Computer Vision HW1 Report

310551079 梁友誠

Part1. Implementation

1. **讀入圖片及光源:**

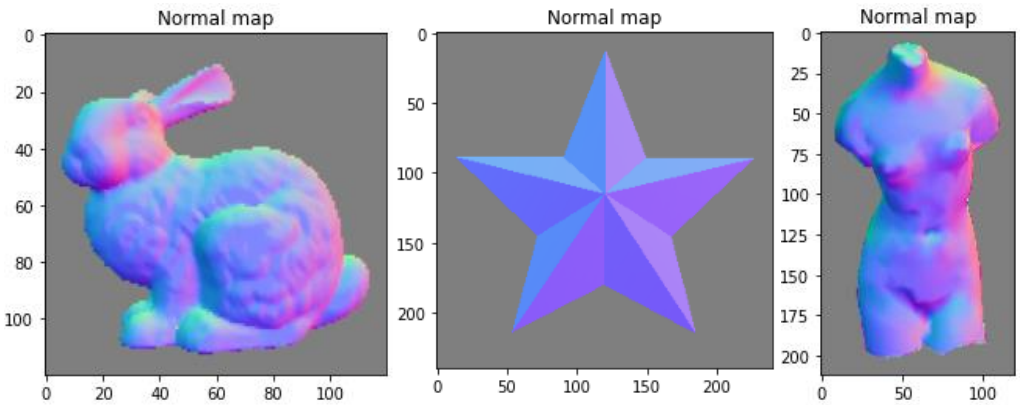
   針對要計算的資料來使用 PIL 套件讀入在./test/目錄下的 6 張圖片並轉成灰階圖(原始圖檔為 RGB 格式)存到 images(h*w*6)，以及讀取./test/目錄下的 LightSource.txt 並用 regular expression (re) 套件來處理檔案中的字串，將裡面的光源數值存入 lights(6*3) 中。

2. **Create mask:**

   將步驟 1 所得到的 6 張灰階圖片利用 opencv 提供的 cv2.threshold()來做影像二值化(image thresholding)，其目的為在最後得到影像的表面圖時可以利用 mask 來將不屬於原本影像範圍的表面給消除，這樣可得到完整物體的表面而不會有其它因計算誤差而產生的表面留在深度圖中。其做法為將每張灰階圖中灰階值大於 0 的 pixel 設為 1 當成是 mask，並將這 6 張圖的 mask 結合成單一張 mask 作為最後的 mask(大小為 h*w)。

3. Normal Estimation:

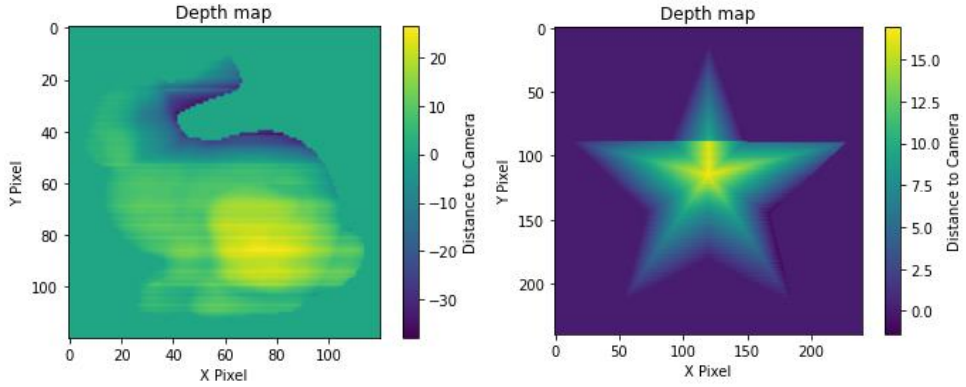
利用步驟 1 所得到的 images 以及 lights 並根據 reflection model 來計算整體圖片的 Normal。其做法為先 normalize 輸入的光源(若沒有做 normalization 會得到錯誤的 normal map)，之後根據圖片的每個位置(pixel)計算該位置的 normal vector，其作法為針對每個 pixel 取其 6 張圖片的灰階值當作 intensity vector(大小為 6*1)，並利用輸入的光源 lights(6*3)來計算 Ax=B，其中 A=光源，B=intensity vector，而 x 就是要求的 normal vector，此計算可利用 numpy 的 least-squares solution (numpy.linalg.lstsq)來做。因此可求出每個 pixel 的 normal vector(3*1)。而此時求出來的 normal 還須轉為 unit normal vector 才可，因此將每個 normal vector 除以 norm (albedo)才是最後計算後要回傳的 normal map。下圖為 bunny，star，venus 所計算的 normal map。

