Computer Vision-HW1 Photometric Stere

109350008 張詠哲

Part 1.

Simply explain your implementation and what kind of "method" you use to enhance the result and compare the result (total result will be show in the end of the report)

1. For normal map

As the equation given by SPEC. I implement the equation:

$$K_d N = (L^T L)^{-1} L^T I$$

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

(I for image, L for light source)

```python
def calculate_normal(I, L):
    # KdN = (LT ∘ L)-1  ∘  LT  ∘  I
    KdN = np.dot(np.dot(np.linalg.inv(np.dot(L.T, L)), L.T), I)
    KdN_norm = np.linalg.norm(KdN, axis=0).reshape(image_row*image_col, 1)

    NM = KdN.T/KdN_norm
    return NM
```

2. For depth map

I implement the method 2 (**Surface Reconstruction 2**), there are some brief descript about my implementation.

We have to solve the equation (detail explanation in stereo (wisc.edu)):

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

But the M is a very large and sparse matrix. Therefore I create a mask to indicate which pixels' normals are valid (non-NaN values) and which ones are invalid (NaN values)

```python
def get_normal_mask(normal_map):
    normal_map = normal_map.reshape(image_row, image_col, 3)
    mask = np.where(np.isnan(normal_map[:, :, 0]), 0, 1)

    return mask
```

And use "check_use" to get the pixel that need to be calculate to reduce the M size.

```
M_filtered = M[:, check_use.astype(bool)]
z = np.dot(np.dot(np.linalg.inv(np.dot(M_filtered.T, M_filtered)), M_filtered.T), V)
```

Full implementation

```
def calculate_depth(N, mask, scale = 50, threshold = 3):
    M = []
    V = []
    check_use = np.zeros((image_row * image_col))
    N = N.reshape(image_row, image_col, 3)

    for i in range(image_row):
        for j in range(image_col-1):
            if mask[i][j] == 0:
                continue

            tmp = np.zeros((image_row * image_col))
            tmp[i * image_col + j] = -1
            tmp[i * image_col + j + 1] = 1
            check_use[i * image_col + j] = 1
            check_use[i * image_col + j + 1] = 1

            M.append(tmp)
            # V.append(-N[i][j][0] / N[i][j][2])
            maxi = max(-N[i][j][0] / N[i][j][2], -threshold)
            mini = min(maxi, threshold)
            V.append(mini)

            if mask[i][j + 1] == 0:
                tmp = np.zeros((image_row * image_col))
                tmp[i * image_col + j + 1] = 1
                check_use[i * image_col + j + 1] = 1
                M.append(tmp)
                V.append(0)
```

```
for i in range(image_col):
    for j in range(image_row-1):
        if mask[j][i] == 0:
            continue

        tmp = np.zeros((image_row * image_col))
        tmp[j * image_col + i] = -1
        tmp[(j + 1) * image_col + i] = 1
        check_use[j * image_col + i] = 1
        check_use[(j + 1) * image_col + i] = 1

        M.append(tmp)
        # V.append(N[j][i][1] / N[j][i][2])
        maxi = max(N[j][i][1] / N[j][i][2], -threshold)
        mini = min(maxi, threshold)
        V.append(mini)

        if mask[j + 1][i] == 0:
            tmp = np.zeros((image_row * image_col))
            tmp[(j+1) * image_col + i] = 1
            check_use[(j+1) * image_col + i] = 1
            M.append(tmp)
            V.append(0)

M = np.array(M, dtype=np.float32)
V = np.array(V, dtype=np.float32).reshape(-1, 1)

# print(M.shape)
# print(check_use.shape)
```

```
    # print(M.shape)
    # print(check_use.shape)
    M_filtered = M[:, check_use.astype(bool)]
    z = np.dot(np.dot(np.linalg.inv(np.dot(M_filtered.T, M_filtered)), M_filtered.T), V)

    idx = 0
    depth_map = []
    z_max = np.max(z)
    z_min = np.min(z)
    z_mid = (z_max+z_min)/2

    for i in range(image_row*image_col):
        if check_use[i] == 1:
            depth_map.append((z[idx][0] - z_mid)*scale / (z_max - z_min))
            idx += 1
        else:
            depth_map.append(0.0)

    depth_map = np.array(depth_map, dtype=np.float32)

    return depth_map
```

