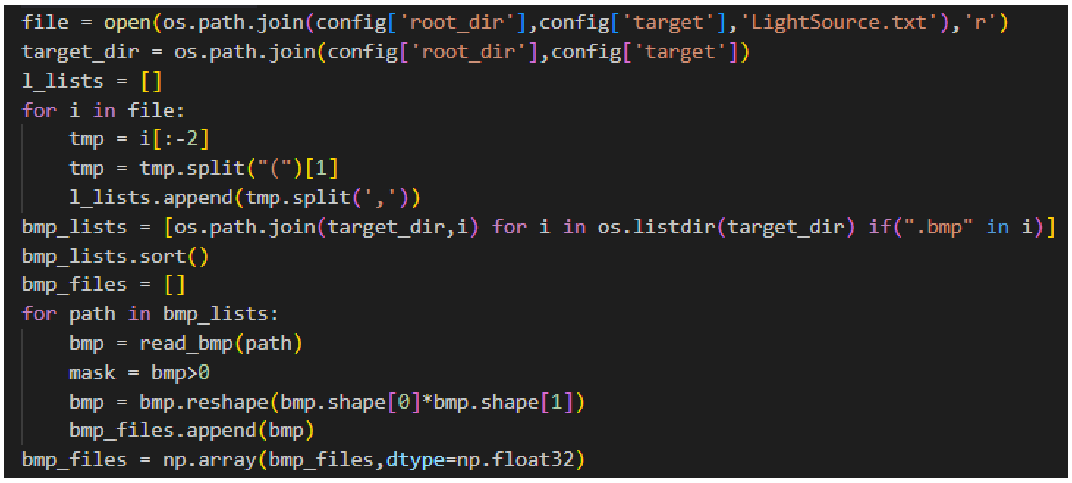
Computer Vision HW1

1. Simply explain your implementation and what kind of method you use to enhance the result and compare the result

I decided to use method 2 to reconstruct the depth of the object. My implementation consists of three steps:

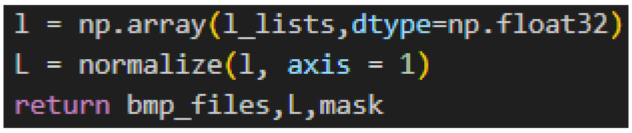
(a). Read the data

At this stage, I read the data and fill it into the corresponding matrix.



```python
file = open(os.path.join(config['root_dir'],config['target'],'LightSource.txt'),'r')
target_dir = os.path.join(config['root_dir'],config['target'])
l_lists = []
for i in file:
    tmp = i[:-2]
    tmp = tmp.split("(")[1]
    l_lists.append(tmp.split(','))
bmp_lists = [os.path.join(target_dir,i) for i in os.listdir(target_dir) if(".bmp" in i)]
bmp_lists.sort()
bmp_files = []
for path in bmp_lists:
    bmp = read_bmp(path)
    mask = bmp>0
    bmp = bmp.reshape(bmp.shape[0]*bmp.shape[1])
    bmp_files.append(bmp)
bmp_files = np.array(bmp_files,dtype=np.float32)
```

Next, I normalize the light source matrix and return the processed matrix,



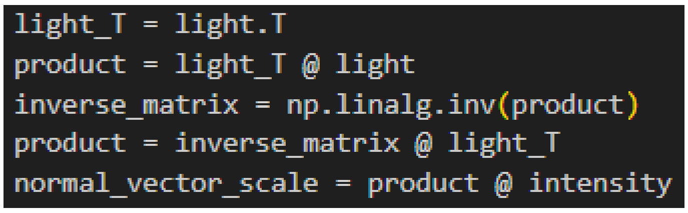
```python
l = np.array(l_lists,dtype=np.float32)
L = normalize(l, axis = 1)
return bmp_files,L,mask
```

(b). Calculate normal vector

In this part, I want the normal vector of an object in the image. I follow the instruction of spec to calculate the normal vector.

$$I = LK_dN \Longrightarrow L^TI = L^TLK_dN \Longrightarrow K_dN = (L^TL)^{-1}L^TI$$

