Homework 1

Computer Vision 2022 Spring

310554031 數據所 葉詠富

Github: https://github.com/frankye1000/NYCU-ComputerVision/tree/master/HW1

➢ Preprocess

1. Import package

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import csr_matrix
from scipy.linalg import pinv, norm
from sklearn.preprocessing import normalize

from HW1 import *
```

2. Set image source(‘bunny’、’star’)

圖片來源種類

```
source='bunny' # 'star'
```

➢ Normal Estimation

3. Read LightSource.txt

   記得光源要做 normalization!(圖 1、圖 2)

read LightSource.txt

讀取光源

```
with open("test/{}/LightSource.txt".format(source),'r') as f:
    L = []
    for l in f.readlines():
        l = l.replace('\n','').replace('(','').replace(')','').split(' ')[1].split(',')
        l = [int(i) for i in l]
        L.append(l)

# to array
L = np.array(L)
# normalize
L = normalize(L)
```

4. Read images

   Use numpy.vstack to stack the image vectors

read images

讀取影像

```
img1 = read_bmp('test/{}/pic1.bmp'.format(source))
h = img1.shape[0]
w = img1.shape[1]
```

```
img1 = read_bmp('test/{}/pic1.bmp'.format(source)).reshape(-1)
img2 = read_bmp('test/{}/pic2.bmp'.format(source)).reshape(-1)
img3 = read_bmp('test/{}/pic3.bmp'.format(source)).reshape(-1)
img4 = read_bmp('test/{}/pic4.bmp'.format(source)).reshape(-1)
img5 = read_bmp('test/{}/pic5.bmp'.format(source)).reshape(-1)
img6 = read_bmp('test/{}/pic6.bmp'.format(source)).reshape(-1)
```

```
I = np.vstack((img1,img2,img3,img4,img5,img6))
I.shape
```

```
(6, 14400)
```

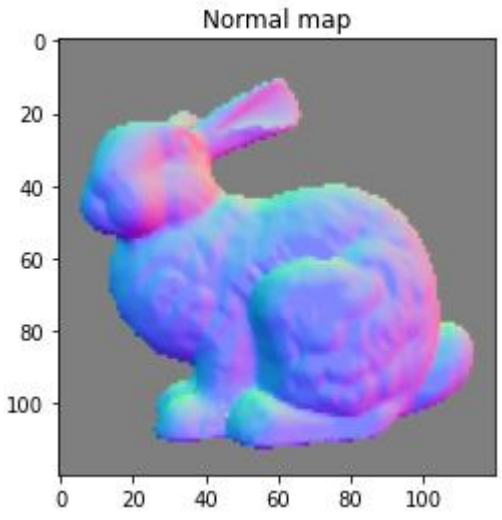
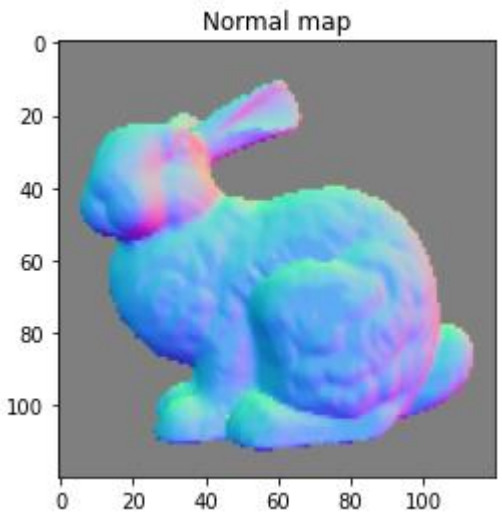
5. Use pseudo inverse calculate KdN

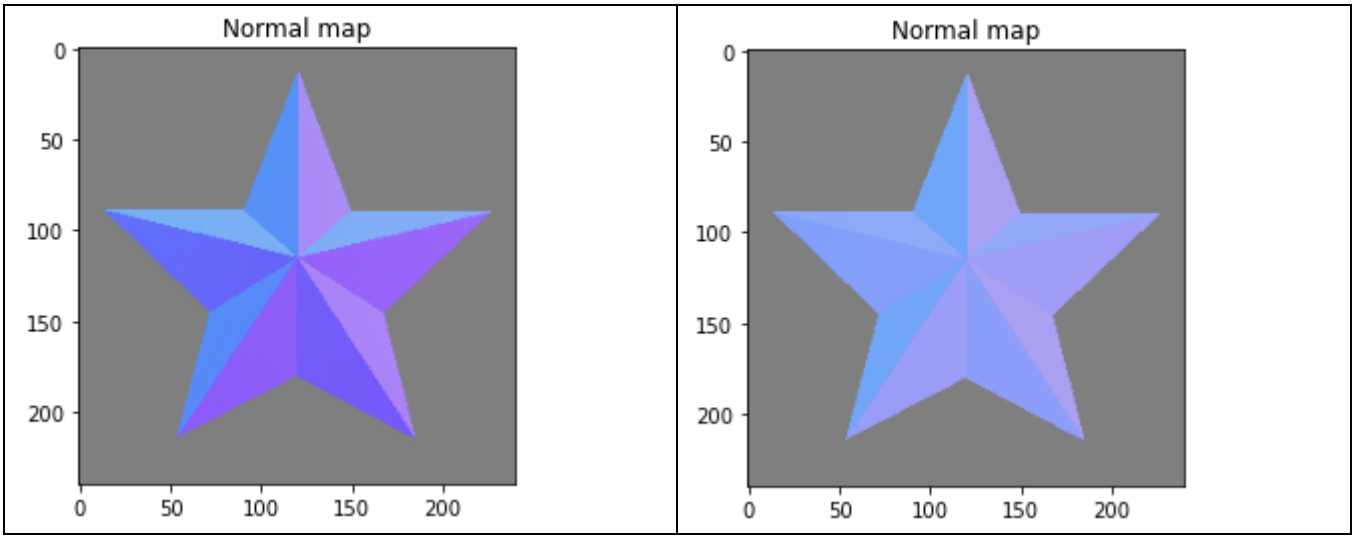
KdN

利用pesudo-inverse計算KdN