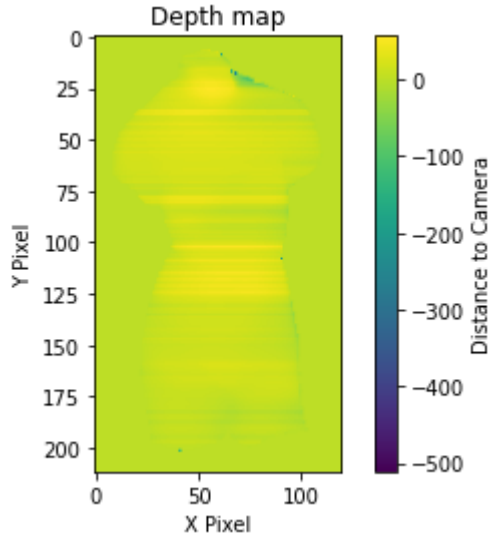


Normal map result

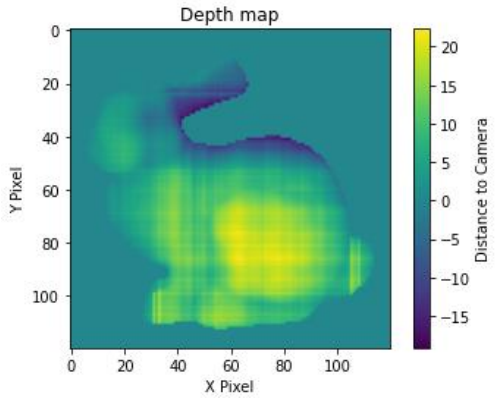
4. Surface Reconstruction:

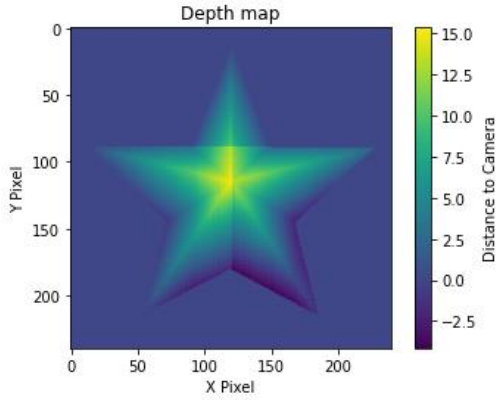
我採用積分法來完成這次作業的表面重建，首先用步驟 3 所得到的 normal map 來計算每個 pixel 的 x 軸梯度(-n1/n3)以及 y 軸梯度(n2/n3)，因為圖片的原點(0,0)是在左上角的部分，因此在求 y 軸梯度時不採用原公式的-n2/n3，而是用 n2/n3 才可得到正確的結果。在一開始重建表面時，我使用圖片的左上角(x,y)=(0,0)為起點，沿著 y 軸梯度積分計算出圖片中 column = 0 的起始高度，再利用此起始高度沿著 x 軸梯度從左到右積分計算出每個 pixel 的高度(h_left2right)，可得到如下的結果。

