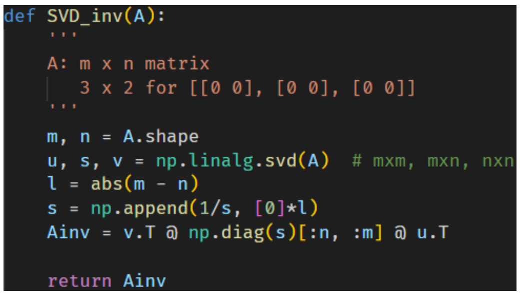
Homework 1

Computer Vision - Photometric Stereo

309553018 林孟學

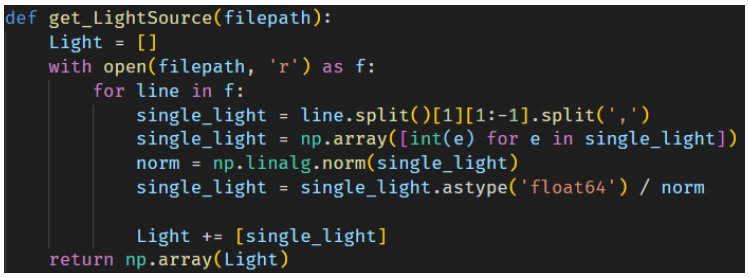
I. My implementation of Photometric Stereo

A. Pseudo inverse using SVD decomposition



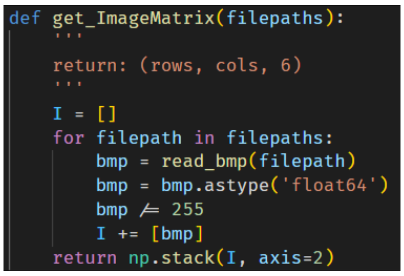
```python
def SVD_inv(A):
    '''
    A: m x n matrix
        3 x 2 for [[0 0], [0 0], [0 0]]
    '''
    m, n = A.shape
    u, s, v = np.linalg.svd(A)  # mxm, mxn, nxn
    l = abs(m - n)
    s = np.append(1/s, [0]*l)
    Ainv = v.T @ np.diag(s)[:n, :m] @ u.T

    return Ainv
```

B. Get Light source matrix (***L***)



```python
def get_LightSource(filepath):
    Light = []
    with open(filepath, 'r') as f:
        for line in f:
            single_light = line.split()[1][1:-1].split(',')
            single_light = np.array([int(e) for e in single_light])
            norm = np.linalg.norm(single_light)
            single_light = single_light.astype('float64') / norm

            Light += [single_light]
    return np.array(Light)
```

C. Get Image matrix (***I***): (rows, cols, **value in pic 1 ~ pic 6**)

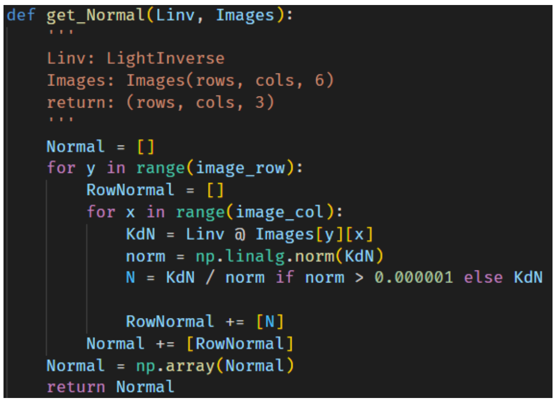


```python
def get_ImageMatrix(filepaths):
    '''
    return: (rows, cols, 6)
    '''
    I = []
    for filepath in filepaths:
        bmp = read_bmp(filepath)
        bmp = bmp.astype('float64')
        bmp /= 255
        I += [bmp]
    return np.stack(I, axis=2)
```

