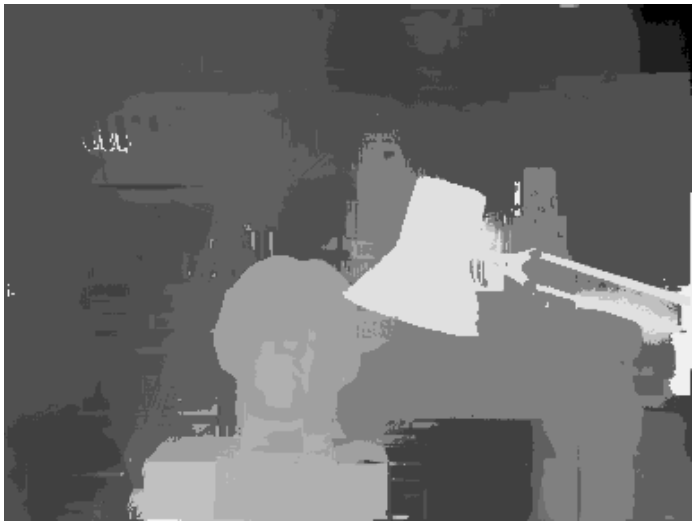
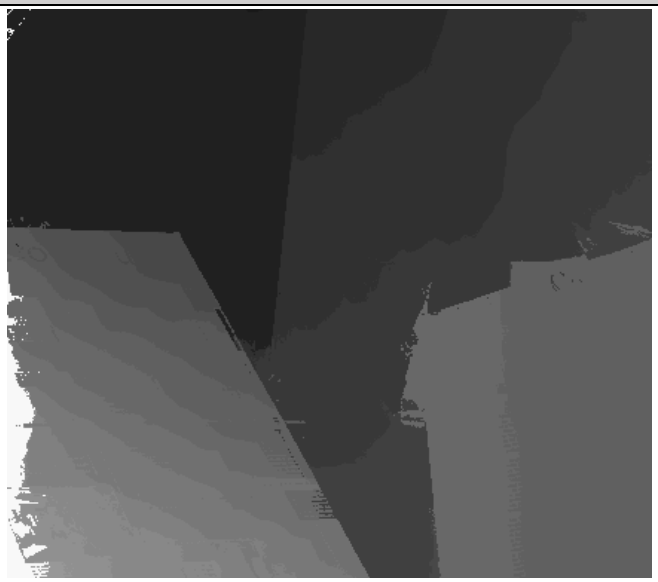
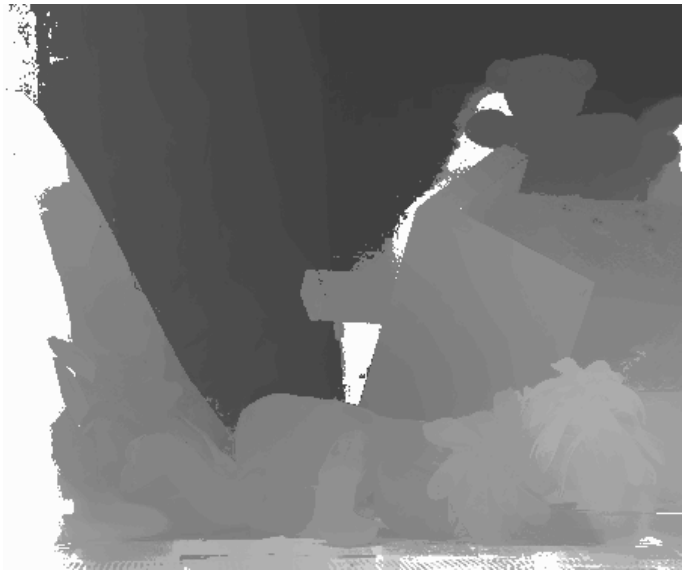
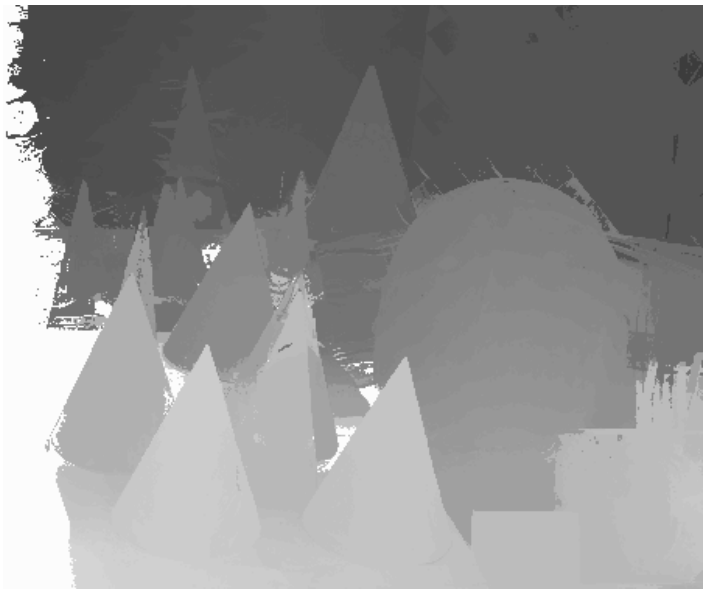
Computer Vision HW4 Report

Student ID: R09246003
Name: 林子敬

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	6.21%
Teddy	20.60%

Describe your algorithm in terms of 4-step pipeline.

我的流程完全照助教講解以及 code 提示的四個步驟

1.