3. method to enhance the result

3.1 low pass filter

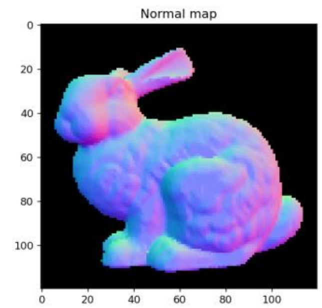
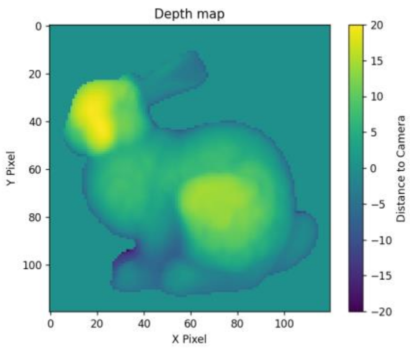
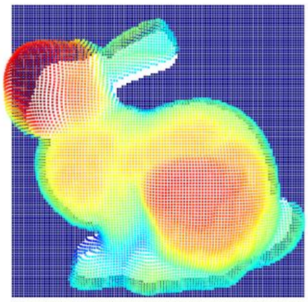
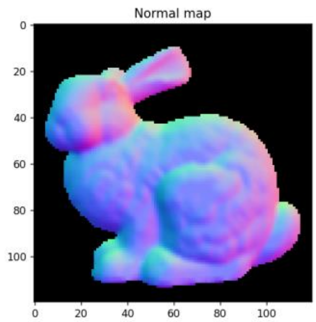
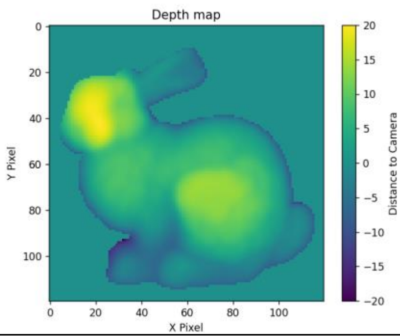
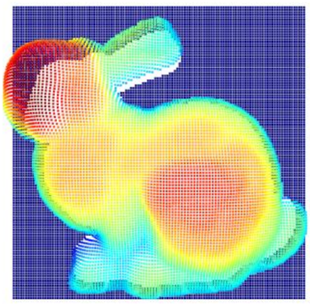
To reduce unnecessary distortion and rugged surface. I add a low pass filter to make the surface smoother (I've also try the high pass filter, but is terrible…).

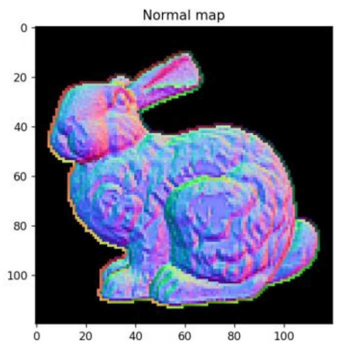
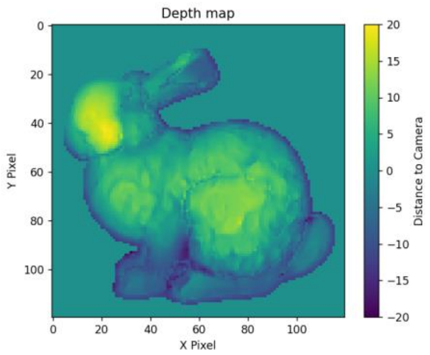
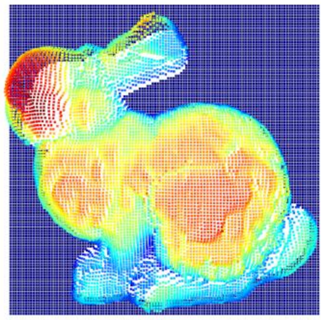
```python
def low_pass_filter(img):
    kernel = np.ones((3, 3), dtype=np.float32) / 9
    pad_image = np.pad(img, 1, mode='edge').astype(np.float32)
    low_pass_image = np.zeros_like(img, dtype=np.float32)

    for i in range(1, pad_image.shape[0]-1):
        for j in range(1, pad_image.shape[1]-1):
            low_pass_image[i-1, j-1] = np.sum(pad_image[i-1:i+2, j-1:j+2] * kernel)

    return low_pass_image
```

```python
def high_pass_filter(img):
    kernel = np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]], dtype=np.float32)
    pad_image = np.pad(img, 1, mode='edge').astype(np.float32)
    high_pass_image = np.zeros_like(img, dtype=np.float32)

    for i in range(1, pad_image.shape[0]-1):
        for j in range(1, pad_image.shape[1]-1):
            high_pass_image[i-1, j-1] = np.sum(pad_image[i-1:i+2, j-1:j+2] * kernel)

    return high_pass_image
```

Compare

And as the following compare table can see. Low pass make the normal and reconstruct result smoother. I think the low pass enhance the result.

Origin			
Low pass filter			

High pass filter			

3.2 Venus

To deal with some extrem normal result. I add a pixel mask. To mask the pixels below the threshold. And I found that if use origin equation $V = \left[\dots -\frac{n_x}{n_z} \dots\dots -\frac{n_y}{n_z} \dots\right]$, the nipple will not be well reconstructed. Therefore I add a threshold in depth map part to limit the $-\frac{n_x}{n_z}$ and $-\frac{n_y}{n_z}$ in a range