D. Calculate Normal vectors pixel by pixel, norm(Linv * v),

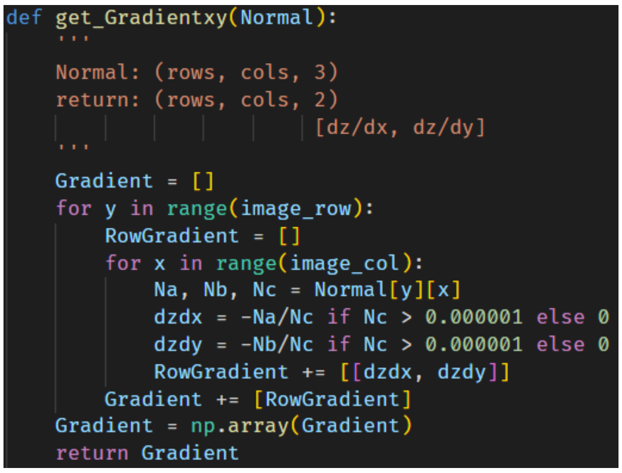
where v=vector of position x, y in **pic 1 ~ pic 6**,

prevent /0 error by 0.000001 threshold(if KdN = 0, ignore empty pixel)



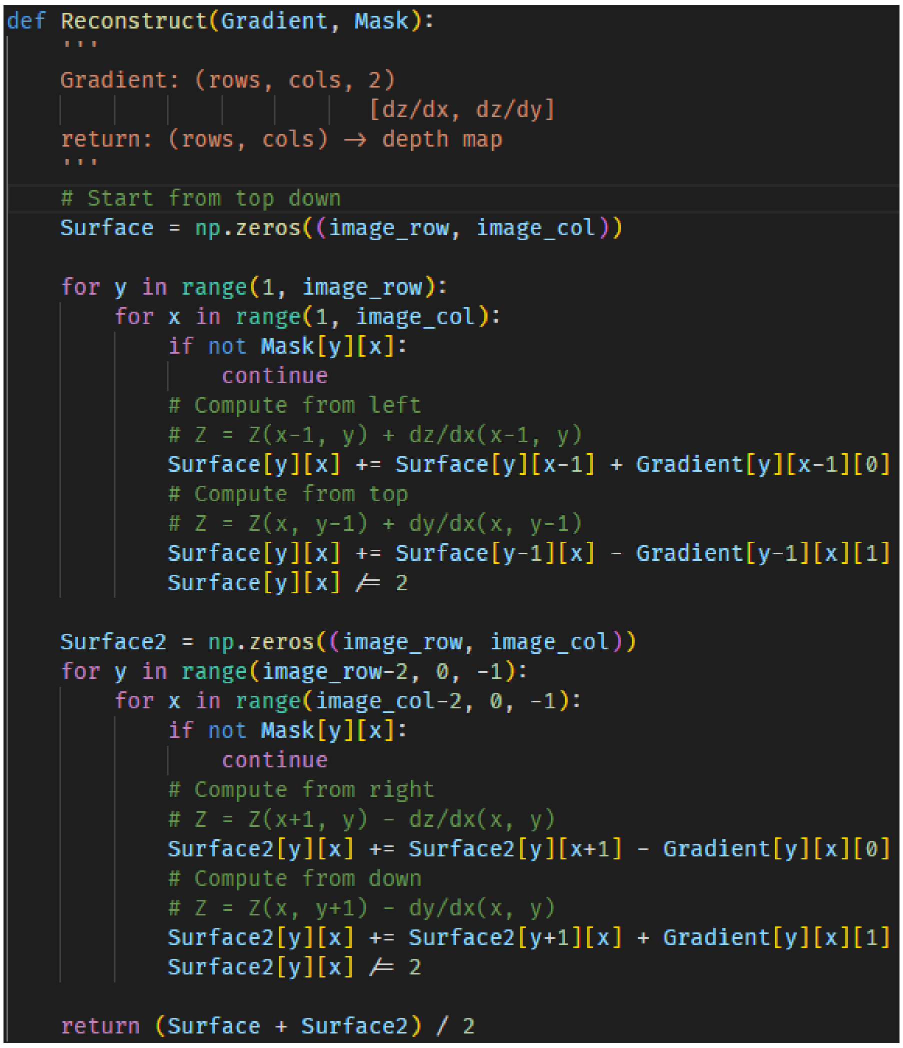
```python
def get_Normal(Linv, Images):
    '''
    Linv: LightInverse
    Images: Images(rows, cols, 6)
    return: (rows, cols, 3)
    '''
    Normal = []
    for y in range(image_row):
        RowNormal = []
        for x in range(image_col):
            KdN = Linv @ Images[y][x]
            norm = np.linalg.norm(KdN)
            N = KdN / norm if norm > 0.000001 else KdN

            RowNormal += [N]
        Normal += [RowNormal]
    Normal = np.array(Normal)
    return Normal
```