```
KdN = pinv((L.T@L))@L.T@I
KdN = KdN.T          # 需要再轉置一次，畫出來結果才是對的
N = normalize(KdN) # 正規化
```

6. Visualize Normal map

圖 1、LightSource with normalization	圖 2、LightSource without normalization
	



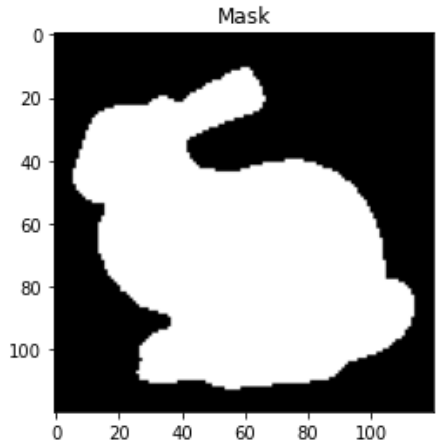
7. Set Mask

Mask

製作mask判斷使用哪個方向向量

```
N_copy = N.reshape((w,w,3))

Mask = np.zeros((w,h))
for x in range(w):
    for y in range(h):
        if (N_copy[x,y] == np.array([0,0,0])).all():
            Mask[x,y] = 0
        else:
            Mask[x,y] = 1
```

```
mask_visualization(Mask)
```



➢ Surface Reconstruction

8. Set M、V, and set index dictionary to record 2D->1D index

```
s = w * h
M = np.zeros((2 * s, s))
V = np.zeros((2 * s, 1))