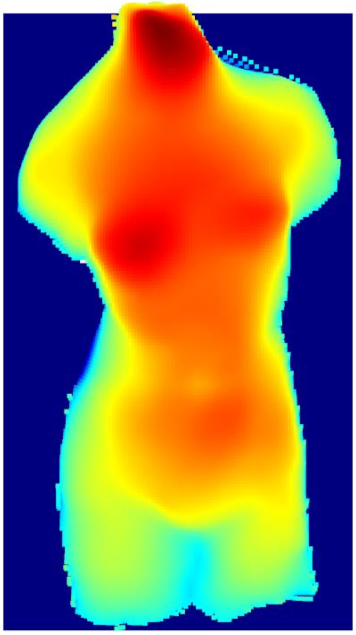
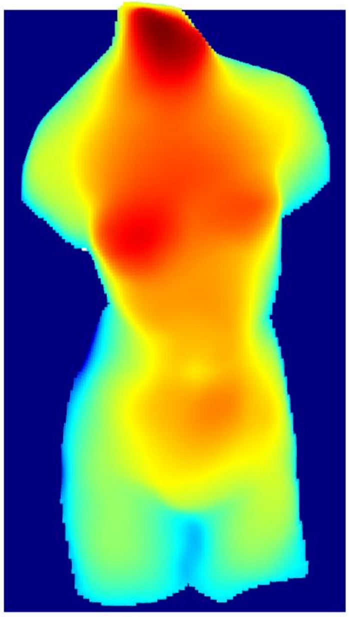
```
def pixel_mask(img, thres_scale = 20):
    '''
    Mask image where pixels below the threshold
    '''
    pixel_sum = np.zeros(img[0].shape)
    for _img in img:
        pixel_sum += _img

    threshold = len(img) * thres_scale
    mask = np.where(pixel_sum < threshold, 0, 1)

    masked_images = [_img * mask for _img in img]
    masked_images = np.array(masked_images, dtype = np.float32)

    return masked_images
```

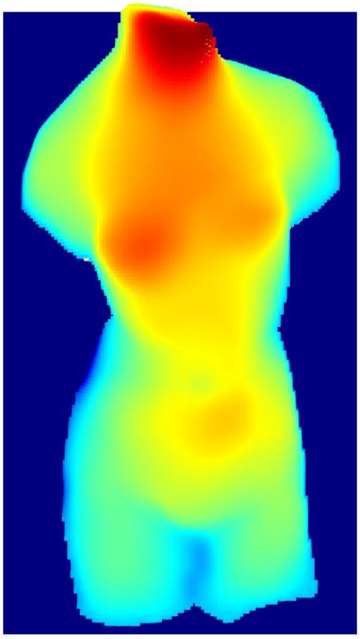
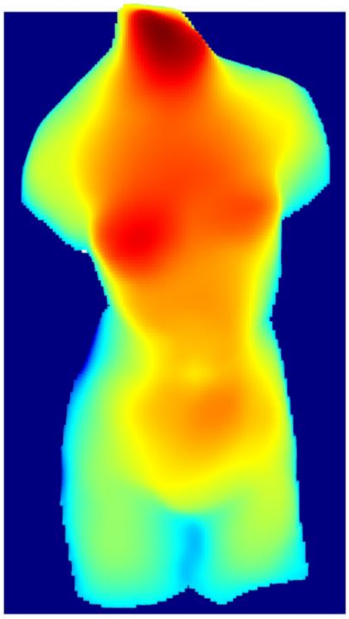
```
# V.append(-N[i][j][0] / N[i][j][2])
maxi = max(-N[i][j][0] / N[i][j][2], -threshold)
mini = min(maxi, threshold)
V.append(mini)
```

Compare

As the following compare table can see. The origin graph edge has many jagged edges and also many strange points. After pixel mask, the weird point are disappear but the nipple become more smaller.

Origin	Pixel mask

Origin	Add threshold
	

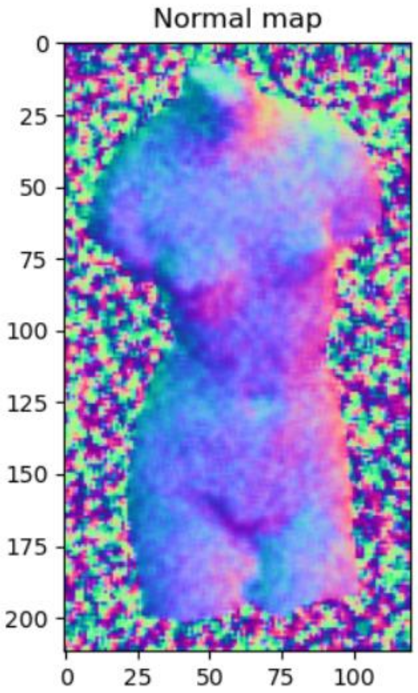
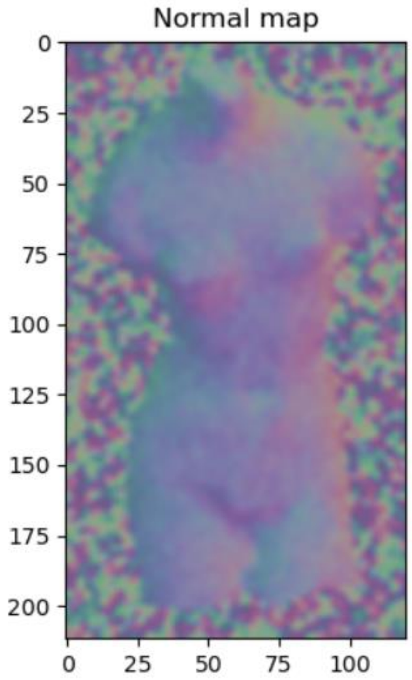
3.3 noisy venus

In this part, gaussian noise has been applied to input. Therefore I take 2 step to deal with the noise. First I add a gaussian filter to smooth and lower the noise. Next I add the pixel mask I mention in “venus” part.