E. Calculate Gradient dz/dx and dz/dy from Normal map with equations in slides



```python
def get_Gradientxy(Normal):
    '''
    Normal: (rows, cols, 3)
    return: (rows, cols, 2)
                          [dz/dx, dz/dy]
    '''
    Gradient = []
    for y in range(image_row):
        RowGradient = []
        for x in range(image_col):
            Na, Nb, Nc = Normal[y][x]
            dzdx = -Na/Nc if Nc > 0.000001 else 0
            dzdy = -Nb/Nc if Nc > 0.000001 else 0
            RowGradient += [[dzdx, dzdy]]
        Gradient += [RowGradient]
    Gradient = np.array(Gradient)
    return Gradient
```

F. Use Gradient map to reconstruct depth map from top left, down right.
Here I use average(top + gradient down, left + gradient right) to smooth the surface, otherwise the depth map will be independent in y direction if I only construct with x gradients. // *Gradient at y direction is negative because of the different y direction in textbook and image numpy array.*



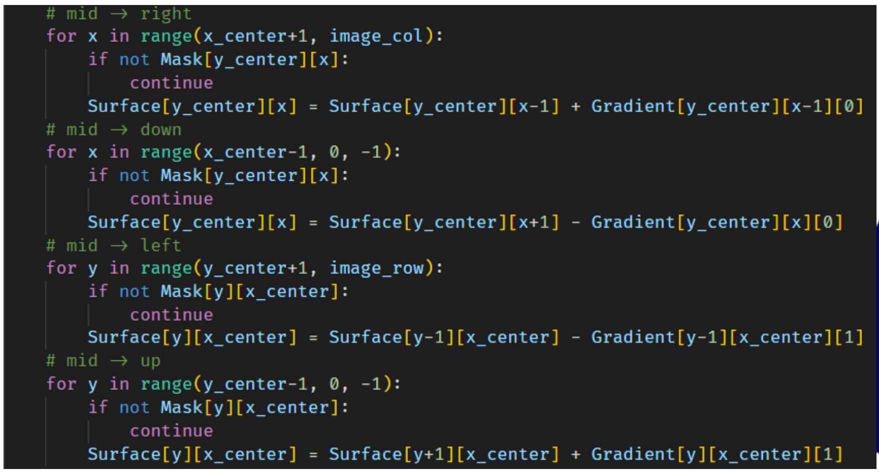
```python
def Reconstruct(Gradient, Mask):
    '''
    Gradient: (rows, cols, 2)
                          [dz/dx, dz/dy]
    return: (rows, cols) -> depth map
    '''
    # Start from top down
    Surface = np.zeros((image_row, image_col))

    for y in range(1, image_row):
        for x in range(1, image_col):
            if not Mask[y][x]:
                continue
            # Compute from left
            # Z = Z(x-1, y) + dz/dx(x-1, y)
            Surface[y][x] += Surface[y][x-1] + Gradient[y][x-1][0]
            # Compute from top
            # Z = Z(x, y-1) + dy/dx(x, y-1)
            Surface[y][x] += Surface[y-1][x] - Gradient[y-1][x][1]
            Surface[y][x] /= 2

    Surface2 = np.zeros((image_row, image_col))
    for y in range(image_row-2, 0, -1):
        for x in range(image_col-2, 0, -1):
            if not Mask[y][x]:
                continue
            # Compute from right
            # Z = Z(x+1, y) - dz/dx(x, y)
            Surface2[y][x] += Surface2[y][x+1] - Gradient[y][x][0]
            # Compute from down
            # Z = Z(x, y+1) - dy/dx(x, y)
            Surface2[y][x] += Surface2[y+1][x] + Gradient[y][x][1]
            Surface2[y][x] /= 2

    return (Surface + Surface2) / 2
```