# 製作index，紀錄2維時的index位置
index = {}
c = 0
for i in range(w):
    for j in range(h):
        index[j,i] = c
        c+=1
```

9. Use Mask to determine vector direction and fill in M & V

   To fill in the M and V matrices according to the following matrix orientation diagram (圖 3) and Mask, the rules are as follows:

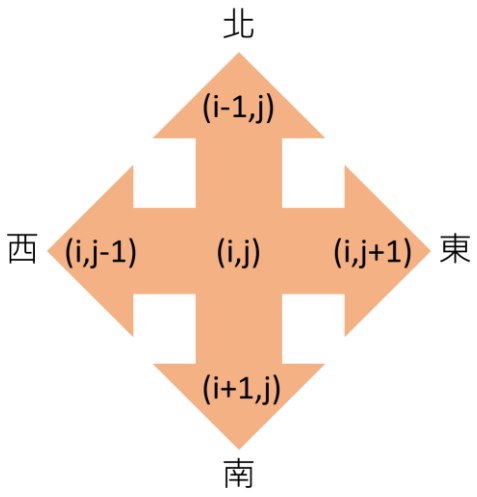
   A. 如果東北有向量，就用東北向量

   B. 如果西北有向量(東沒有向量)，就用西北向量

   C. 如果東南有向量(北沒有向量)，就用東南向量

   D. 如果西南有向量(東北都沒有向量)，就用西南向量

   E. 如果東南西北都沒有向量，就把 z 設成 0



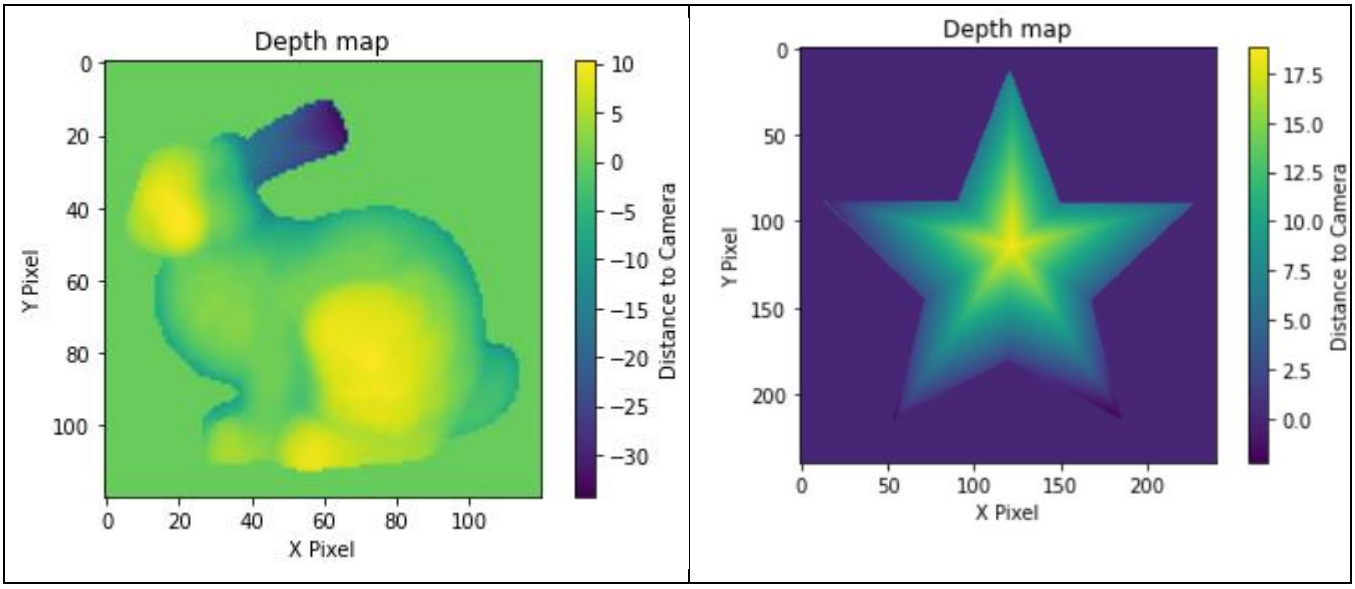
(圖 3)