我先把 Il 和 Ir 每個 pixel 在 window_size=5 以內的值改成 binarry pattern 分別存成兩個 h*w*75 的 array 裡使得算 cost 可以直接 access 這個 array 來算 Hamming distance。接著再把邊邊兩排 array 用第三排來替換。

```
# >>> Cost Computation
    # TODO: Compute matching cost
    # [Tips] Census cost = Local binary pattern -> Hamming distance
    # [Tips] Set costs of out-of-bound pixels = cost of closest valid pixel
    # [Tips] Compute cost both "Il to Ir" and "Ir to Il" for later left-right
    f= 5
    h = int((f-1)/2)
    filter = np.ones((f,f,3))
    filter[h,h,0] = filter[h,h,1] = filter[h,h,2] = 0

    bp1 = np.zeros((Il.shape[0], Il.shape[1], f*f*3))
    bp2 = np.zeros((Il.shape[0], Il.shape[1],f*f*3))
    for i in tqdm.tqdm(range(h,Il.shape[0]-h)):
        for j in range(h,Il.shape[1]-h):
            bp1[i,j,:] = (((Il[i-h:i+h+1,j-h:j+h+1,:]<=Il[i,j,:])*1)*filter).ravel()
            bp2[i,j,:] = (((Ir[i-h:i+h+1,j-h:j+h+1,:]<=Ir[i,j,:])*1)*filter).ravel()
    costL = np.zeros((Il.shape[0], Il.shape[1], max_disp))
    for i in  tqdm.tqdm(range(h,Il.shape[0]-h)):
        for j in range(h,Il.shape[1]-h):
            for k in range(max_disp):
                if k < j-1:
                    costL[i,j,k] = np.sum(bp1[i,j,:]!=bp2[i,j-k,:])
                else:
                    costL[i,j,k:] = costL[i,j,k-1]
                    break

    for tmp in range(h):
        costL[:,tmp,:] = costL[:,h,:]

        costL[:,-(tmp+1),:] = costL[:,-(h+1),:]
    for tmp in range(h):
        costL[tmp,:,:] = costL[h,:,:]

        costL[-(tmp+1),:,:] = costL[-(h+1),:,:]

    costR = np.zeros((Il.shape[0], Il.shape[1], max_disp))
    for i in  tqdm.tqdm(range(h,Il.shape[0]-h)):
        for j in range(h,Il.shape[1]-h):
            for k in range(max_disp):
                if k < (Il.shape[1]-j-1):
                    costR[i,j,k] = np.sum(bp1[i,j+k,:]!=bp2[i,j,:])
                else:
```

```
                costR[i,j,k:] = costR[i,j,k-1]
                break
    for tmp in range(h):
        costR[:,tmp,:] = costR[:,h,:]

        costR[:,-(tmp+1),:] = costR[:,-(h+1),:]
    for tmp in range(h):
        costR[tmp,:,:] = costR[h,:,:]

        costR[-(tmp+1),:,:] = costR[-(h+1),:,:]
```

2.

再來就是兩個 cost array 逐個 channel 過一個 JBF 用 winner-take-all 後的結果當 joint

```
# >>> Cost Aggregation
    # TODO: Refine the cost according to nearby costs
    # [Tips] Joint bilateral filter (for the cost of each disparty)
    tmp = np.uint8(np.argmin(costL, axis=2))
    for k in range(costL.shape[2]):
        costL[:,:,k] = xip.jointBilateralFilter(tmp,np.uint8(costL[:,:,k]),27,40,5)

    tmp = np.uint8(np.argmin(costR, axis=2))
    for k in range(costR.shape[2]):
        costR[:,:,k] = xip.jointBilateralFilter(tmp,np.uint8(costR[:,:,k]),27,40,5)
```

3.

然後 Winner-take-all

```
# >>> Disparity Optimization
    # TODO: Determine disparity based on estimated cost.
    # [Tips] Winner-take-all
    labelsL = np.argmin(costL, axis=2)
    labelsR = np.argmin(costR, axis=2)
```

4.

這裡我就用 Left-right consistency check 再來用 hole filling 以及 Weighted median filtering 用原圖當 joint, r=25

```
# >>> Disparity Refinement
    # TODO: Do whatever to enhance the disparity map
    # [Tips] Left-right consistency check -> Hole filling -> Weighted median filtering
    for i in range(labelsL.shape[0]):
        for j in range(labelsL.shape[1]):
            if labelsL[i,j] != labelsR[i,j-labelsL[i,j]]:
                labelsL[i,j] = -1
```

```
FL = np.zeros(labelsL.shape)
for i in range(FL.shape[0]):
    for j in range(FL.shape[1]):
        if labelsL[i,j] == -1:
            if j == 0:
                FL[i,j] = max_disp
            else:
                FL[i,j] = labelsL[i,j-1]
        else:
            FL[i,j] = labelsL[i,j]

FR = np.zeros(labelsL.shape)
for i in range(FL.shape[0]):
    for j in range(FL.shape[1]):
        if labelsL[i,FL.shape[1]-1-j] == -1:
            if j == 0:
                FR[i,FL.shape[1]-1-j] = max_disp
            else:
                FR[i,FL.shape[1]-1-j] = labelsL[i,FL.shape[1]-j]
        else:
            FR[i,FL.shape[1]-1-j] = labelsL[i,FL.shape[1]-1-j]
labels = np.minimum(FL,FR)
labels = xip.weightedMedianFilter(np.uint8(Il), np.uint8(labels),25)
```