II. Methods to enhance the result (reconstruction)

A. In addition to I.F reconstruction method, I also tried some different methods.

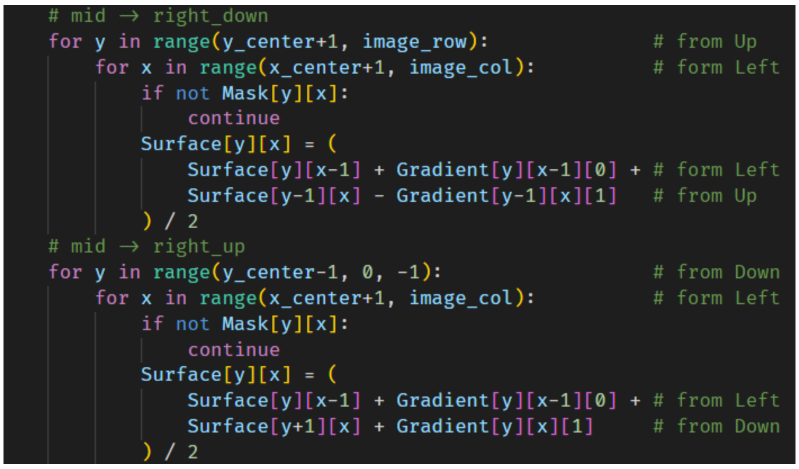
B. Reconstruct from center

1. Reconstruct from center to (x_center, top), (x_center, down), (left, y_center), (right, y_center)



```
# mid → right
for x in range(x_center+1, image_col):
    if not Mask[y_center][x]:
        continue
    Surface[y_center][x] = Surface[y_center][x-1] + Gradient[y_center][x-1][0]
# mid → down
for x in range(x_center-1, 0, -1):
    if not Mask[y_center][x]:
        continue
    Surface[y_center][x] = Surface[y_center][x+1] - Gradient[y_center][x][0]
# mid → left
for y in range(y_center+1, image_row):
    if not Mask[y][x_center]:
        continue
    Surface[y][x_center] = Surface[y-1][x_center] - Gradient[y-1][x_center][1]
# mid → up
for y in range(y_center-1, 0, -1):
    if not Mask[y][x_center]:
        continue
    Surface[y][x_center] = Surface[y+1][x_center] + Gradient[y][x_center][1]
```

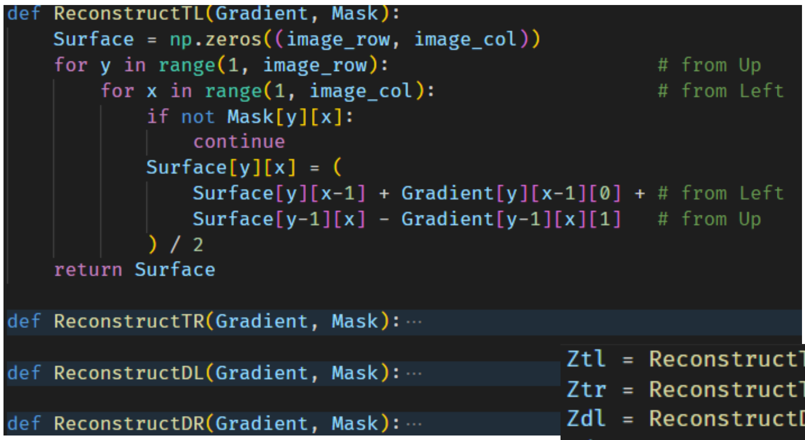
2. Reconstruct remaining pixels by values calculate in previous step



