformations, Fourier Transforms.

## 4.2 Arithmetic Operations

We will begin from the simplest, i.e., a Point Process to adjust the contrast of images. Before that, we are going to take a look at some arithmetic operations with the grayscale value, e.g., adding and subtraction some values.

k

255-n

255-k+1 n+1

Fig. 4.2 Adding Fig. 4.3. Subtracting

These operations can be represented by the following mapping:

y= ∈[0 .. 255] and y ∈[0 .. 255]

Examples:

Y = f(x) = x + C or x – C # C is a constant

Y = C x # y <- 255 if y > 255; y<- 0 if y < 0

In Fig. 4.2,

new value = (the original value + k) mod 256, (0 <= k <= 255)

In Fig. 4.3,

new value = (the original value + 255 – n) mod 256 ( 0<= n <=255)

In summary, with these simple operations we have applied, the contrast was not improved; only the brightness or darkness was changed. What we may want is to *increase* the contrast as the objects in Figure 4.8; or *decrease* the contrast of the objects inside an image, e.g., improve the contrast, of an image in Figure 4.9. We just call it “*contrast enhancement*”.

There are two ways to perform contrast enhancement: histogram stretching and histogram equalization. In the following sections, we will explain the principles and demonstrate with programs.

## 4.3 Histogram Stretching (Contrast Stretching) (P. 74 – P. 77)

The histogram



Figure 4.10: Image of wo chickens and the histogram

In Figure 4.10, there are two chickens: one is extremely white, and another is extremely dark. We can show the total number of pixels with various gray levels using a histogram. In this

The *histogram* of an image is *a graph indicating the number of pixels with specific gray levels in a given image*. We can apply *Histogram Stretching* to an image if a stretching function is given. Otherwise, we need to apply *Histogram Equalization* to change the contrast of an image.

## Histogram Stretching (P. 73 – P. 74)

In this section, we use simpler examples to illustrate how Histogram Stretching is completed with a given shtretching range of gray levels from one range to another.

The original gray levels and the number of pixels with each gray level is shown in the table below:

i ni

|  |  |
| --- | --- |
| 0 | 15 |
| 5 | 70 |
| 6 | 110 |
| 7 | 45 |
| 8 | 70 |
| 9 | 35 |
| 15 | 15 |



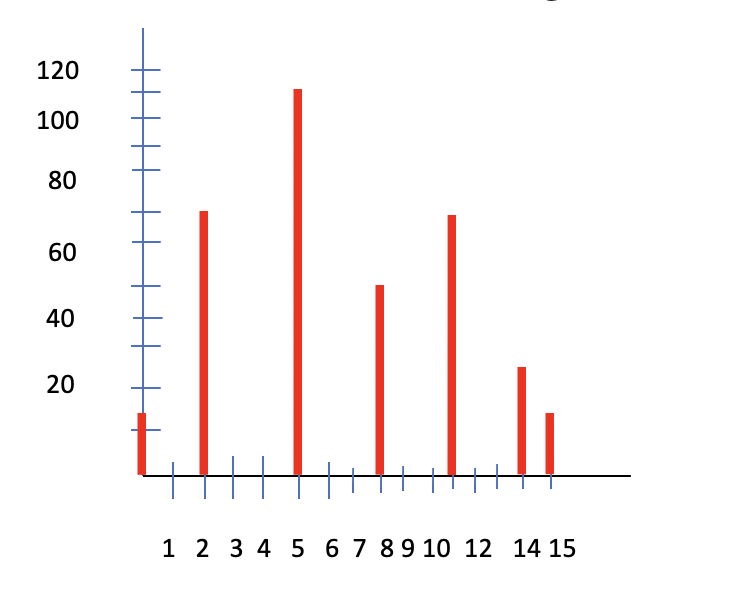
Fig. 4.11 Before Fig. 4.12 Stretching Function

Fig. 4.13 After Stretching

The problem of Histogram Stretching requires the background about finding the equation of a straight line. Usually, the stretching is given as ranges. For example, the example stretches the range of gray levels falling in a – b to c – d as indicated in Fig. 4.12 where the range of 5 – 9 is stretched to the range of 2 - 14.