```python
# 用Mask來判斷向量方向並製作 M & V
i_row_pointer = 0
j_row_pointer = s


for j in range(w):
    for i in range(h):
        nx = N_copy[i][j][0]
        ny = N_copy[i][j][1]
        nz = N_copy[i][j][2]
        '''超過圖片範圍就丟棄'''
        if i-1>=0 and j-1>=0 and i+1<120 and j+1<120:
            '''填表'''
            if Mask[i,j+1]>0 and Mask[i-1,j]>0:           # 如果東北方向存在
                M[i_row_pointer][index[(i,j+1)]] = 1        # (i,j+1)
                M[j_row_pointer][index[(i-1,j)]] = 1        # (i-1,j)
                M[i_row_pointer][index[(i,j)]]   = -1       # (i,j)
                M[j_row_pointer][index[(i,j)]]   = -1       # (i,j)
                V[i_row_pointer] = -nx/nz if nz!=0 else 0
                V[j_row_pointer] = -ny/nz if nz!=0 else 0
            elif Mask[i,j-1]>0 and Mask[i-1,j]>0:         # 如果西北方向存在(東不存在)
                M[i_row_pointer][index[(i,j-1)]] = 1        # (i,j-1)
                M[j_row_pointer][index[(i-1,j)]] = 1        # (i-1,j)
                M[i_row_pointer][index[(i,j)]]   = -1       # (i,j)
                M[j_row_pointer][index[(i,j)]]   = -1       # (i,j)
                V[i_row_pointer] =  nx/nz if nz!=0 else 0
                V[j_row_pointer] = -ny/nz if nz!=0 else 0
            elif Mask[i,j+1]>0 and Mask[i+1,j]>0:         # 如果東南方向存在(北不存在)
                M[i_row_pointer][index[(i,j+1)]] = 1        # (i,j+1)
                M[j_row_pointer][index[(i+1,j)]] = 1        # (i+1,j)
                M[i_row_pointer][index[(i,j)]]   = -1       # (i,j)
                M[j_row_pointer][index[(i,j)]]   = -1       # (i,j)
                V[i_row_pointer] = -nx/nz if nz!=0 else 0
                V[j_row_pointer] =  ny/nz if nz!=0 else 0
            elif Mask[i,j-1]>0 and Mask[i+1,j]>0:        # 如果西南方向存在(東、北不存在)
                M[i_row_pointer][index[(i,j-1)]] = 1        # (i,j-1)
                M[j_row_pointer][index[(i+1,j)]] = 1        # (i+1,j)
                M[i_row_pointer][index[(i,j)]]   = -1       # (i,j)
                M[j_row_pointer][index[(i,j)]]   = -1       # (i,j)
                V[i_row_pointer] = nx/nz if nz!=0 else 0
                V[j_row_pointer] = ny/nz if nz!=0 else 0
        i_row_pointer += 1
        j_row_pointer += 1
```

10. Use pseudo inverse calculate z, and reshape z

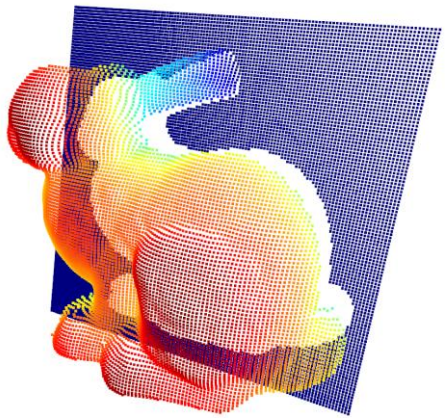
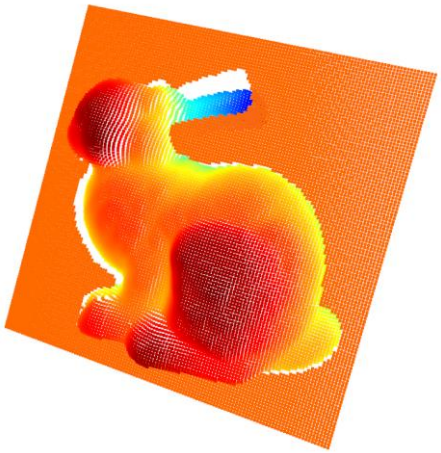
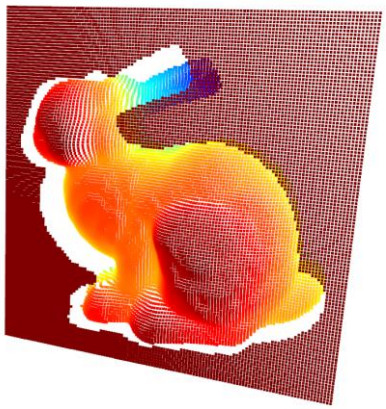
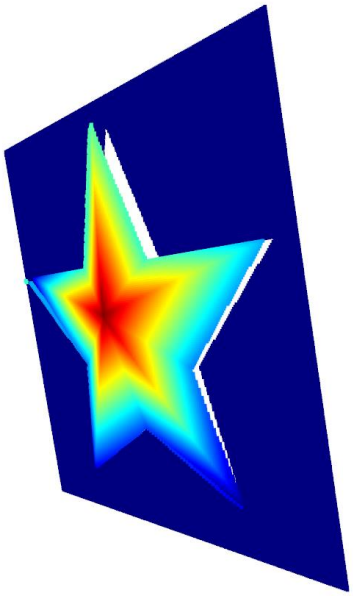
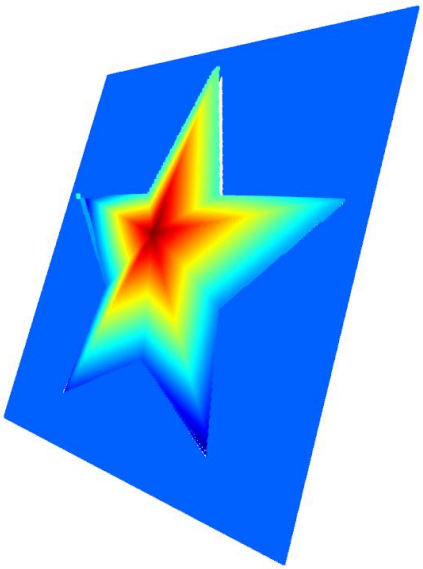
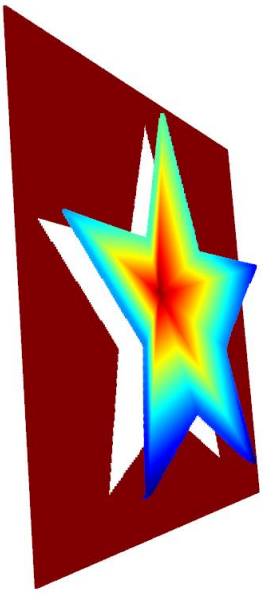
```python
# 算出z
z = pinv(M.T@M)@M.T@V