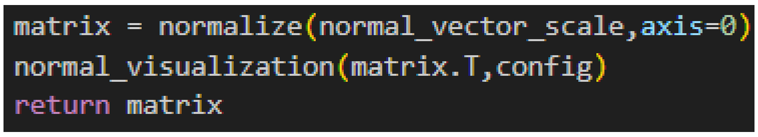
I use "light" to represent "L" and use "normal_vector_scale" to represent "K_dN".



```python
light_T = light.T
product = light_T @ light
inverse_matrix = np.linalg.inv(product)
product = inverse_matrix @ light_T
normal_vector_scale = product @ intensity
```

Finally, I get the final matrix (3x(height x width)). I normalize this vector through the first axis and get the final result.

$$N = \frac{K_dN}{||K_dN||}$$



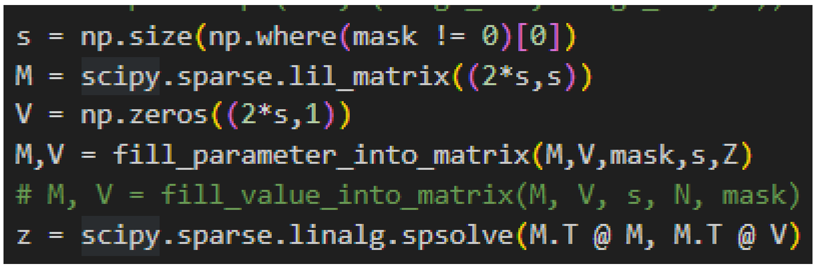
```
matrix = normalize(normal_vector_scale,axis=0)
normal_visualization(matrix.T,config)
return matrix
```

(c). Reconstruct the depth of object via normal vector

At this part, I follow the instruction of spec to infer relative depth from the normal vector.

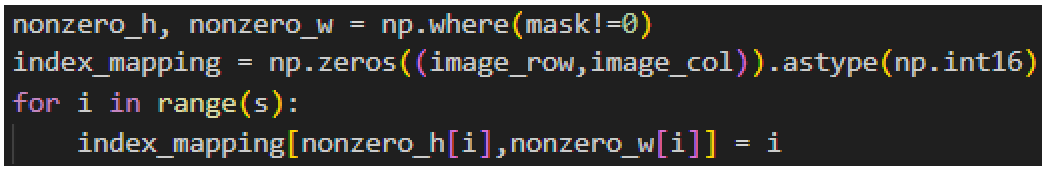
$$z = (M^T M)^{-1} M^T V$$

However, I need to improve the algorithm due to the complexity of calculation. Given an image with shape 100x100, the shape of M would be (20000x10000). Thus, the operation of $M^T M$ and $(M^T M)^{-1}$ would be consuming. Since M is a sparse matrix, I could use the package ("scipy") to speed up these operations.



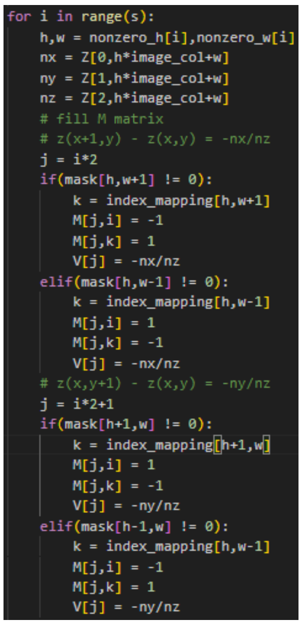
```
s = np.size(np.where(mask != 0)[0])
M = scipy.sparse.lil_matrix((2*s,s))
V = np.zeros((2*s,1))
M,V = fill_parameter_into_matrix(M,V,mask,s,Z)
# M, V = fill_value_into_matrix(M, V, s, N, mask)
z = scipy.sparse.linalg.spsolve(M.T @ M, M.T @ V)
```

Besides, we also utilize the technique of masks to help to decrease the complexity and improve the quality of the result. First, we construct the mapping function to record the relation between the location of calculation and the location of the image.