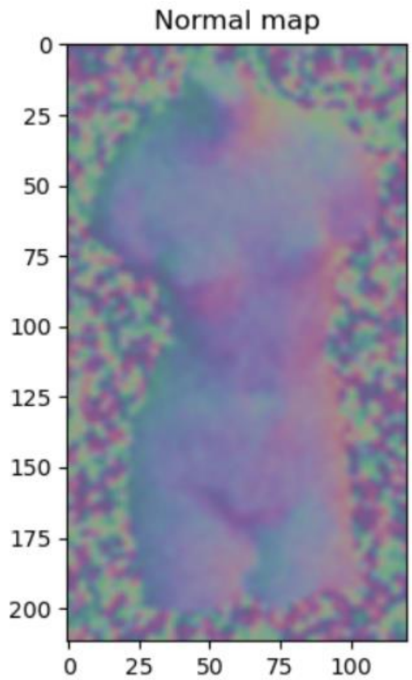
```python
def gaussian_filter(img):
    kernel = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]], dtype=np.float64)
    kernel /= np.sum(kernel)  # Normalize the kernel

    pad_image = np.pad(img, 1, mode='edge').astype(np.float64)
    gau_image = np.zeros_like(img, dtype=np.float64)

    for i in range(1, pad_image.shape[0]-1):
        for j in range(1, pad_image.shape[1]-1):
            gau_image[i-1, j-1] = np.sum(pad_image[i-1:i+2, j-1:j+2] * kernel)

    return gau_image
```

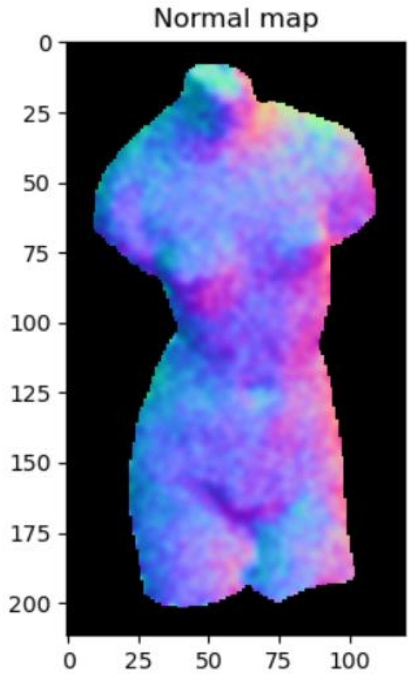
Compare

As the following compare table can see. After filtering by gaussian filter. The noise seems to be lower. But the noise is still here. And after pixel mask, the background noise is well removed. And I found that different threshold scale in pixel mask will affect the normal map result (as indicated by the red box).