### Procedure to Find the Mapping Function

The procedure to find the equation for the straight line from the point (a, c) to (b, d) is illustrated below:

**Step #1** Find the slope m of the straight line.

*m* = (d-c)/(b-a) = 3

**Step#2** Use the point (a, c) and the slope m to represent the equation of the straight line as

*m* = (y-c)/(x-a)

**Step #3** Simplify the above equation:

y - c = *m* (x – a) or y = *m* (x – a) + c

y = 3 (x – 5) + 2

x = 5, 6, 7, 8, 9 => y = 2, 5, 8, 11, 14

(This is the solution before drawing the final Histogram as in Fig. 4.13)

*(There is an error in Fig. 4.13 of our textbook: y = 5 instead of 4 when x = 6)*

Now, you may take a look at the effects of the Histogram Stretching. A range of gray levels of 5 to 9 is changed to the range of 2 to 14. The gray level values are spread wider than the original gray levels. That simply “stretches out” the “contrast” of pixel gray values.

## In summary

If we provide a function to map one image to another with the goal of enlarging the range of gray levels (defined by Graylevelmax – Graylevelmin), this operation is called a *Histogram Stretching*. If the operation of image stretching is based on the function of a uniform distribution, the operation is called a *Histogram Equalization*.

### EXAMPLES

**Example #1** Shift the grayscale value ranges from 1-8 to 1-20

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 2 | 4 | 5 |
| 7 | 7 | 8 | 2 |
| 3 | 1 | 2 | 3 |
| 5 | 4 | 6 | 7 |

Method #1 Histogram Stretching

Derive a mapping function

Slope (m) = (20-1)/(8-1) = 19/7 = 2.7

Equation for the mapping function: m = (y-1)/(x-1)

y-1 = m (x – 1)

y = m (x -1) + 1

|  |  |  |
| --- | --- | --- |
| Old value | New value | rounded |
| 1 | 1 | 1 |
| 2 | 3.7 | 4 |
| 3 | 6.4 | 6 |
| 4 | 9.1 | 9 |
| 5 | 11.8 | 12 |
| 6 | 14.5 | 15 |
| 7 | 17.2 | 17 |
| 8 | 19.9 | 20 |

The grayscale vaues of the original image will become:

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 4 | 9 | 12 |
| 17 | 17 | 20 | 4 |
| 6 | 1 | 4 | 6 |
| 12 | 9 | 15 | 17 |

Method #2 Applying Histogram Equalization

When the stretch function is not given, we use Histogram Equalization.

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 2 | 4 | 5 |
| 6 | 6 | 8 | 2 |
| 3 | 1 | 2 | 3 |
| 5 | 4 | 6 | 6 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Intensity | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| # Pixels | 1 | 3 | 3 | 2 | 2 | 1 | 3 | 1 |
| prob | 0.0625 | 0.1875 | 0.1875 | 0.125 | 0.125 | 0.0625 | 0.1875 | 0.0625 |

Total # of pixels = 4 x 4 = 16

Cumulative Probability (C.P.)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0.0625 | 0.25 | 0.4375 | 0.5625 | 0.6875 | 0.75 | 0.9375 | 1 |
| C.P.\*20 | 1.25 | 5 | 8.75 | 11.25 | 13.75 | 15 | 18.75 | 20 |
| New Intensity | 1 | 5 | 8 | 11 | 13 | 15 | 18 | 20 |

The grayscale values become:

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | 5 | 11 | 13 |
| 18 | 18 | 20 | 5 |
| 8 | 1 | 5 | 8 |
| 13 | 11 | 15 | 18 |

**Example #2 *Histogram Equalization***

Given an image with the following ***grayscale levels (i)*** and ***the number of pixels*** with the respective grayscale levels ***(ni*),** apply a Histogram Equalization

i = 0 1 2 3 4 5 6 7 8 9 10

ni= 10 0 0 0 0 20 20 30 10 10 0

ni 10 10 10 10 10 30 50 80 90 100 100

ni xRatio 1 1 1 1 1 3 5 8 9 10 10

Note: Ratio = grayscale levels / total number of pixels = 10/100 = 0.1



1 2 3 4 5 6 7 8 9 10

40

30

20

10



1 2 3 4 5 6 7 8 9

40

30

20

10

## CV Programming - Histograms (Taken from the OpenCV Tutorial)

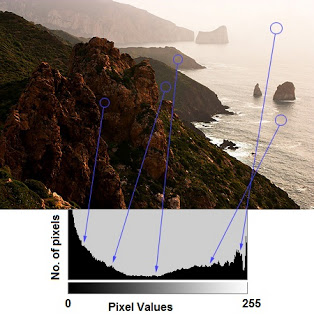
### Goals

1. Find histograms, using both OpenCV and Numpy functions
2. Plot histograms, using OpenCV and Matplotlib functions
3. You will see these functions : **cv2.calcHist()**, **np.histogram()** etc.