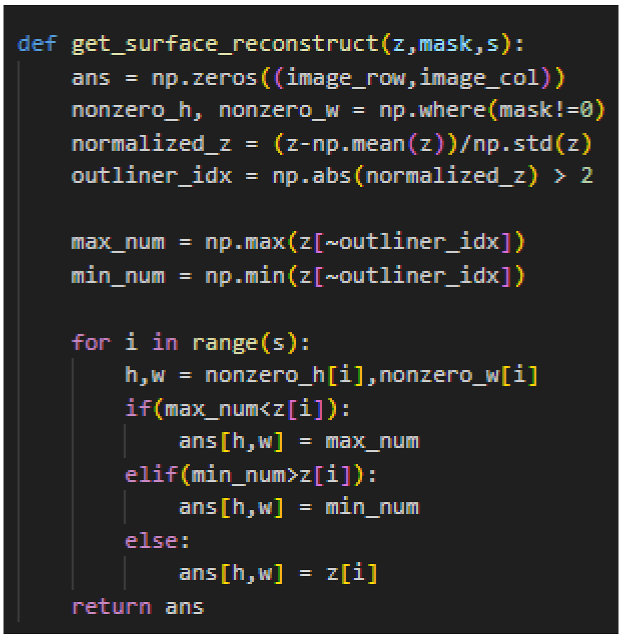
```
nonzero_h, nonzero_w = np.where(mask!=0)
index_mapping = np.zeros((image_row,image_col)).astype(np.int16)
for i in range(s):
    index_mapping[nonzero_h[i],nonzero_w[i]] = i
```

We iterate the array and fill the value into the corresponding location in the matrix as the instruction of spec.



```
for i in range(s):
    h,w = nonzero_h[i],nonzero_w[i]
    nx = Z[0,h*image_col+w]
    ny = Z[1,h*image_col+w]
    nz = Z[2,h*image_col+w]
    # fill M matrix
    # z(x+1,y) - z(x,y) = -nx/nz
    j = i*2
    if(mask[h,w+1] != 0):
        k = index_mapping[h,w+1]
        M[j,i] = -1
        M[j,k] = 1
        V[j] = -nx/nz
    elif(mask[h,w-1] != 0):
        k = index_mapping[h,w-1]
        M[j,i] = 1
        M[j,k] = -1
        V[j] = -nx/nz
    # z(x,y+1) - z(x,y) = -ny/nz
    j = i*2+1
    if(mask[h+1,w] != 0):
        k = index_mapping[h+1,w]
        M[j,i] = 1
        M[j,k] = -1
        V[j] = -ny/nz
    elif(mask[h-1,w] != 0):
        k = index_mapping[h,w-1]
        M[j,i] = -1
        M[j,k] = 1
        V[j] = -ny/nz
```

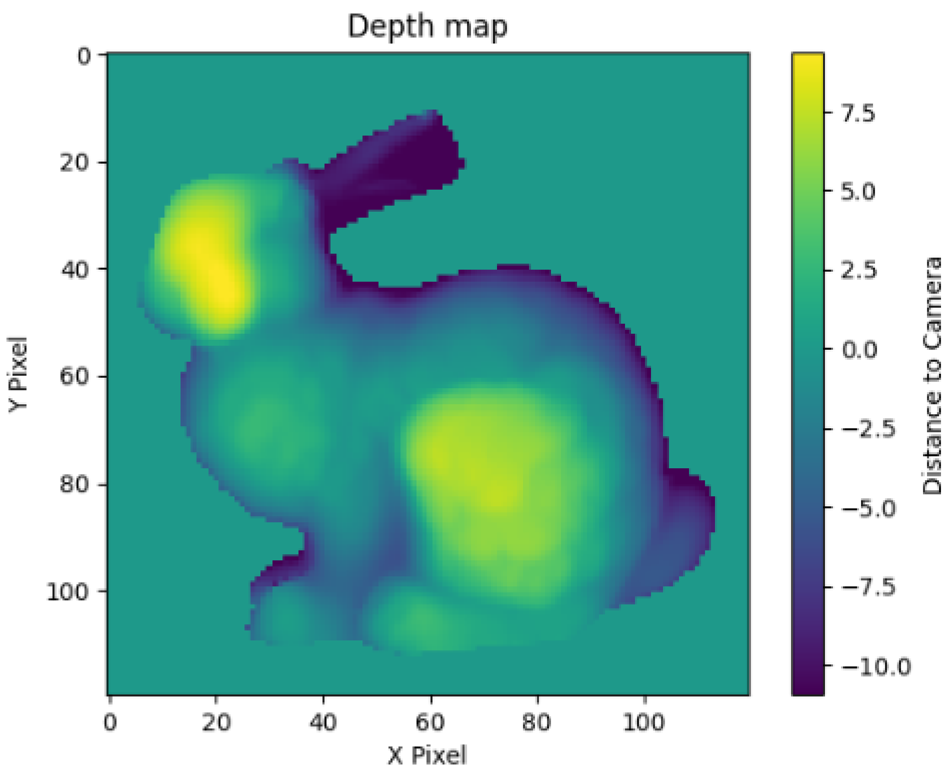
Finally, we reconstruct the result to the depth. We consider depths exceeding two standard deviations from the mean of the entire image as outliers, and replace them with mean ± std.