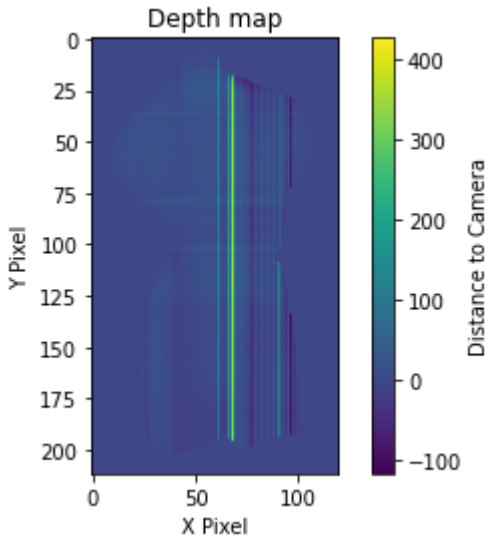




從以上結果可看出只使用 x 軸梯度積分所算出的高度有很明顯的橫向條紋，並且在 star 的高度圖中有明顯的斷層存在，因此除了考慮用 x 軸積分外也需考慮 y 軸的積分才可得到較正確的結果。同樣以圖片的左上角(x,y)=(0,0)為起點，先沿著 x 軸梯度積分計算出圖片中 row = 0 的起始高度，再利用此起始高度沿著 y 軸梯度從上到下積分計算出每個 pixel 的高度(h_top2bottom)，在把此結果與剛剛所得到的 h_left2right 取平均可得到如下結果。可看出 bunny 以及 star 的結果有變得較好，而 venus 則是得到較差的結果(表面突然突起)，因此上述方法仍有改善的空間。