### Theory

So, what is a histogram? You can consider a histogram as a graph or plot, which gives you an overall idea about the intensity distribution of an image. It is a plot with pixel values (ranging from 0 to 255, not always) in X-axis and corresponding number of pixels in the image on Y-axis.

It is just another way of understanding the image. By looking at the histogram of an image, you get intuition about contrast, brightness, intensity distribution etc. of that image. Almost all image processing tools today, provides features on histogram. Below is an image from [Cambridge in Color website](http://www.cambridgeincolour.com/tutorials/histograms1.htm), and I recommend you to visit the site for more details.



You can see the image and its histogram. (Remember, this histogram is drawn for grayscale image, not color image). Left region of histogram shows the number of darker pixels in image and right region shows the number of brighter pixels. From the histogram, you can see dark region is more than brighter region, and the amount of mistunes (pixel values in mid-range, say around 127) is very few.

### Find Histogram

Now we have an idea on what is histogram, we can look into how to find this. Both OpenCV and Numpy come with in-built function for this. Before using those functions, we need to understand some terminologies related with histograms.

**BINS**: The above histogram shows the number of pixels for every pixel value, i.e. from 0 to 255, i.e. you need 256 values to show the above histogram. But consider, what if you need not find the number of pixels for all pixel values separately, but number of pixels in an interval of pixel values? say for example, you need to find the number of pixels lying between 0 to 15, then 16 to 31, ..., 240 to 255. You will need only 16 values to represent the histogram. And that is what is shown in example given in [OpenCV Tutorials on histograms](http://docs.opencv.org/doc/tutorials/imgproc/histograms/histogram_calculation/histogram_calculation.html#histogram-calculation).

So, what you do is simply split the whole histogram to 16 sub-parts and value of each sub-part is the sum of all pixel count in it. This each sub-part is called “BIN”. In first case, number of bins where 256 (one for each pixel) while in second case, it is only 16. BINS is represented by the term **histSize** in OpenCV docs.

**DIMS**: It is the number of parameters for which we collect the data. In this case, we collect data regarding only one thing, intensity value. So here it is 1.

**RANGE**: It is the range of intensity values you want to measure. Normally, it is [0,256], ie all intensity values.

### Histogram Calculation in OpenCV

So now we use **cv2.calcHist()** function to find the histogram. Let’s familiarize with the function and its parameters:

**cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])**

1. images: it is the source image of type uint8 or float32. it should be given in square brackets, i.e., “[img]”.
2. channels: it is also given in square brackets. It is the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is [0]. For color image, you can pass [0], [1] or [2] to calculate histogram of blue, green or red channel respectively.
3. mask: mask image. To find histogram of full image, it is given as “None”. But if you want to find histogram of particular region of image, you have to create a mask image for that and give it as mask. (I will show an example later.)
4. histSize: this represents our BIN count. Need to be given in square brackets. For full scale, we pass [256].
5. ranges: this is our RANGE. Normally, it is [0,256].

So, let’s start with a sample image. Simply load an image in grayscale mode and find its full histogram.

img = cv2.imread('home.jpg',0)

hist = cv2.calcHist([img],[0],None,[256],[0,256])

hist is a 256x1 array, each value corresponds to number of pixels in that image with its corresponding pixel value.

### Histogram Calculation in Numpy

Numpy also provides you a function, **np.histogram()**. So instead of calcHist() function, you can try below line :

hist,bins = np.histogram(img.ravel(),256,[0,256])

hist is same as we calculated before. But bins will have 257 elements, because Numpy calculates bins as 0-0.99, 1-1.99, 2-2.99 etc. So final range would be 255-255.99. To represent that, they also add 256 at end of bins. But we don’t need that 256. Up to 255 is sufficient.

