Abdoulaye Karim
422005b
422005b@upm.edu.sg

# part 1:

(a-) Image resolution:

Image size : 2048 X 2048 px

i) find resolution (lp/mm) to fit 5X5 cm

Note: 1 cm = 10mm

∴ pixel per MM = $\frac{\text{number of pixel}}{\text{size (mm)}} = \frac{2048}{50} = \frac{1024}{25} = 40.96$ MM //

∴ the pixel needed to fit the image into 5X5 cm space = 40.96 MM.

ii) to calculate resolution in dpi for the image to Fit 2X2 inch space.

Note : 1inch = 25.4mm

∴ pixel per inch (dpi) = $\frac{\text{Number of pixel}}{\text{size (inch)}} = \frac{2048}{2} = 1024$ dpi (inch) //

∴ the dpi for the image to fit 2X2 inch = 1024 dpi (inch) //

(B) Linear Indexing in 3D Arrays:

$S = x + M(y + Nz)$

i- $S = x + M(y + Nz)$

$= x + my + Mnz$ //

where x, y, z = coordinate of 3D array element.

MNP = dimension of 3D array along

$S = x + my + Mnz$

x: offset within the current row

my: offset base on the number of rows

Mnz: the offset of how deep the element is in (z)

ii- general form N-dimension

$$S = X_1 + M_1(X_2 + M_2(X_3 + M_3(X_n + M_n)))$$

where:

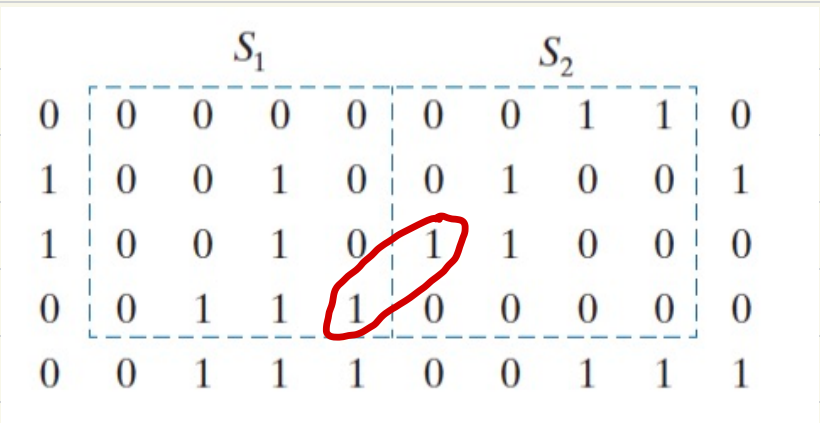
$X_1 X_2 \dots X_n$ = indices along each dimension.

$M_1 M_2 \dots M_n$ = size of the dimension.

$$\therefore S = X_1 + M_1(X_2 + M_2(X_3 + \cdots + M_{n-1} X_n))$$

∴ This expression provides a framework for calculating the linear index in any N-dimensional array based on the contributions of each dimension.

(c)




V=1

4 = None

3 = 1

N = 1

# Part 2:

```
def bilinear_interpolation(image, x, y):  * take image of 2D array
    * to get the dimension of the image
    width = len(image[0])   * No. of rows
    hight = len(image)      * No. of colums

    * the coordinate of the surrounding pixel.
    x1 = floor(x)   * using floor & ceil function to get the
    x2 = ceil(x)      nearest intiger coordinates.
    y1 = floor(y)
    y2 = ceil(y)

    * bounding condition checking
    if x1 < 0 OR x2 >= width OR y1 < 0 OR
    y2 >= hieght :
        Return "out of bound"

    * getting the value of the surrounding pixel
    v11 = image[y1][x1]   * top left
    v21 = image[y1][x2]   * top right
    v12 = image[y2][x1]   * bottom left
    v22 = image[y2][x2]   * bottom right
```

* calculating the weights & distance from the surrounding pixel to $(x, y)$

$dx = x - x_1$

$dy = y - x_1$

* The bilinear interpolation formula & * Interpolated Value of $p(x, y)$

$$p = (V_{11} * (1 - dx) * (1 - dy)) + (V_{21} * dx * (1 - dy)) + (V_{12} * (1 - dx) * dy) + (V_{22} * dx * dy)$$

Return p * This is the interpolated value.

The python code based on my pseudocode and Sample image

Curren

read_pixel_from_image.py | interpolation.ipynb | waziri-Lab04.ipynb

Code | Go To | Managed: http://localhost:8889 | Py

```python
import math
def BilinearInterpolation(image, x, y):
    width = len(image[0])
    height = len(image)
    x1 = math.floor(x)
    x2 = math.ceil(x)
    y1 = math.floor(y)
    y2 = math.ceil(y)
    if x1 < 0 or x2 >= width or y1 < 0 or y2 >= height:
        return "Out of bounds"
    Q11 = image[y1][x1]
    Q21 = image[y1][x2]
    Q12 = image[y2][x1]
    Q22 = image[y2][x2]
    dx = x - x1
    dy = y - y1
    P = (Q11 * (1 - dx) * (1 - dy) +
         Q21 * dx * (1 - dy) +
         Q12 * (1 - dx) * dy +
         Q22 * dx * dy)
    return P
image = [
    [100, 150, 200, 250],
    [120, 170, 220, 270],
    [140, 190, 240, 290],
    [160, 210, 260, 310]
]
x = 1.5
y = 1.5
result = BilinearInterpolation(image, x, y)
print("Interpolated value:", result)
```

✓ [2] < 10 ms

```
Interpolated value: 205.0
```

.ipynb 83:1