# reshape z
z_r = np.reshape(z,(w,h),order='F')
```

11. Visualize Depth map



12. Save 3D ply
13. Show 3D ply

Set different baseline value

baseline_val = min(Z_map)	baseline_val = mean(Z_map)	baseline_val = max(Z_map)
		
		

➢ Discussion and improve

1. When reconstructing the surface, you can use Mask to only calculate patterns such as rabbits and stars, and reduce sparse matrix operations, thereby reducing time and memory usage.
   For example, the star was originally calculated to be s=240*240=57600, but if only the N part is counted, s=16626

```
# 只需計算!!!!!!!!!
len(index)
```

```
]: 16626
```

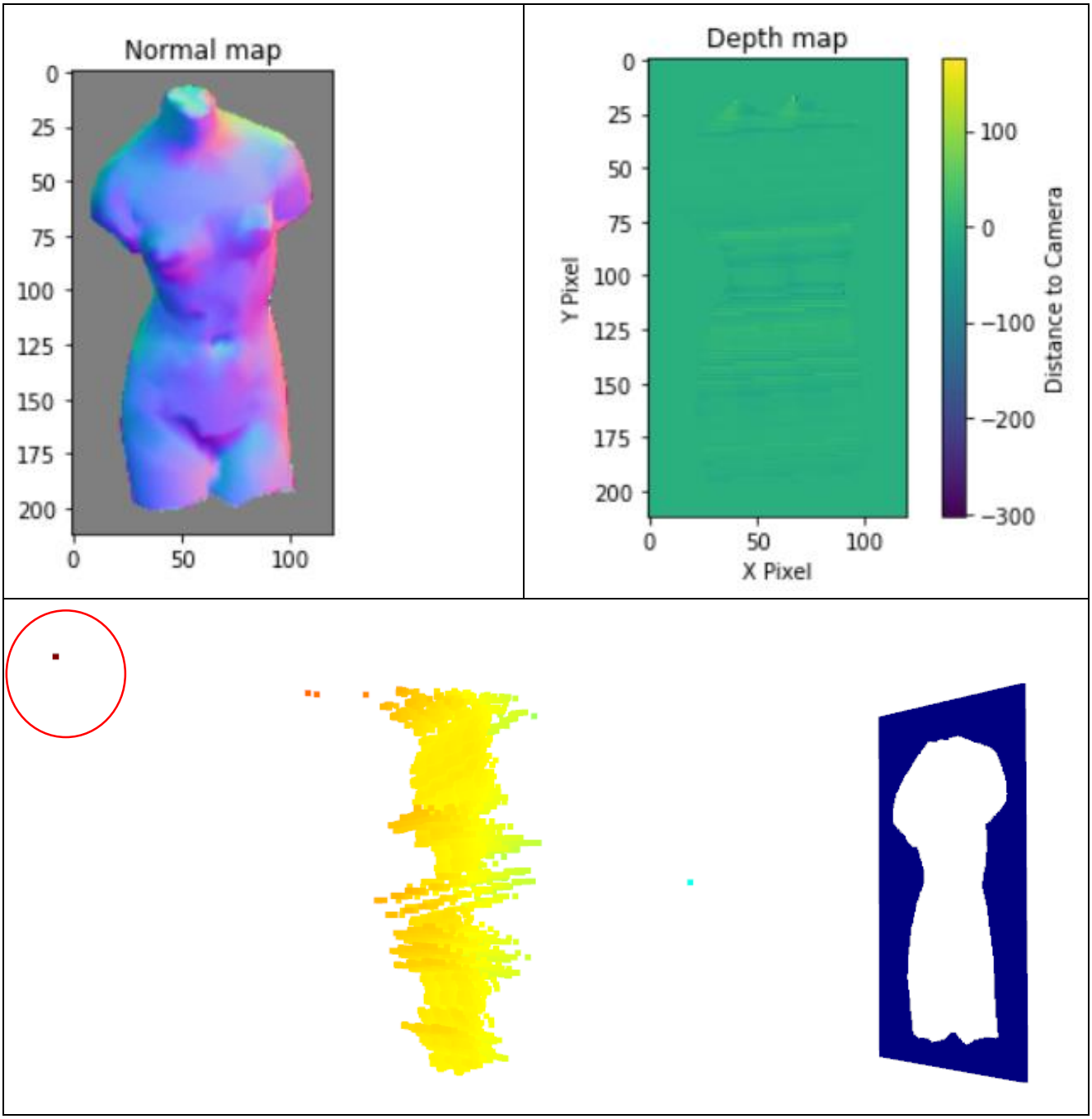
Surface Reconstruction(新版)