**See also**

Numpy has another function, **np.bincount()** which is much faster than (around 10X) np.histogram(). So, for one-dimensional histograms, you can better try that. Don’t forget to set minlength = 256 in np.bincount. For example, hist = np.bincount(img.ravel(),minlength=256)

**Note**

OpenCV function is faster than (around 40X) than np.histogram(). So, stick with OpenCV function.

Now we should plot histograms, but how ?

#### 2. Using OpenCV

Well, here you adjust the values of histograms along with its bin values to look like x,y coordinates so that you can draw it using cv2.line() or cv2.polyline() function to generate same image as above. This is already available with OpenCV-Python2 official samples. [Check the Code](https://github.com/Itseez/opencv/raw/master/samples/python2/hist.py)

### Application of Mask

We used cv2.calcHist() to find the histogram of the full image. What if you want to find histograms of some regions of an image? Just create a mask image with white color on the region you want to find histogram and black otherwise. Then pass this as the mask.

**The material in this lecture note was taken from Chapter 6 of the reference book; and the OpenCV tutorial.**

*Geometric Transformation* means transforming a geometric entity (e.g. point, line, object) to another. This can happen in any dimension. For example, a 2D image can be transformed to another with translation, rotation, scaling, or shear.

## II. Homogeneous Coordinates

Before we discuss about Geometric Transformation, we need to take a look at an important concept about coordinates. Often, we believe the claim, “If two lines that are in parallel, they will never intersect.” However, in the world of Computer Vision, a close view from the person standing in the middle of a railway track shows that the two lines that form the railway will intersect. How can we represent this concept with a regular x- and y-axis?

When we begin to learn Geometry, we learn how to represent the location in Cartesian Coordinates. For a 2D position, we use (x, y) to represent a point, for a 3D position, we use (x, y, z) to represent a point. However, in the Cartesian world, two lines in parallel will not intersect with each other.

As matter of fact, in the world of Computer Vision, a 3D object will be viewed as a 2D image on the retina of the eye. Our brain identifies each point of the image in the retina as a ray into the 3D world. If we stand in the middle of two rails of a railway track, the two rails that are in parallel will intersect at the far end. The introduction of Homogeneous Coordinates solves the issue.

We use the concept of Homogeneous Coordinates to represent a point in a 2D world (x, y) as (x, y, 1). In general, we use (x/w, y/w) in Cartesian Coordinates to represent (x, y, w) in Homogeneous Coordinates.

Cartesian Coordinates Homogeneous Coordinates

1D: (3) (3, 1)

2D: (4, 6) => (4, 6, 1)

3D: (4, 6, 8) (4, 6, 8, 1)

Converting from the Cartesian space to the Homogeneous space, we just add a constant “1” as another parameter. It does not change the “dimension” of the point.

The 2D points (8, 12), (4, 6), (2, 3) in Cartesian Coordinates can be connected with a straight line. The Homogeneous Coordinates (8, 12, 4), (4, 6, 2), and (2, 3, 1) are represented as (2, 3) in the Cartesian (or Euclidean) space. That is exactly why we call these points “homogeneous” in the Homogeneous Space.

Cartesian Coordinates Homogeneous Coordinates

1D: (1/3) 2D: (1, 3)

2D: (2/6, 4/6) <= 3D: (2, 4, 6)

3D: (2, 4, 6) 4D: (4, 8, 12, 2)

2D: (x/w, y/w) 3D: (x, y, w)

2D: () 3D: (x, y, 0)

For a point at the far end of the railway track, we can use (x, y, 0) in Homogeneous Coordinates or ( in Cartesian Coordinates. The value of infinity is derived from x/0 and y/0. In the case of the intersection point of the two lines of a railway, we can represent the point as (0, 0, 0). Using the homogeneous coordinate to represent a 3D point (x, y, z) as a 4D point (x, y, z, 1). For the point at infinity, we represent it as (x, y, z, 0) where (x, y, z) is the direction of the point from the origin.