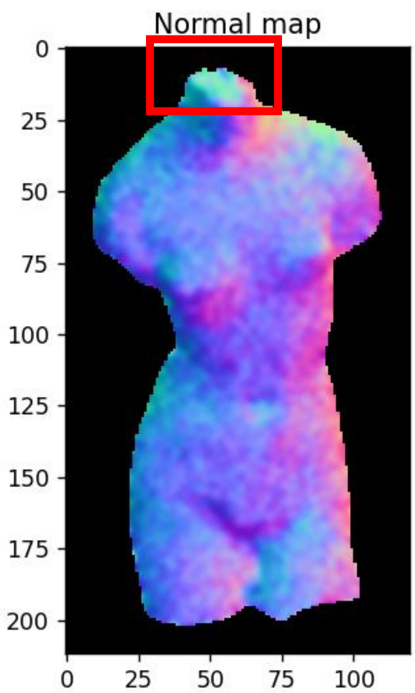
No gaussian filter	Gaussian filter
	

No pixel mask	Pixel mask

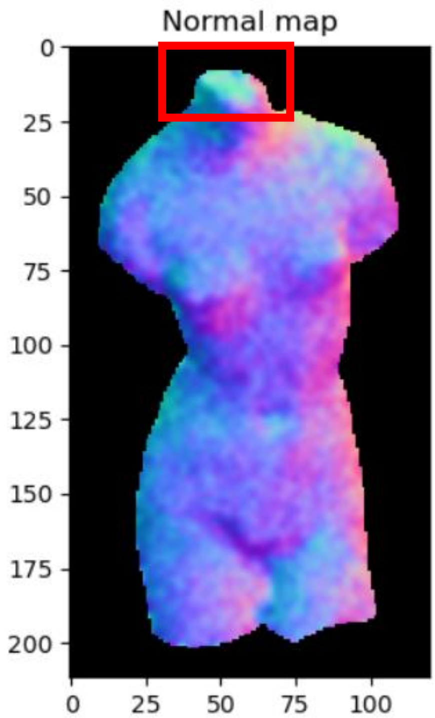




Threshold scale = 10

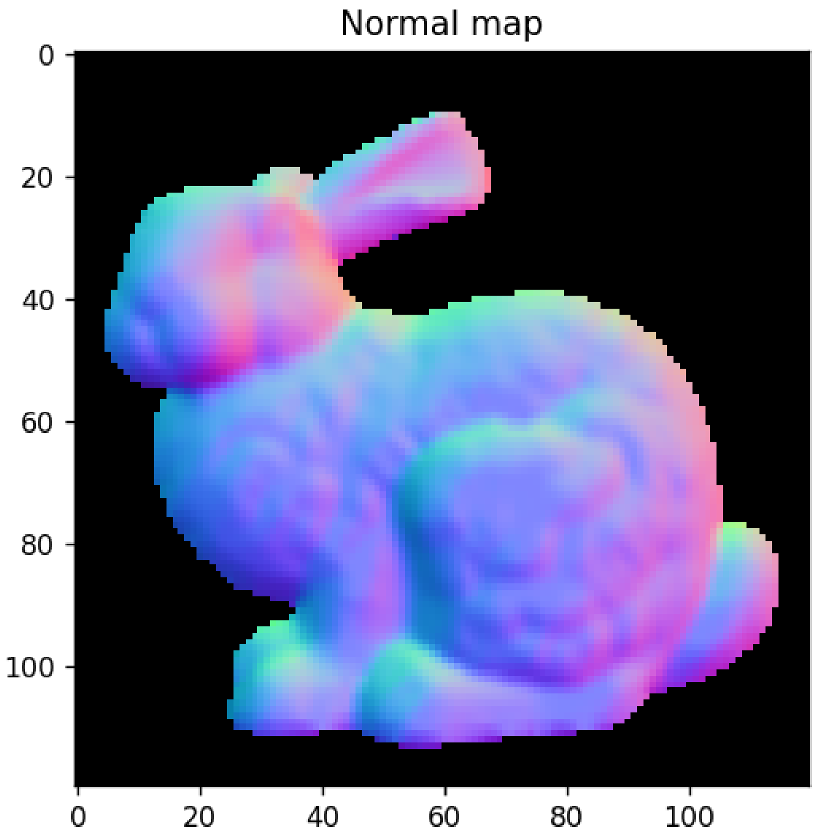