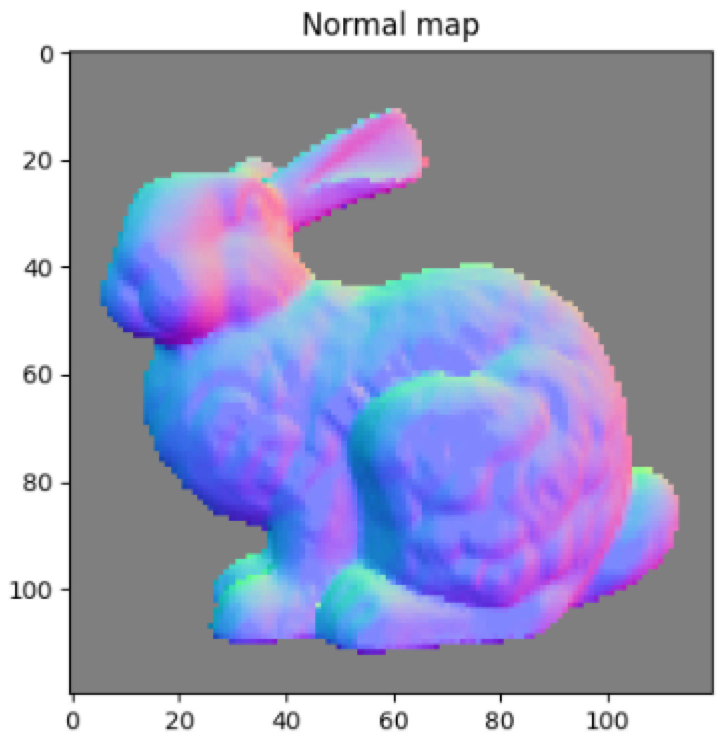


```python
def get_surface_reconstruct(z,mask,s):
    ans = np.zeros((image_row,image_col))
    nonzero_h, nonzero_w = np.where(mask!=0)
    normalized_z = (z-np.mean(z))/np.std(z)
    outliner_idx = np.abs(normalized_z) > 2

    max_num = np.max(z[~outliner_idx])
    min_num = np.min(z[~outliner_idx])

    for i in range(s):
        h,w = nonzero_h[i],nonzero_w[i]
        if(max_num<z[i]):
            ans[h,w] = max_num
        elif(min_num>z[i]):
            ans[h,w] = min_num
        else:
            ans[h,w] = z[i]
    return ans
```