The reason we introduce the concept of Homogeneous Coordinates before talking about geometric transformations is because **we can represent the geometric operations as matrix and vector multiplications**. **Without using Homogeneous Coordinates, we can only represent the operations with systems of equations without using “transformation matrices”.**

In the rest of the chapter, we will represent points or vectors as 4 X 1 column vectors. Therefore, a 3D Cartesian point P = (x, y, z) will be written in the Homogeneous space as

x

y

P = z

1

Note: We always use a constant “1” to convert from Cartesian space to Homogeneous space.

## III. Linear Transformations

Linear transformation is a special kind of transformation. Given two points P and Q, the transformation L is considered a linear transformation if

L (aP + bQ) = a x L(p) + b x L(Q)

There are three types of linear transformations: *Euclidean, affine*, and *projective (or Perspective, or Homograph)*.

1. *Euclidean* – preserves length and angles.

A square cannot be changed to a rectangle by an Euclidean transformation. Translation and rotation are Euclidean transformations.

1. *Affine* – preserves the ratios of lengths and angles. A square can be transformed to a rectangle or rhombus by an affine transformation but cannot be formed to a general quadrilateral. Examples of affine transformations are shear and scaling. This means that parallel lines will remain parallel and intersecting lines will remain intersecting with Euclidean or affine transformations.

Based on the description online at the [Wikipedia website](https://en.wikipedia.org/wiki/Affine_transformation), an affine transformation preserves the following:

1. [**collinearity**](https://en.wikipedia.org/wiki/Collinearity) between points: three or more points which lie on the same line (called collinear points) continue to be collinear after the transformation.
2. [**parallelism**](https://en.wikipedia.org/wiki/Parallel_(geometry)): two or more lines which are parallel, continue to be parallel after the transformation.
3. [**convexity**](https://en.wikipedia.org/wiki/Convex_set)**of sets**: a convex set continues to be convex after the transformation. Moreover, the [extreme points](https://en.wikipedia.org/wiki/Extreme_point) of the original set are mapped to the extreme points of the transformed set.
4. The **ratios of lengths of parallel line segments** are preserved.
5. *Projective/Perspective transformation* – parallel lines become non-parallel. An example is shown in Fig. 6.2 of the handout. The projective transformation of a camera captured image and the relevant vanishing points.

In the following, we represent the concepts of geometric transformations with matrix forms.

## Euclidean and Affine Transformations

### Translation: P (x, y) -> P’ (x’, y’)

X’ = x + tx

Y’ = y + ty

x’ x + tx 1 0 tx x

P’ = Y’ = y + ty = 0 1 ty y = T (tx, ty) x P

1 1 0 0 1 1

Note: The last row is always (0 0 1) in the transformation matrix due to the use of Homogeneous Coordinates. As we mentioned previously, a translation operation can be represented as a matrix multiplication with a vector. The matrix T (tx, ty) is known as the “transformation matrix” of translation. There are two parameters that can change in the matrix. Thus, we sometimes call that the “*degree of freedom*” of a *translation operation* is 2.

### Rotation – We will illustrate this later in this lecture note

Section 6.3.2 of the reference book & the OpenCV Tutorial

### Scaling - Section 6.3.3 of the reference book & the OpenCV Tutorial

We will discuss the scaling operation later in the programming section.

### Shear – Section 6.3.4 of the reference book & the OpenCV Tutorial

### Shearing is to change the value of a point (in a 2D space) (x, y) only in one direction based on another dimension. For example, the following is a shear in the x direction. It is also called y-shearing. Intuitively, the original shape is shifted toward x-axis without changing y.

We can represent the transformation of y-shearing as follows:

X’ = x + a y

Y’ = y

(0,1)

(1,1)

(0,0) (1,0)

## Before shearing

If a = 0.5,

(0, 1) -> ( 0.5, 1) (x’= x + a; y=0 + 0.5= 0.5, y’= y = 1)

(1, 1) -> (1.5, 1)

(1, 0) -> (1, 0)

(0, 0) -> (0, 0)

(1,1)

(0,1)

(0,0) (1,0)

After shearing in the x director with the sharing parameter 0.5, the square turned into a “tilted” square. Note that the parallel lines are still parallel. *Shearing* is an example of *affine transformations* because the shape may change but the parallelism of lines is kept.

### Exercise

Draw a shearing in the y direction with the parameter of 0.5 using the original shape of a square.