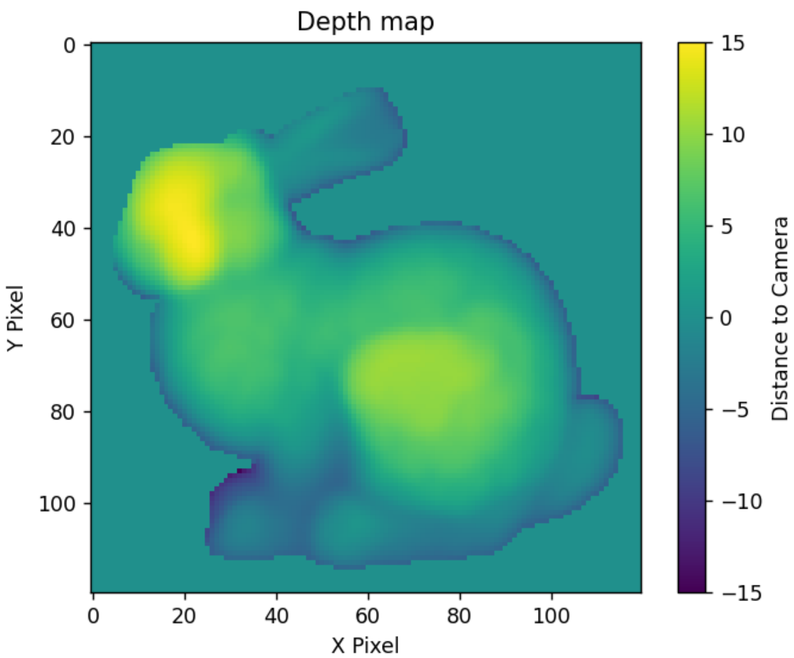
Threshold scale = 20

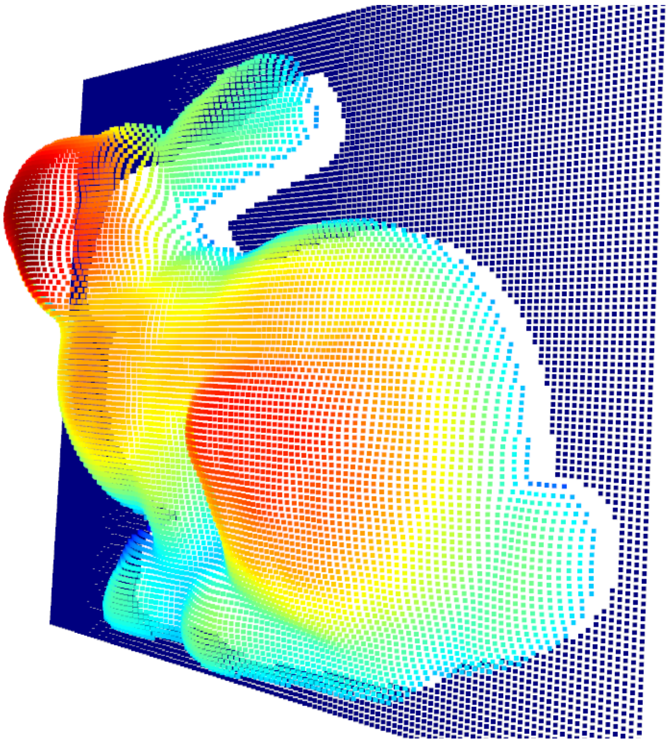


Part 2
Total result

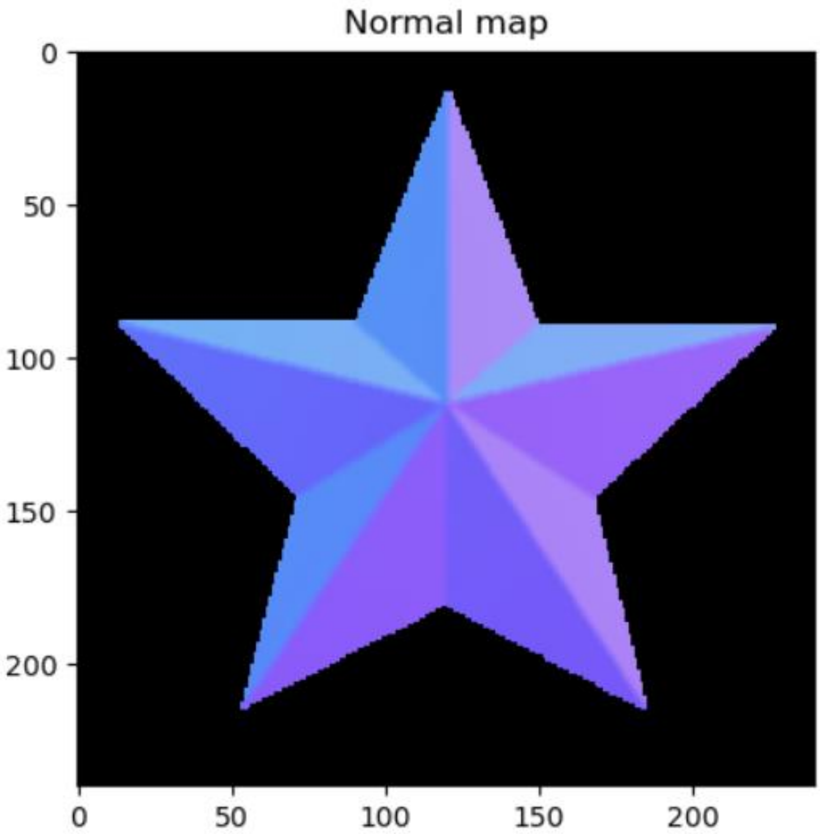
1. Bunny

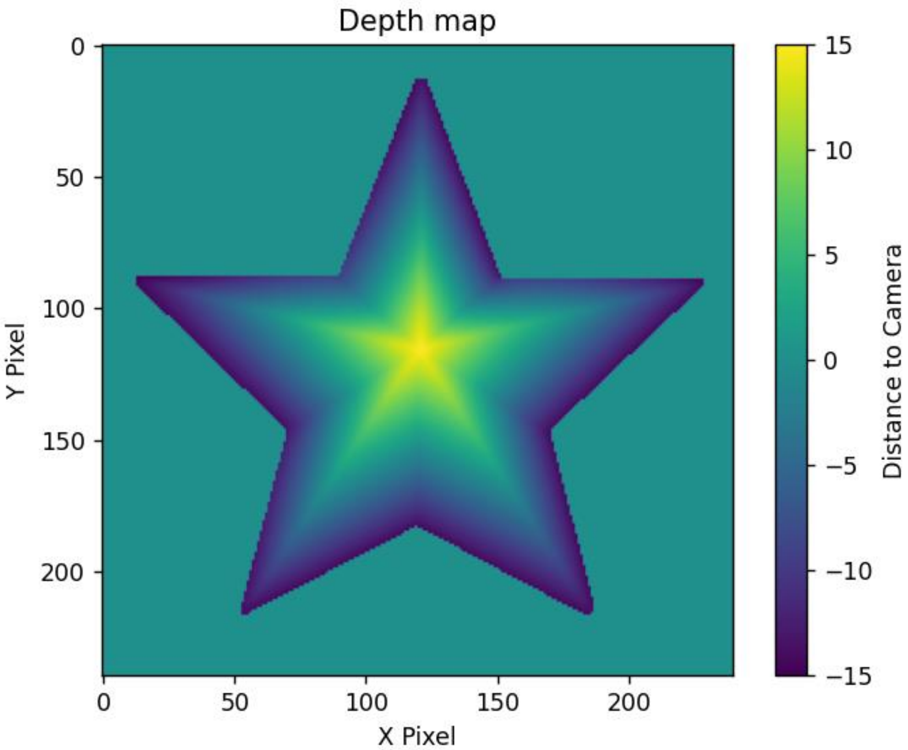