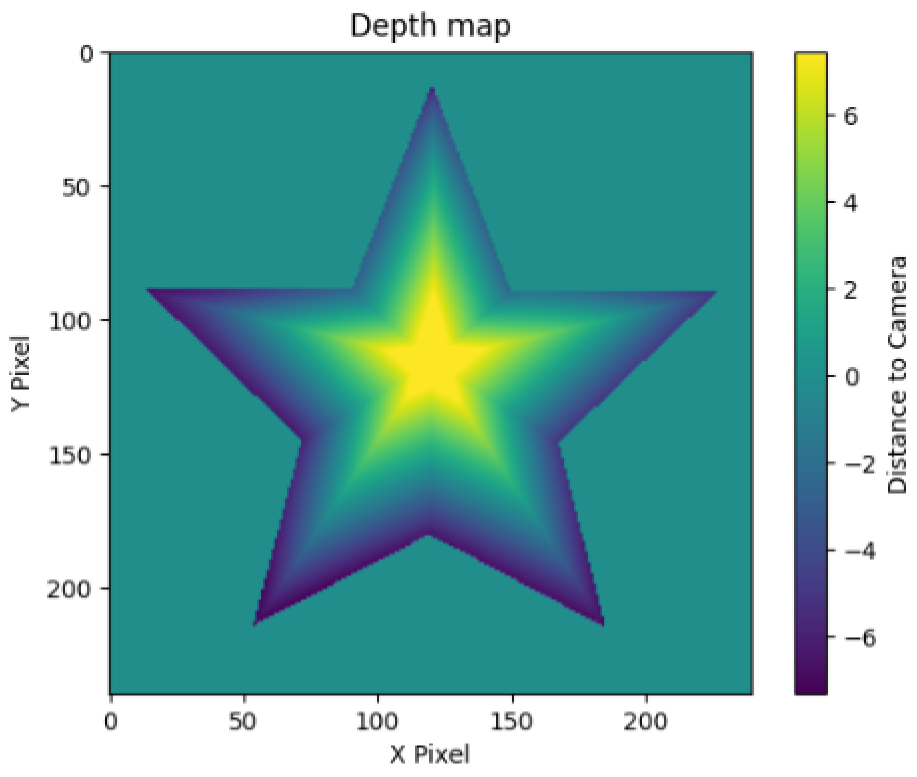
2. Show the normal map and depth map of “bunny” & “star”

