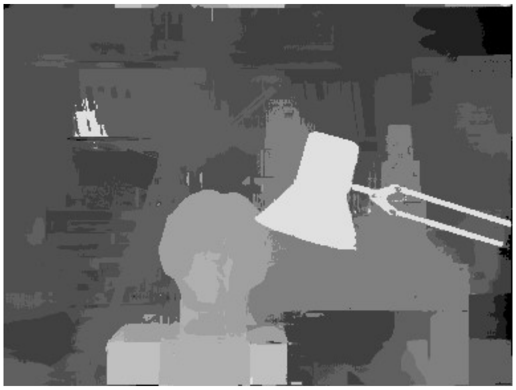
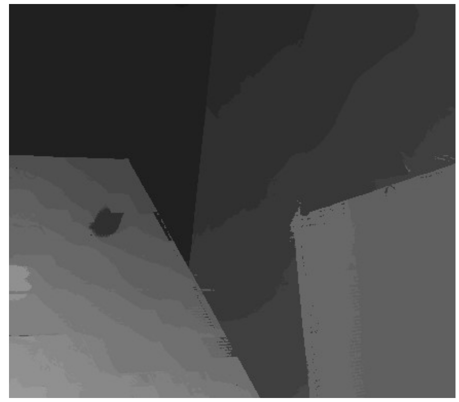
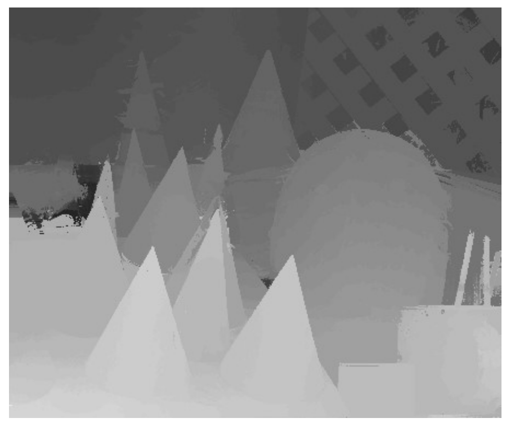
Computer Vision HW4 Report

Student ID: B10505047
Name: 邱郁喆

Visualize the disparity map of 4 testing images.

Tsukuba	Venus
	
Teddy	**Cones**
	

Report the bad pixel ratio of 2 testing images with given ground truth (Tsukuba/Teddy).

	bad pixel ratio
Tsukuba	4.56%
Teddy	13.97%

Describe your algorithm in terms of 4-step pipeline.

Step 1. cost computation : 先對 image 做 padding，再直接用 for 迴圈去對左右圖中每個像素值取得 local binary pattern，最後也一樣是用 for loop 去求 hamming distance 進而得到 census cost

```python
# image padding
padded_Il = cv2.copyMakeBorder(Il, 1, 1, 1, 1, borderType=cv2.BORDER_CONSTANT, value=0) # (h+2, w+2, ch)
padded_Ir = cv2.copyMakeBorder(Ir, 1, 1, 1, 1, borderType=cv2.BORDER_CONSTANT, value=0)

# image patch -> local binary pattern
Il_code = np.zeros((h, w, 8, ch), dtype=np.bool) # type should be bool in order to use ^ (xor)
Ir_code = np.zeros((h, w, 8, ch), dtype=np.bool)
for i in range(1, h+1):
    for j in range(1, w+1): # clockwise order (start from the left point)
        Il_code[i-1, j-1, 0] = padded_Il[i-1, j-1] < padded_Il[i, j]
        Il_code[i-1, j-1, 1] = padded_Il[i-1, j] < padded_Il[i, j]
        Il_code[i-1, j-1, 2] = padded_Il[i-1, j+1] < padded_Il[i, j]
        Il_code[i-1, j-1, 3] = padded_Il[i, j+1] < padded_Il[i, j]
        Il_code[i-1, j-1, 4] = padded_Il[i+1, j+1] < padded_Il[i, j]
        Il_code[i-1, j-1, 5] = padded_Il[i+1, j] < padded_Il[i, j]
        Il_code[i-1, j-1, 6] = padded_Il[i+1, j-1] < padded_Il[i, j]
        Il_code[i-1, j-1, 7] = padded_Il[i, j-1] < padded_Il[i, j]

        Ir_code[i-1, j-1, 0] = padded_Ir[i-1, j-1] < padded_Ir[i, j]
        Ir_code[i-1, j-1, 1] = padded_Ir[i-1, j] < padded_Ir[i, j]
        Ir_code[i-1, j-1, 2] = padded_Ir[i-1, j+1] < padded_Ir[i, j]
        Ir_code[i-1, j-1, 3] = padded_Ir[i, j+1] < padded_Ir[i, j]
        Ir_code[i-1, j-1, 4] = padded_Ir[i+1, j+1] < padded_Ir[i, j]
        Ir_code[i-1, j-1, 5] = padded_Ir[i+1, j] < padded_Ir[i, j]
        Ir_code[i-1, j-1, 6] = padded_Ir[i+1, j-1] < padded_Ir[i, j]
        Ir_code[i-1, j-1, 7] = padded_Ir[i, j-1] < padded_Ir[i, j]
```

```python
# Census cost = Local binary pattern -> Hamming distance
l_cost = np.zeros((h, w, max_disp+1), dtype=np.float32)
r_cost = np.zeros((h, w, max_disp+1), dtype=np.float32)

for d in range(max_disp + 1):
    for i in range(h):
        for j in range(w):
            if(j-d >= 0):
                l_cost[i, j, d] = np.sum((Il_code[i, j]^Ir_code[i, j-d]).astype(np.uint8), axis=(0, 1))
            else: # cannot go left d
                l_cost[i, j, d] = l_cost[i, d, d]
for d in range(max_disp + 1):
    for i in range(h):
        for j in range(w):
            if(j+d <= w-1):
                r_cost[i, j, d] = np.sum((Ir_code[i, j]^Il_code[i, j+d]).astype(np.uint8), axis=(0, 1))
            else: # cannot go right d
                r_cost[i, j, d] = r_cost[i, w-1-d, d]
```