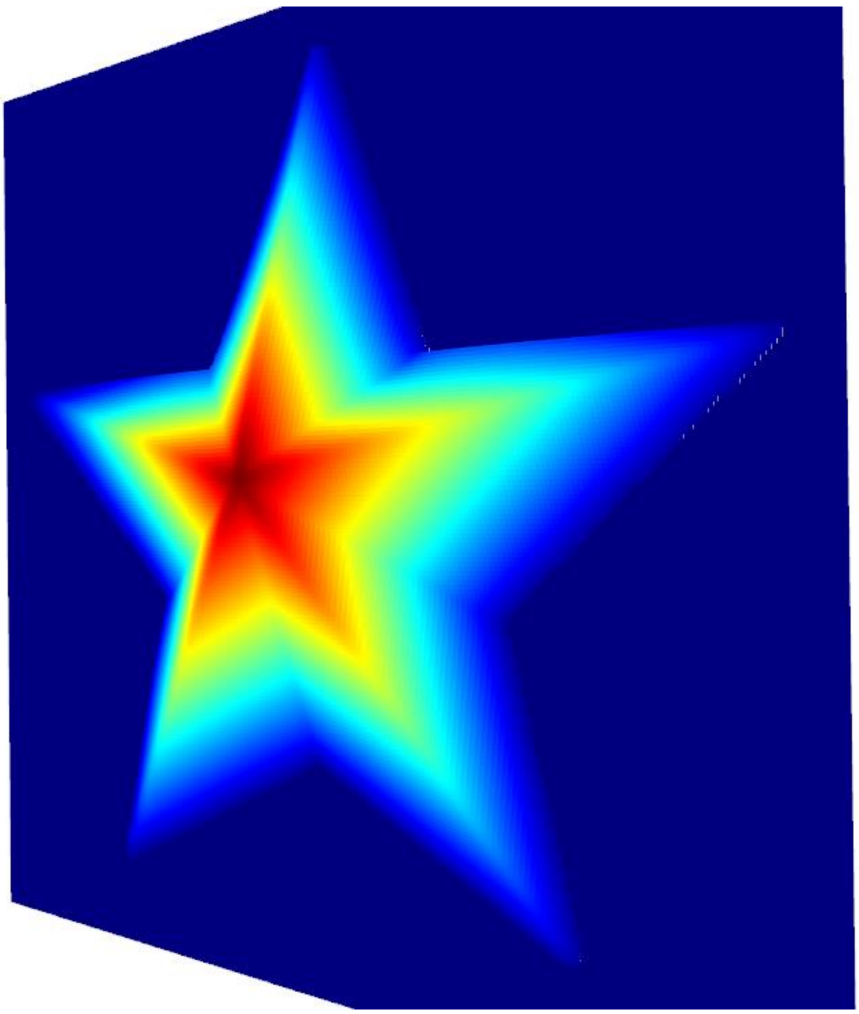




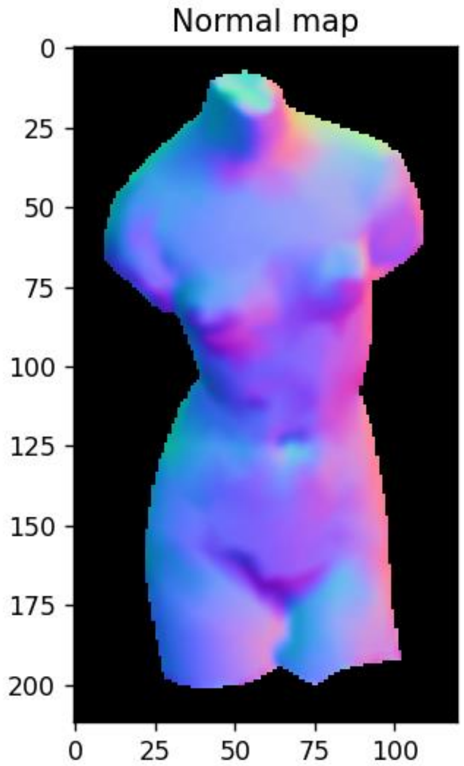
2. Star



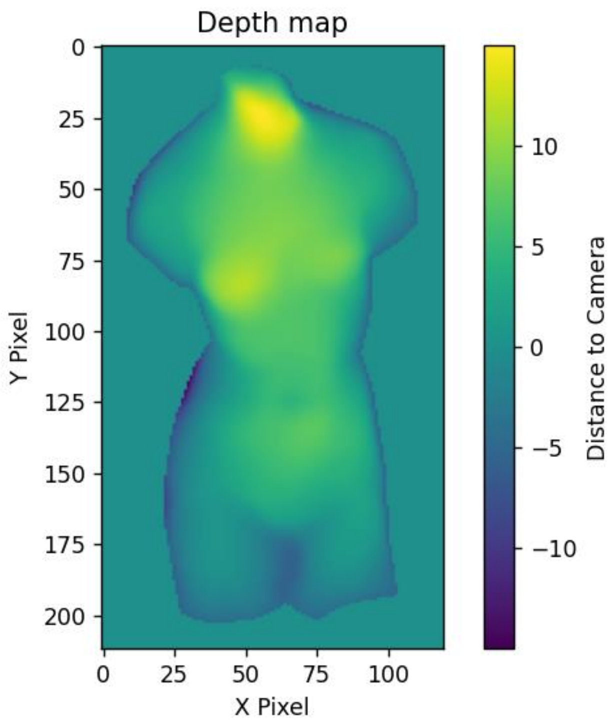

Depth map
0
50
100
150
200
Y Pixel
0
50
100
150
200
X Pixel
15
10
5
0
−5
−10
−15
Distance to Camera