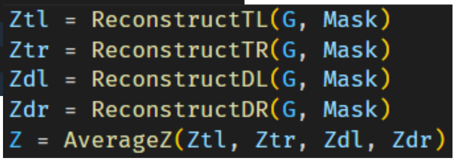
```
# mid → right_down
for y in range(y_center+1, image_row):                # from Up
    for x in range(x_center+1, image_col):            # form Left
        if not Mask[y][x]:
            continue
        Surface[y][x] = (
            Surface[y][x-1] + Gradient[y][x-1][0] +   # form Left
            Surface[y-1][x] - Gradient[y-1][x][1]     # from Up
        ) / 2
# mid → right_up
for y in range(y_center-1, 0, -1):                    # from Down
    for x in range(x_center+1, image_col):            # form Left
        if not Mask[y][x]:
            continue
        Surface[y][x] = (
            Surface[y][x-1] + Gradient[y][x-1][0] +   # from Left
            Surface[y+1][x] + Gradient[y][x][1]       # from Down
        ) / 2
```

…

C. Reconstruct from top left, top right, down left, down right, and average them

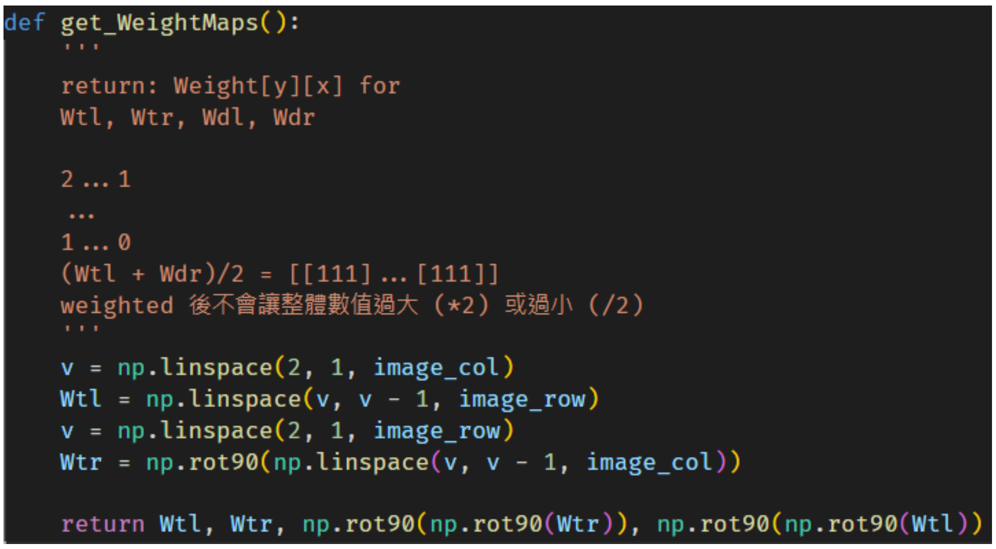


```
def ReconstructTL(Gradient, Mask):
    Surface = np.zeros((image_row, image_col))
    for y in range(1, image_row):                     # from Up
        for x in range(1, image_col):                 # from Left
            if not Mask[y][x]:
                continue
            Surface[y][x] = (
                Surface[y][x-1] + Gradient[y][x-1][0] + # from Left
                Surface[y-1][x] - Gradient[y-1][x][1]   # from Up
            ) / 2
    return Surface

def ReconstructTR(Gradient, Mask):…

def ReconstructDL(Gradient, Mask):…

def ReconstructDR(Gradient, Mask):…
```