Step 2. cost aggregation： 將上一步得到的 cost 過 joint bilateral filter，得到比較 smooth 的 cost

```python
# >>> Cost Aggregation
# TODO: Refine the cost according to nearby costs
# [Tips] Joint bilateral filter (for the cost of each disparty)
for d in range(max_disp+1):
    l_cost[:, :, d] = xip.jointBilateralFilter(Il, l_cost[:, :, d], -1, 4, 12)
    r_cost[:, :, d] = xip.jointBilateralFilter(Ir, r_cost[:, :, d], -1, 4, 12)
```

Step 3. Disparity optimization： 用 winner-take-all 的概念，對於每個 pixel 都選擇 cost 最小的 disparity

```python
# >>> Disparity Optimization
# TODO: Determine disparity based on estimated cost.
# [Tips] Winner-take-all
D_L = np.argmin(l_cost, axis=2) # (h, w)
D_R = np.argmin(r_cost, axis=2)
```

Step 4. Disparity refinement： 首先做 left-right consistency check，也就是對於左右圖的 disparity map D_L 和 D_R 檢查 consistency (對每個座標點看 $D_L(x,y) = D_R(x - D_L(x,y), y)$是否成立)，對於不成立的座標點將其標成 hole。下一步就是做 hole filling，對於每個 hole 分別往左和往右找第一個有 disparity 的點，並取兩者之中較小的值填入，在這一步中我有先對 D_L 在其左右邊界做 padding 使得不會有在左右邊界找不到值的問題。最後則是再讓 disparity map 過 weighted median filter，使結果更好。

```python
# >>> Disparity Refinement
# TODO: Do whatever to enhance the disparity map
# [Tips] Left-right consistency check -> Hole filling -> Weighted median filtering

# Left-right consistency check
# Note: D_R are only used in this step
for y in range(h):
    for x in range(w):
        if (x-D_L[y, x] >= 0) and (D_L[y, x] == D_R[y, x-D_L[y,x]]):
            continue   # keep the computed disparity
        else:
            D_L[y, x] = -1  # mark hole (invalid disparity)

# Hole filling
# pad 1 row each at leftmost and rightmost
padded_D_L = cv2.copyMakeBorder(D_L, 0, 0, 1, 1, cv2.BORDER_CONSTANT, value=max_disp) # (h, w+2)
for y in range(h):
    for x in range(w):
        if D_L[y, x] == -1:
            l = 1
            while(padded_D_L[y, x+1-l] == -1): # padded_D_L[y, x+1-l] = D_L[y, x-l]
                l += 1
            F_L = padded_D_L[y, x+1-l] # the disparity map filled by closest valid disparity from left

            r = 1
            while(padded_D_L[y, x+1+r] == -1): # padded_D_L[y, x+1+r] = D_L[y, x+r]
                r += 1
            F_R = padded_D_L[y, x+1+r] # the disparity map filled by closest valid disparity from right

            D_L[y, x] = min(F_L, F_R) # pixel-wise minimum

# Weighted median filtering
labels = xip.weightedMedianFilter(Il.astype(np.uint8), D_L.astype(np.uint8), 18, 3)

return labels.astype(np.uint8)
```