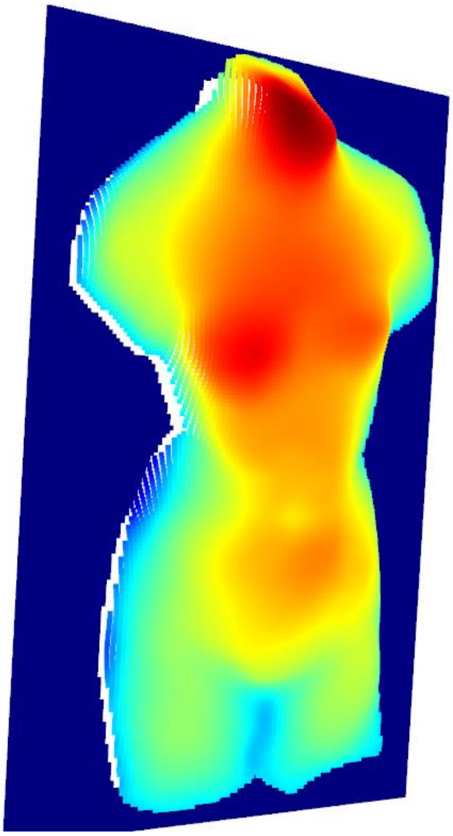
3. Venus



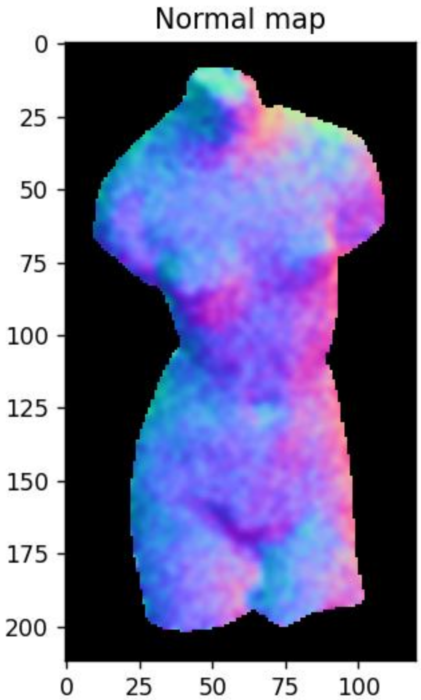
Normal map
0
25
50
75
100
125
150
175
200
0
25
50
75
100

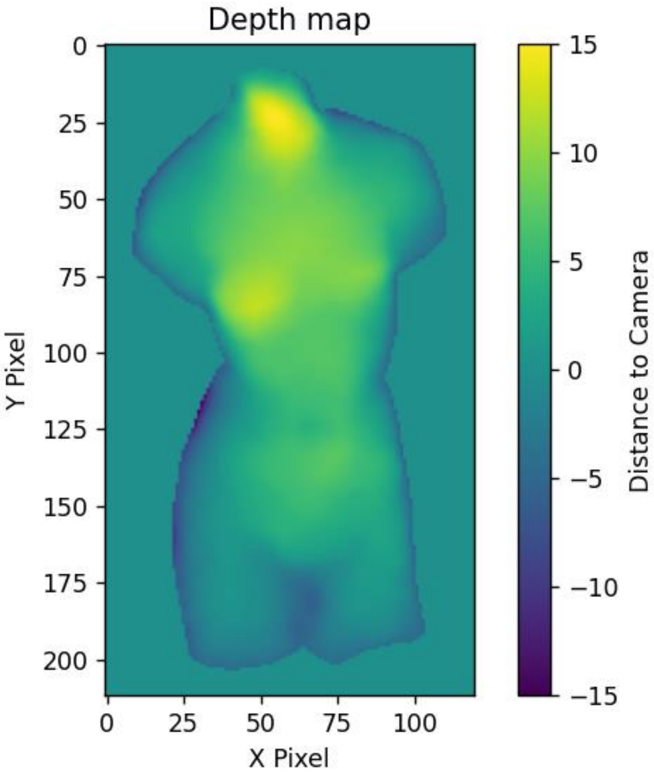


Depth map
Y Pixel
X Pixel
Distance to Camera
0
25
50
75
100
125
150
175
200
0
25
50
75
100
10
5
0
−5
−10



4. Noisy venus




Depth map
Y Pixel
X Pixel
Distance to Camera
0
25
50
75
100
125
150
175
200
0
25
50
75
100
15
10
5
0
−5
−10
−15