```
s = len(calculate_pixels)
M = np.zeros((2 * s, s))
V = np.zeros((2 * s, 1))
```

```
# 用Mask來判斷向量方向並製作 M & V
i_row_pointer = 0
j_row_pointer = s
for key in index:
    i = key[0]
    j = key[1]

    nx = N_copy[i][j][0]
    ny = N_copy[i][j][1]
    nz = N_copy[i][j][2]
    '''超過圖片範圍就丟棄'''
    if i-1>=0 and j-1>=0 and i+1<w and j+1<h:
        '''填表'''
        if Mask[i,j+1]>0 and Mask[i-1,j]>0:           # 如果東北方向存在
            M[i_row_pointer][index[(i,j+1)]] = 1        # (i,j+1)
            M[j_row_pointer][index[(i-1,j)]] = 1        # (i-1,j)
            M[i_row_pointer][index[(i,j)]]   = -1       # (i,j)
            M[j_row_pointer][index[(i,j)]]   = -1       # (i,j)
            V[i_row_pointer] = -nx/nz if nz!=0 else 0
            V[j_row_pointer] = -ny/nz if nz!=0 else 0
        elif Mask[i,j-1]>0 and Mask[i-1,j]>0:         # 如果西北方向存在(東不存在)
            M[i_row_pointer][index[(i,j-1)]] = 1        # (i,j-1)
            M[j_row_pointer][index[(i-1,j)]] = 1        # (i-1,j)