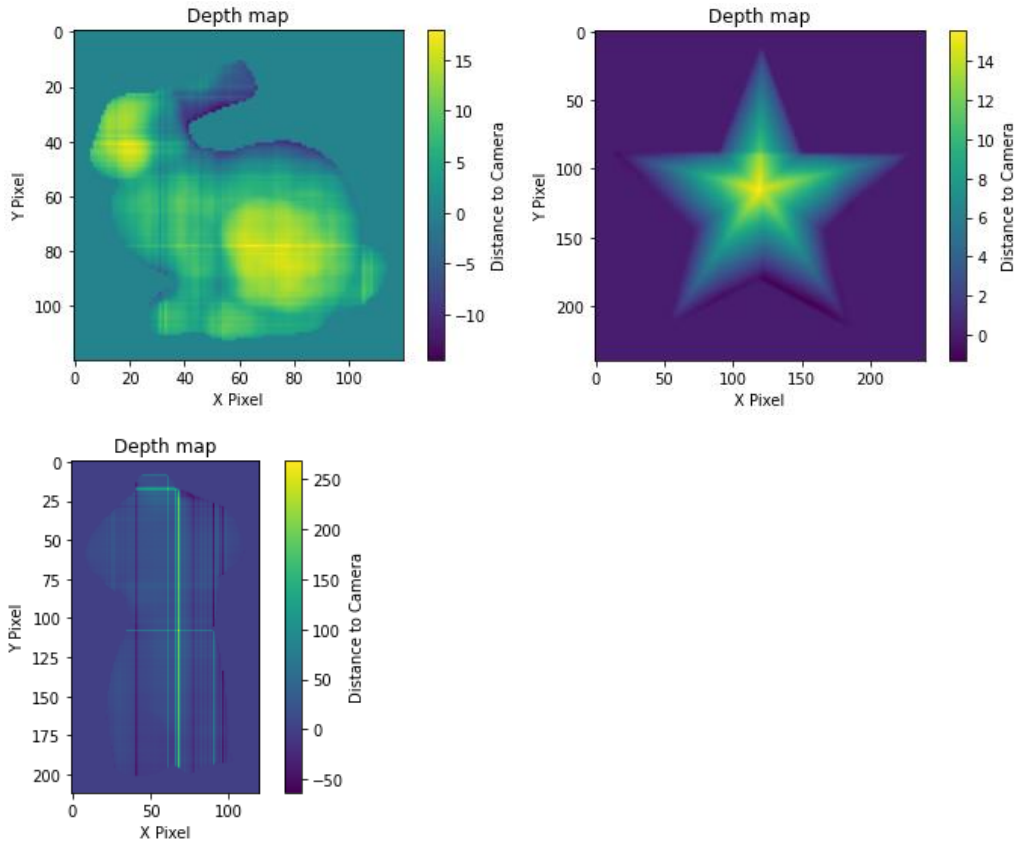






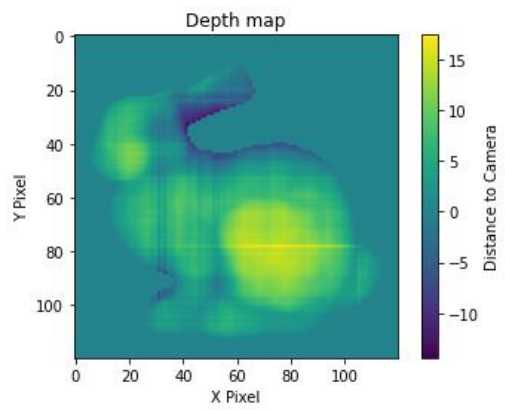
5. Surface Reconstruction (Smooth):

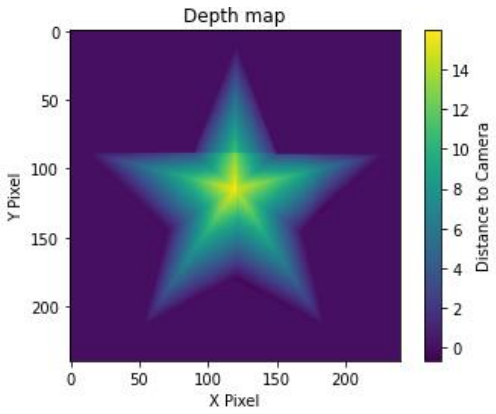
為了得到更平滑的結果，除了從圖片左到右，從上到下的積分外也需考慮另一方向(從右到左，從下到上)的積分，這樣才可藉由相反方向的積分來得到更正確的高度。因此在實作時，我採用圖片右上角(x,y)=(w,0)為原點，分別沿著 y 軸求出 column = w 的原始高度再沿著 x 軸梯度從右到左積分計算出每個 pixel 的高度(h_right2left)，也利用 x 軸梯度積分計算出圖片中 row = h 的起始高度，利用此起始高度沿著 y 軸梯度從下到上積分計算出每個 pixel 的高度(h_bottom2top)。最後利用步驟 4 的 h_left2right 與 h_top2bottom 取平均(h_left2right + h_top2bottom + h_right2left + h_bottom2top)/4 來得到最後的結果。

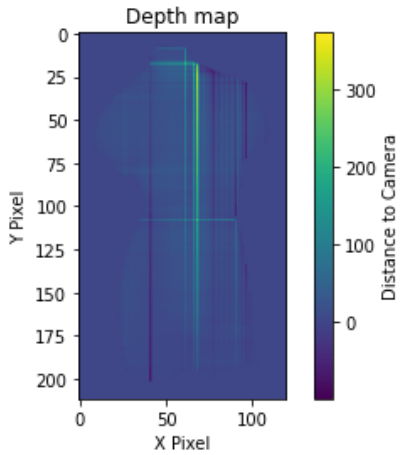


上面結果與步驟 4 所得到的結果相比又更加平滑，因此使用多個不同方向的積分來求高度可獲得更好的結果，對於 bunny 來說鼻子那邊的高度有更加明顯，而對於 star 來說則是消除了較明顯的斷層，至於 venus 則是維持不好的結果。

除了此用全部平均的方法外，我也使用了對於不同方向的積分給予不同的權重來計算高度，舉例來說從左到右的積分中，越靠近左邊的 pixel 高度我就給予較高的權重，而越靠近右邊的 pixel 高度我就給予較低的權重。下圖為利用此方法所得出的結果，可看出與平均法的差異並不大，但用此方法可降低高度差距過大所造成的不平滑，因此這也是我最後採用的方法。







6. Conclusion:

經過各種不同方法的測試，使用多個方向的梯度積分並給予不同權重能得到較平滑的結果，但對於本身就存在雜訊或著落差太大的資料則無法利用積分法的方式求出較好的結果，而會使得結果高度有明顯落差。下圖為我在這次作業中所計算出的 ply 結果。