## (Summary) Linear Transformations

L(aP+bQ) = a L(P) + b L(Q)

1. Euclidean Transformations: translation, rotation

Euclidean transformations preserve the lengths and angles.

1. Affine Transformations: scaling, shearing

Affine transformations preserve the ratios of lengths and angles.

## Some Observations

1. Euclidean transformations are a subset of affine transformations.
2. Euclidean transformations (translation and rotation) are called rigid body transformations that retain the shape of the original object.
3. Affine (shearing and scaling) can change the shape of the original object, but the parallel lines are still in parallel.
4. Prospective or perspective transformations change the shape, the parallelism of lines.
5. The translation matrix cannot be expressed as a 3 x 3 matrix multiplication with a vector in Cartesian Space, while scaling, rotation, or shear can be. We need to use the Homogeneous Space to represent the translation matrix in matrix and vector multiplication.
6. For rotation around any point other than the origin, we also need to use the homogeneous representation for the transformation matrix.

## VI. Chapter 6 (Textbook) Image Geometry

The material hereto after was taken from Chapter 6 in our textbook. OpenCV Tutorial does not cover this part about Data Interpolation and Scaling. The concept is very useful in practice. Think about this: how can we enlarge an image or shrink an image? In the original image, there are fewer number of pixel data. In our textbook, the concept of data interpolation, scaling, and rotation are discussed.

## 6.1 – 6.3 Data Interpolation

There are many situations in which we may want to change the shape, orientation, size of an image. We may want to increase the sampling size of an image so that it may improve the clarity. We may also want to decrease the sampling size without losing the clarity. Here we provide the explanation on how to increase the sampling size for a given 1D and 2D data. Suppose we want to increase the sampling size from 4 pixels to 8 pixels for a given range. Then, we can enlarge the image. We call the operation to fit more pixels into the given range ***interpolation***. For example, **the original sample range for 4 pixels will be enlarged to 8 pixels.** The “guessing” at the function values f(xi’) is called “***interpolation***”. We can use two methods to do the interpolation:

1. Nearest Neighbor Scaling
2. Linear (1D) Scaling or Bilinear (2D) Scaling

### Method 1 – Nearest Neighbor Scaling







In the process of adding more pixels, we need to decide (1) the value of the pixel coordinates, and (2) the gray levels. We ignore the first step because it is intuitively straight forward. We still need to determine the grayscale vaues for each new pixel. We can use the grayscale value of the nearest neighbor as the diagram illustrated. The disadvange is that the image may look “jagged”.

### Method 2 – Linear Scaling

In the following , we illustrate another method that makes the processed image smoother. The grayscale value of the pixel at F can be derived using the slope of the straight line formed by connecting the two pixels at the original image. We need to assume that we know the value of below. We can find the grayscale value of F as follows:

[F – f(x1)] / = [f(x2) – f(x1)] / 1

(Here we also assume that the distance between two pixels is 1.)

After simplifying the equation, we can conclude that

F = [f(x2) – f(x1)] + f(x1)

This is computable if both grayscale values of x1 and x2 are known. This method is known as a linear interpolation in a 1D image. For 2D images, we call the interpolation method Bilinear.





Applying Linear Scaling to an image, we call it Bilinear Interpolation.

We ignore the details below.



In this digram, the open circles are the original image pixels, while the solid dots are the interpolated pixels. This diagram illustrates the concept of Bilinear method in a 2D image. In the following image, the above diagram is annotated with coordinates and scales, e.g.,



If you are interested in the derivation, please refer to the textbook.

Here are the resulting images applying these two methods, respectively.

A picture containing text, person

Description automatically generated

Original Image



As you can see that the Nearest Neighbor Scaling method generates a somewhat “jagged” edges, whereas Bilinear Interpolation generates a smoother image.

6.3 General Data Scaling (optional) - Skipped

This method is a generalization of the interpolation methods we mentioned previously, using the interpolated pixel as the origin. We may ignore the details.





6.4 Scaling of Images

These are examples taken from our textbook. The first image is the original image. You can easily conclude that the Nearest Neighbor generates an image with jagged edges comparing with Bilinear.

3

A picture containing computer

Description automatically generatedAn old photo of a person

Description automatically generatedoriginal image


Bilinear

Nearest Neighbor

Original Image