            M[i_row_pointer][index[(i,j)]]   = -1       # (i,j)
            M[j_row_pointer][index[(i,j)]]   = -1       # (i,j)
            V[i_row_pointer] =  nx/nz if nz!=0 else 0
            V[j_row_pointer] = -ny/nz if nz!=0 else 0
        elif Mask[i,j+1]>0 and Mask[i+1,j]>0:         # 如果東南方向存在(北不存在)
            M[i_row_pointer][index[(i,j+1)]] = 1        # (i,j+1)
            M[j_row_pointer][index[(i+1,j)]] = 1        # (i+1,j)
            M[i_row_pointer][index[(i,j)]]   = -1       # (i,j)
            M[j_row_pointer][index[(i,j)]]   = -1       # (i,j)
            V[i_row_pointer] = -nx/nz if nz!=0 else 0
            V[j_row_pointer] =  ny/nz if nz!=0 else 0
        elif Mask[i,j-1]>0 and Mask[i+1,j]>0:        # 如果西南方向存在(東、北不存在)
            M[i_row_pointer][index[(i,j-1)]] = 1        # (i,j-1)
            M[j_row_pointer][index[(i+1,j)]] = 1        # (i+1,j)
            M[i_row_pointer][index[(i,j)]]   = -1       # (i,j)