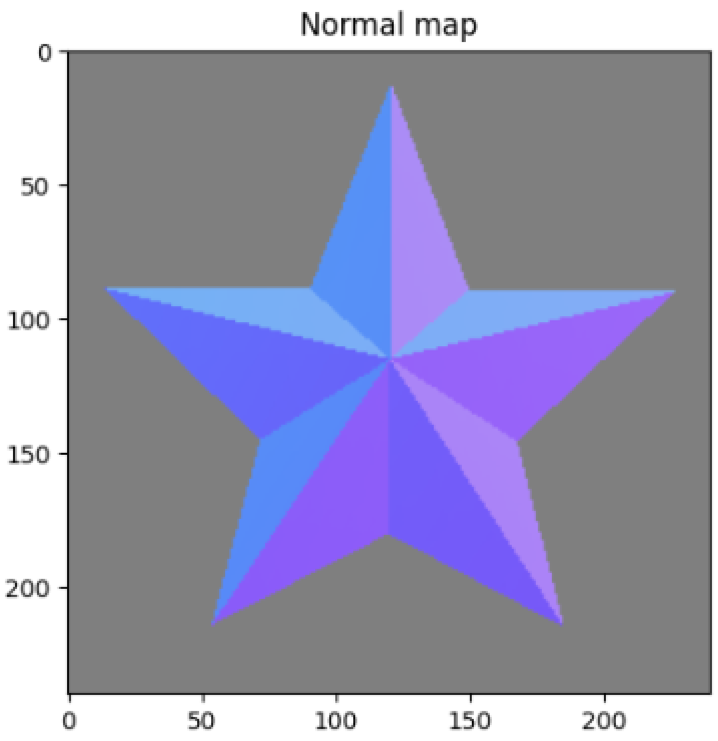
(a). bunny




Normal map
0
20
40
60
80
100
0
20
40
60
80
100

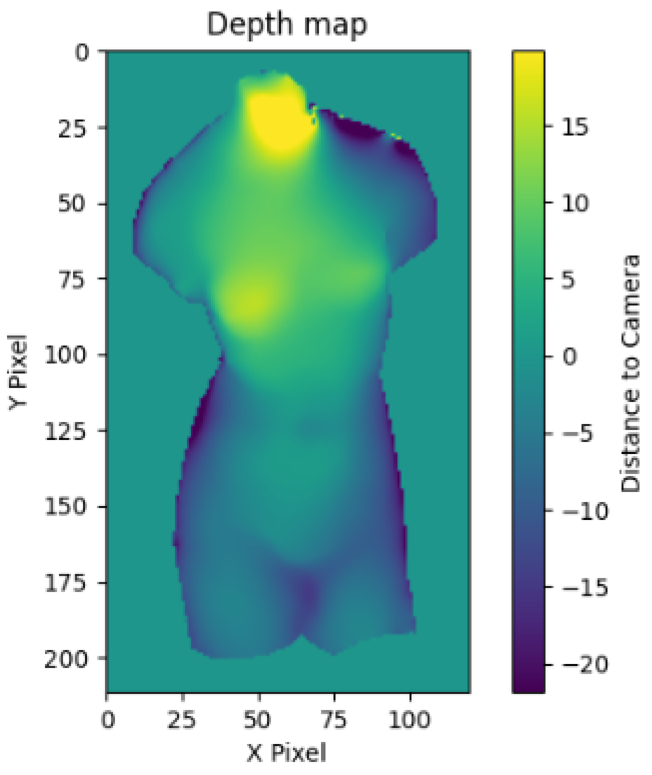
(b). star

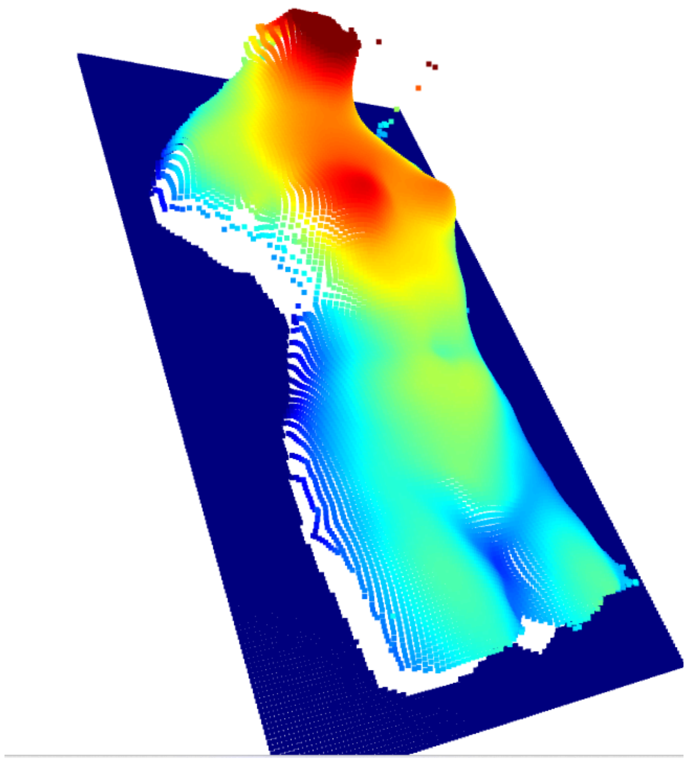

Depth map
Y Pixel
X Pixel
0
50
100
150
200
0
50
100
150
200
Distance to Camera
6
4
2
0
−2
−4
−6



3. Explain how I reconstruct surfaces of “venus”

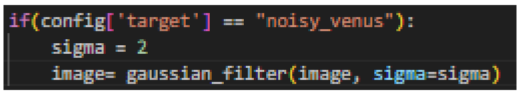
I utilize the trick that I mentioned in question 1. I consider depths exceeding two standard deviations from the mean of the entire image as outliers, and replace them with mean ± std. Due to this trick, we can get ideal result.





