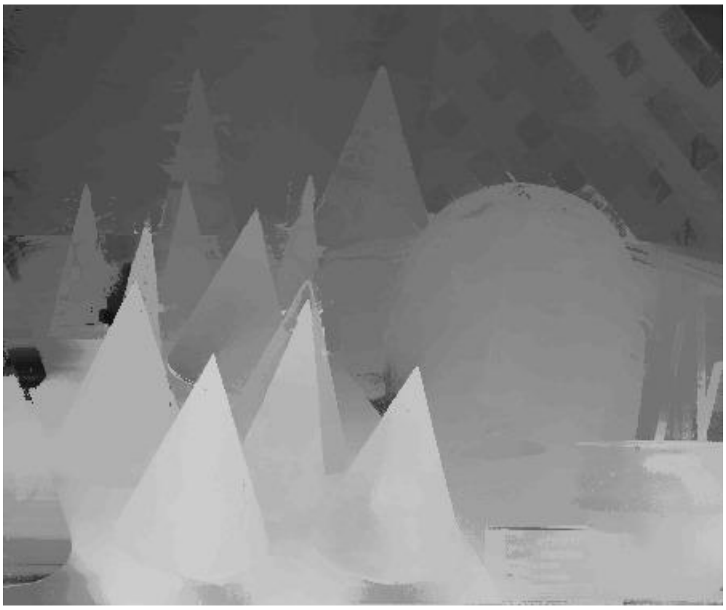
電腦視覺-HW4 Report

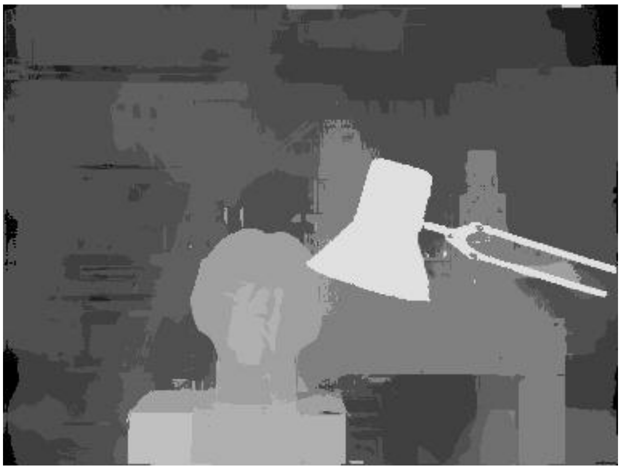
R08921053 電機丙研二 梁峻瑋

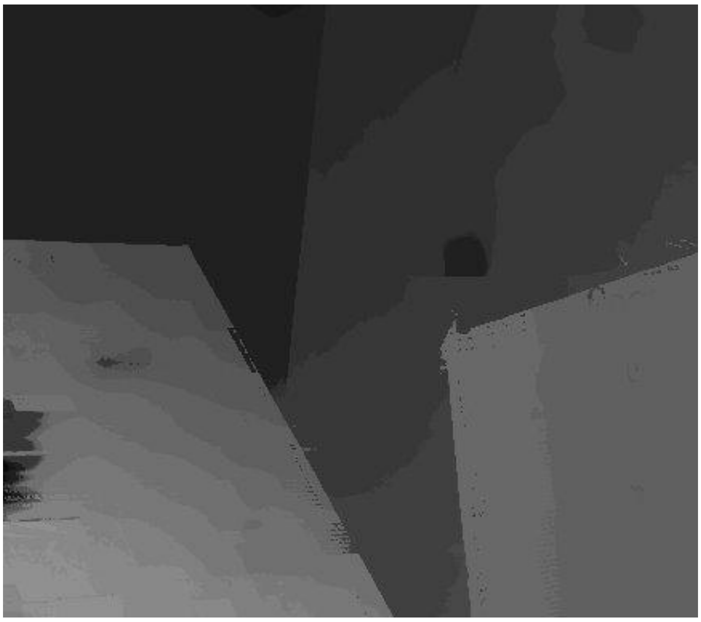
#Part

Visualize the disparity map for all 4 testing images

<table>
<tr><td>Cones</td><td>runtime = 3m13.402s</td></tr>
<tr><td colspan="2"></td></tr>
</table>

<table>
<tr><td>Teddy</td><td>runtime = 4m39.097s</td></tr>
<tr><td colspan="2"></td></tr>
</table>

Tsukuba	runtime = 0m48.979s
	

Venus	runtime = 1m42.568s
	

#Part2:

Report the bad pixel ratio for 2 testing images with given gt.