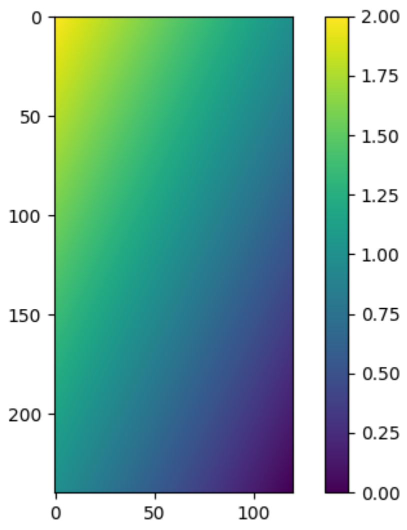
```
Ztl = ReconstructTL(G, Mask)
Ztr = ReconstructTR(G, Mask)
Zdl = ReconstructDL(G, Mask)
Zdr = ReconstructDR(G, Mask)
Z = AverageZ(Ztl, Ztr, Zdl, Zdr)
```

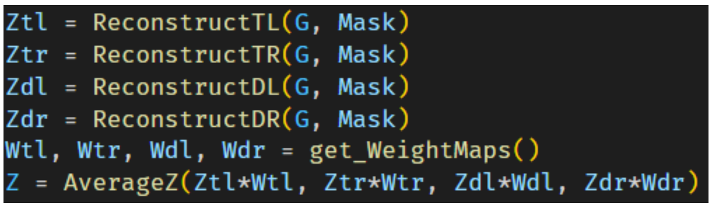
D. Similar to II.C, but use weighted average according to their reconstruct begin points.



```
def get_WeightMaps():
    '''
    return: Weight[y][x] for
    Wtl, Wtr, Wdl, Wdr

    2 ... 1
     ...
    1 ... 0
    (Wtl + Wdr)/2 = [[111] ... [111]]
    weighted 後不會讓整體數值過大 (*2) 或過小 (/2)
    '''
    v = np.linspace(2, 1, image_col)
    Wtl = np.linspace(v, v - 1, image_row)
    v = np.linspace(2, 1, image_row)
    Wtr = np.rot90(np.linspace(v, v - 1, image_col))

    return Wtl, Wtr, np.rot90(np.rot90(Wtr)), np.rot90(np.rot90(Wtl))
```

Like this. Weight map for top left



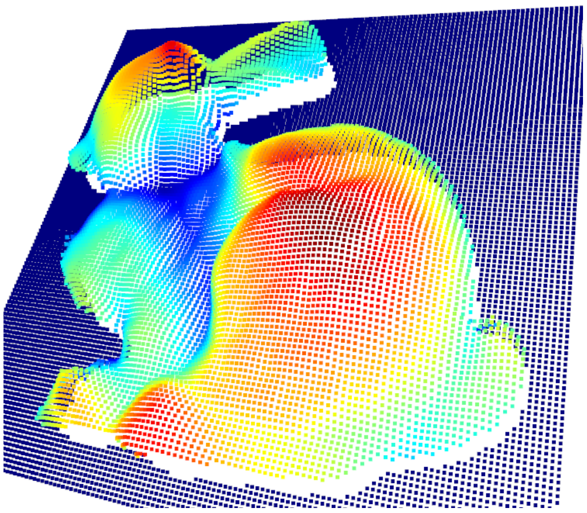
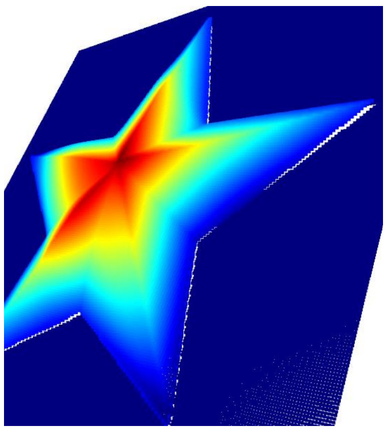


```
Ztl = ReconstructTL(G, Mask)
Ztr = ReconstructTR(G, Mask)
Zdl = ReconstructDL(G, Mask)
Zdr = ReconstructDR(G, Mask)
Wtl, Wtr, Wdl, Wdr = get_WeightMaps()
Z = AverageZ(Ztl*Wtl, Ztr*Wtr, Zdl*Wdl, Zdr*Wdr)
```