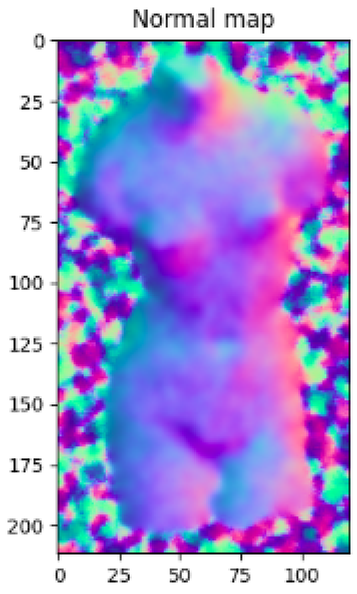
4. **Reconstruct surfaces of “noisy venus” where Gaussian noise has been applied to the input images.**

Because of the noise, we can not reconstruct the surface accurately, Thus, I denoise the gaussian noise and then reconstruct the surface.

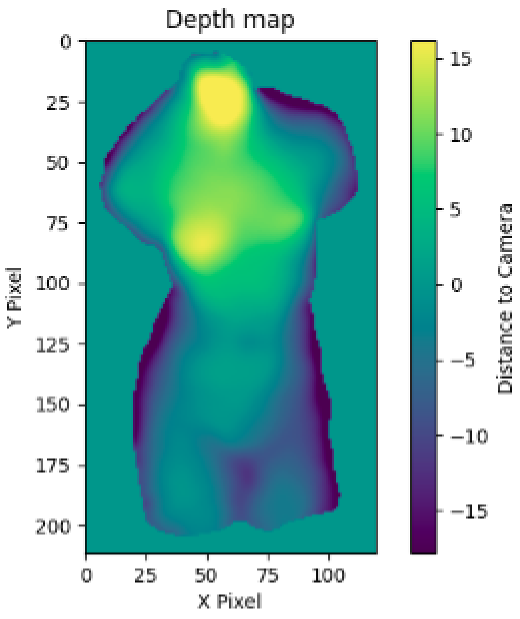


```
if(config['target'] == "noisy_venus"):
    sigma = 2
    image= gaussian_filter(image, sigma=sigma)
```

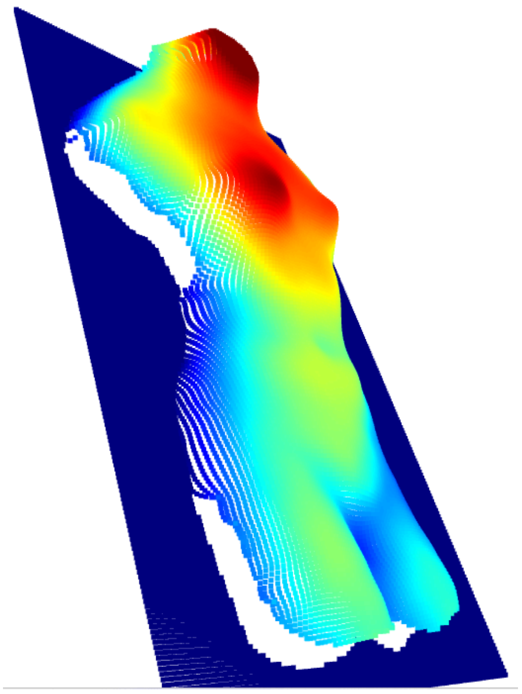
(a). Denoise venus’ normal vector:



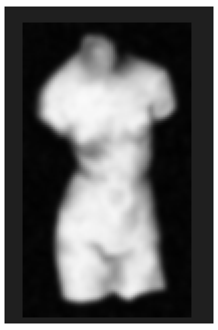
(b). Denoise venus’ depth image:

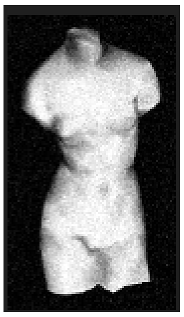


(c). Denoise venus' ply image:



(d). Comparison between denoised image and noisy image:





From this comparison, you can find that **although we can remove environmental noise through denoising, it also removes details from the image, so it shouldn't be too thorough.**