Teddy	Bad pixel ratio: 14.07%
Tsukuba	Bad pixel ratio: 4.12%

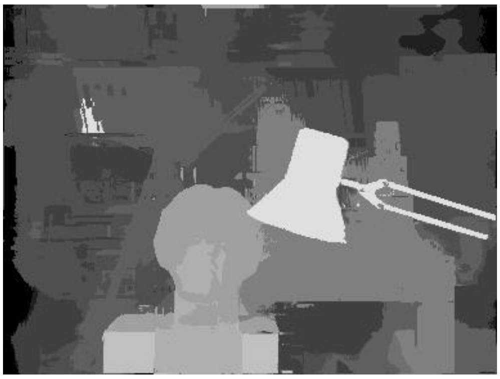
#Part3: Explain your algorithm in terms of the standard 4-step pipeline.

Step1: (含 Step2)

首先，我試著用 Census cost 來取代原本使用的 L1-norm cost，結果因為無法在十分鐘限制內跑完，因此不能走這條路，如下圖所示意:

#Runtime bottleneck

Teddy, Runtime: 13m2.336s	
Census cost Bad pixel ratio: 9.98%	L1-norm cost Bad pixel ratio: 14.07%
	

Tsukuba, Runtime: 2m18.669s	
Census cost Bad pixel ratio: 4.53%	L1-norm cost Bad pixel ratio: 4.12%
	

其次，為了解決上面的 runtime bottleneck，我試著優化 runtime．由於 step1 部分是整個程式中，花費>80%時間的區塊．因此，我針對不同的 for-loop 順序去測試 runtime，結果幫助甚微，如下圖的結果:

#Cache Experiment: for-loop order and run-time relationship

order	run time of Tsukuba
s->y->x	2m7.902s
s->x->y	2m0.549s
y->s->x	2m7.699s
y->x->s	2m36.278s
x->s->y	2m1.316s
x->y->s	2m1.122s

綜合上述討論，最後我只能放棄 Census cost，選擇 L1-norm cost，並且使用三層 for-loop 的基礎實作方法，也順便在迴圈的結尾完成 step2 的「jointBilateralFilter」工作，如同附圖所示意:

```
cost_Il2Ir = np.zeros((max_disp+1, h, w), dtype=np.float32)
cost_Ir2Il = np.zeros((max_disp+1, h, w), dtype=np.float32)

for s in range(max_disp+1):
    for x in range(w):
        xs_lft = max(x-s, 0)
        xs_rig = min(x+s, w-1)
        for y in range(h):
            cost_Il2Ir[s, y, x] = dist(Il[y, x], Ir[y, xs_lft])
            cost_Ir2Il[s, y, x] = dist(Ir[y, x], Il[y, xs_rig])
    cost_Il2Ir[s,] = xip.jointBilateralFilter(Il, cost_Il2Ir[s,], 30, 5, 5)
    cost_Ir2Il[s,] = xip.jointBilateralFilter(Ir, cost_Ir2Il[s,], 30, 5, 5)
```

值得一提的是, jointBilateralFilter 的參數選擇，對於結果的影響非常大，可能從 28%的 bad pixel ratio 降低到 4%的 bad pixel ratio!!

Step3:

對 disparity 取 argmin，只需要用下列兩行就可完成:

```
winner_dispL = np.argmin(cost_Il2Ir, axis=0)
winner_dispR = np.argmin(cost_Ir2Il, axis=0)
```

step4:

做 Left-right consistency check, Hole filling. 最後可以選擇加一層 Weighted median filtering. 由於這邊佔 runtime 的比例很少，直接用暴力法實作，也不大影響效率. 至於 bad pixel ratio，也不會被實作方式影響，只取決於 Weighted median filtering 的參數，如下圖所示意:

```
for y in range(h):
    for x in range(w):
        if x-winner_dispL[y,x]>=0 and winner_dispL[y,x] == winner_dispR[y,x-winner_dispL[y,x]]:
            continue
        else:
            winner_dispL[y,x]=-1

for y in range(h):
    for x in range(w):
        if winner_dispL[y,x] == -1:
            l = 0
            r = 0
            while x-l>=0 and winner_dispL[y,x-l] == -1:
                l+=1
            if x-l < 0:
                FL = max_disp
            else:
                FL = winner_dispL[y,x-l]

            while x+r<=w-1 and winner_dispL[y,x+r] == -1:
                r+=1
            if x+r > w-1:
                FR = max_disp
            else:
                FR = winner_dispL[y, x+r]
            winner_dispL[y,x] = min(FL, FR)

labels = xip.weightedMedianFilter(Il.astype(np.uint8), winner_dispL.astype(np.uint8), 18, 1)
```