            M[j_row_pointer][index[(i,j)]]   = -1       # (i,j)
            V[i_row_pointer] = nx/nz if nz!=0 else 0
            V[j_row_pointer] = ny/nz if nz!=0 else 0
    i_row_pointer += 1
    j_row_pointer += 1
```

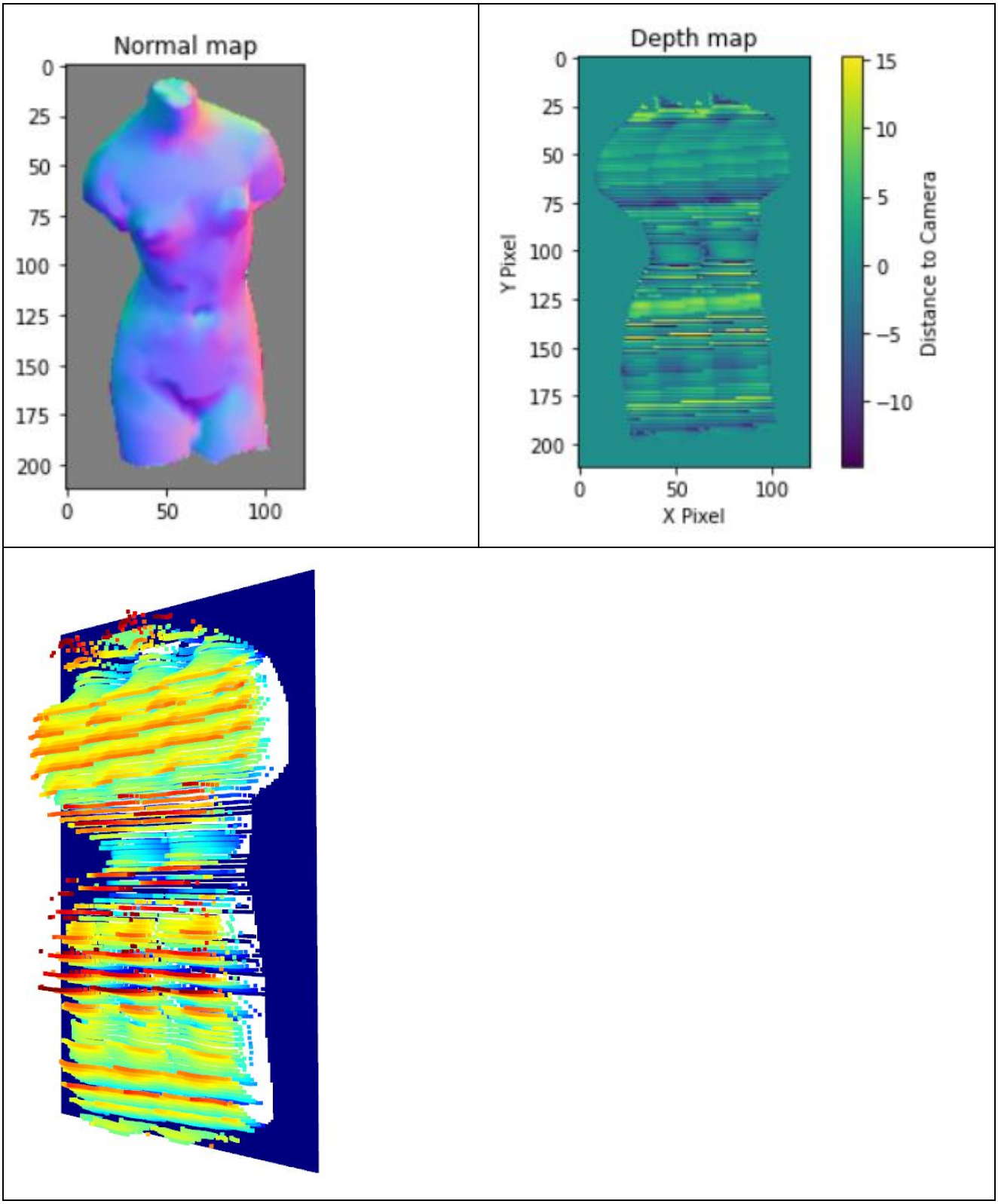
2. Bonus- venus

   If you don't deal with anything, you will find that **outliers** appear when you directly count the venus images.

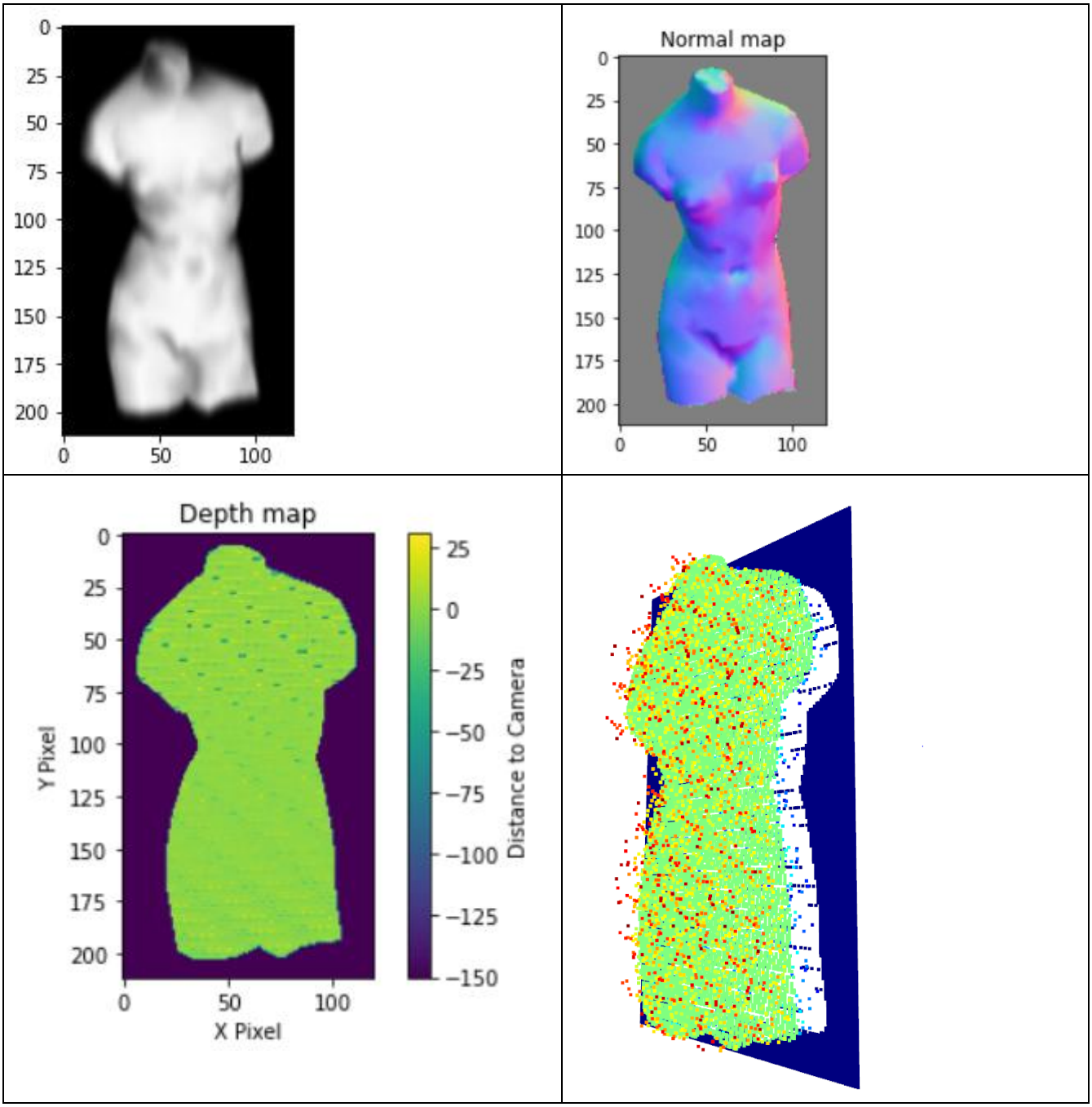


So I use statistical formula to calculate IQR, then find the upper and lower bounds, and set the z of the outlier outside the boundary to 0. The 3D image presented will not have the outlier phenomenon.

IQR = Q3 - Q1
Lowerbound = Q1 - 1.5*IQR
Upperbound = Q1 - 1.5*IQR



另一個嘗試先對圖片進行高斯模糊，降低噪點，再進行運算。



➢ Conclusion

In the previous attempt, bunny is the most suitable for 3D images, and venus is the least suitable. The initial observation should be that the position of the light source affects 3D imaging.