III. Compare results in part 2.

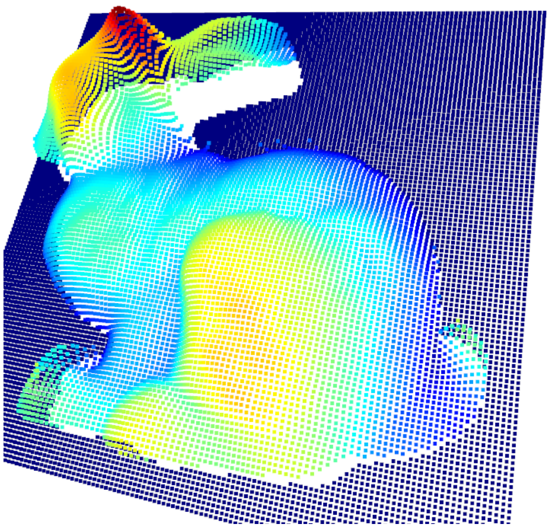
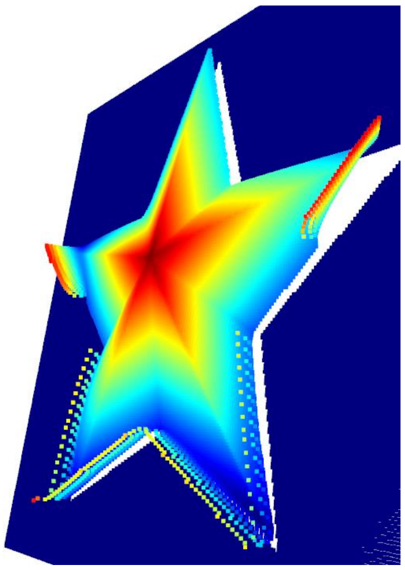
A. Original: left top + right down

看起來還行，平滑平滑

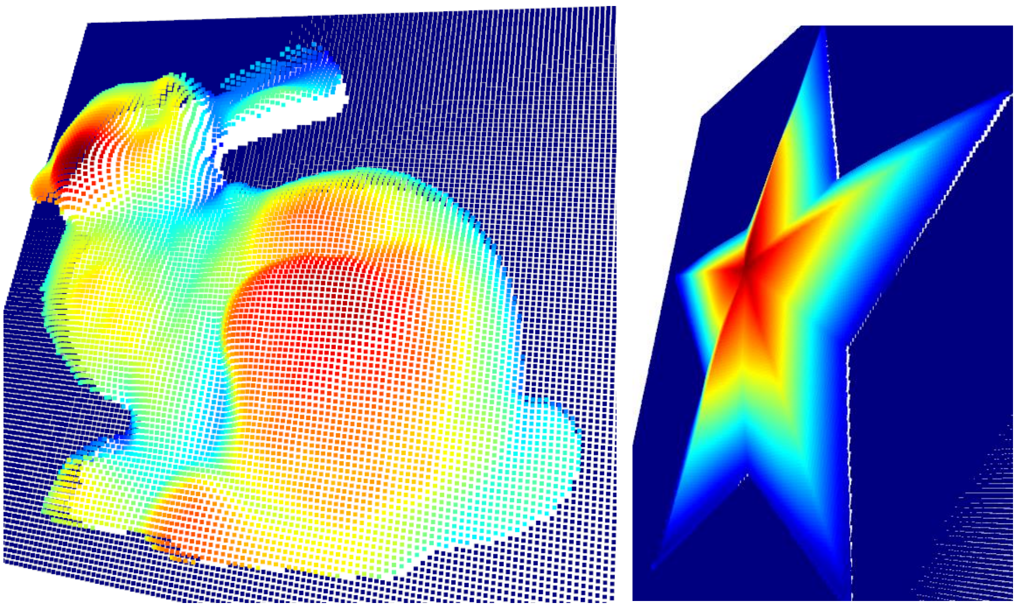



B. From center

在星星邊緣得到非常奇怪的結果，或許是因為太遠離中心了

C. left top + right top + left down + right down



D. Weighted average of III.C

看起來差不多，理論上有 weighted 應該要比較好，不過 weighted 反而星星邊緣比較不服貼