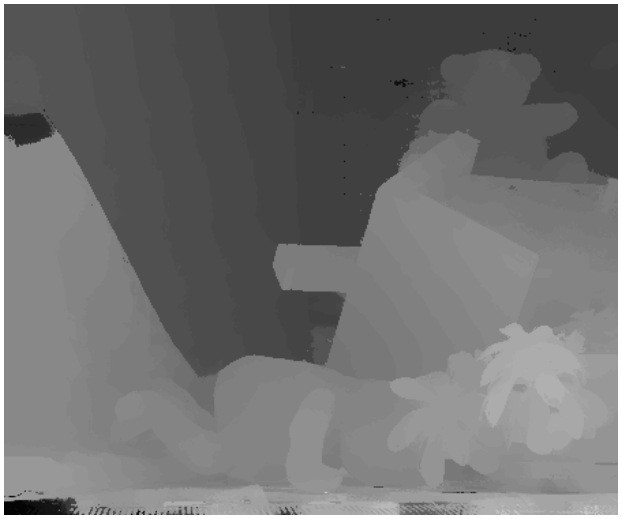
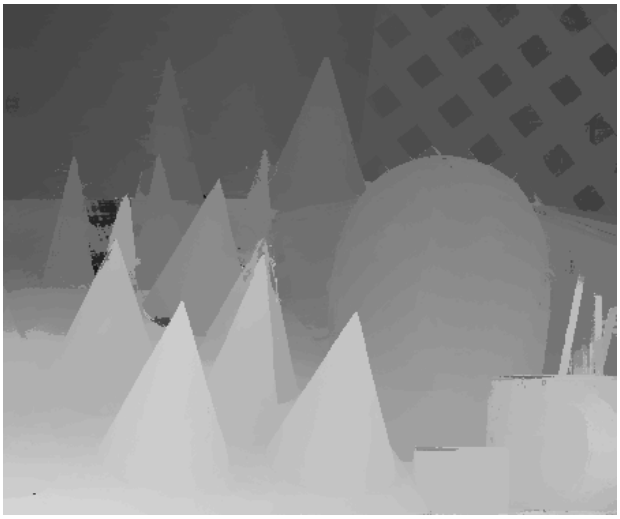
Homework #4

June 10, 2021

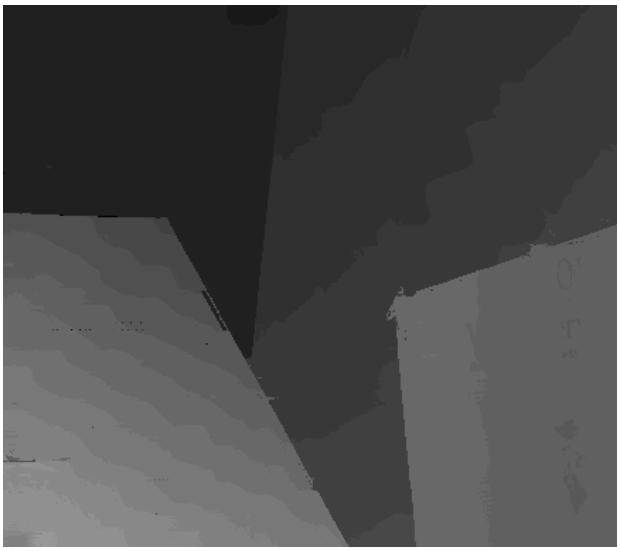
Disparity map



(a) Teddy



(b) Cones



(c) Venus

(d) Tsukuba

Bad pixel ratio

Teddy: 9.49%

Tsukuba: 3.18%

Algorithm

Cost Computation

使用 Cencus Cost，首先先求出每個 pixel 的 local binary pattern，也就是每個對應的 window 小於該 image pixel 的 boolean value。

可以使用 numpy slicing index 加速，直接將對應的 window 切出來計算。

Listing 1: computeDisp.py (line 28-33)

```
28      pattern_l = np.zeros((h, w, window_size * window_size, 1), dtype=np.bool)
29      pattern_r = np.zeros((h, w, window_size * window_size, 1), dtype=np.bool)
30      for i in range(h):
31          for j in range(w):
32              pattern_l[i, j] = (Il_pading[i: i+window_size, j: j+window_size] <
                ↪  Il_pading[i+padding_size, j+padding_size]).reshape(-1,1)
33              pattern_r[i, j] = (Ir_pading[i: i+window_size, j: j+window_size] <
                ↪  Ir_pading[i+padding_size, j+padding_size]).reshape(-1,1)
```

接者計算每個 pixel 對應到每個 disparity 的 hamming distance，也就是兩個 local binary pattern 的 XOR 的總和，可以求得左圖對右圖以及右圖對左圖每個 disparcity 的 cost。

這裡同樣可以使用 slicing index 加速，可以發現計算右圖對左圖的 cost 時，shift 只會到該 pixel 的 y，而計算左圖對右圖時，shift 只會到 w-y。

Listing 2: computeDisp.py (line 35-40)

```
35      cost_l = np.ones((h, w, max_disp+1), dtype=np.float32) * (window_size ** 2)
36      cost_r = np.ones((h, w, max_disp+1), dtype=np.float32) * (window_size ** 2)
37      for i in range(h):
38          for j in range(w):
39              cost_l[i, j, :j+1] = np.logical_xor(pattern_l[i, j], pattern_r[i,
                ↪  j::-1])[:max_disp+1].sum(-1).sum(-1)
40              cost_r[i, j, :w-j] = np.logical_xor(pattern_r[i, j], pattern_l[i,
                ↪  j:w:])[:max_disp+1].sum(-1).sum(-1)
```

Cost Aggregation

使用 joint bilateral filter 使用灰階圖當 guidance image，讓 cencus cost 更平滑。

Listing 3: computeDisp.py (line 48-50)

```
48      for shift in range(max_disp+1):
49          cost_l[:, :, shift] = xip.jointBilateralFilter(Il_gray, cost_l[:, :,
            ↪  shift], 9, 0.9, 9)
50          cost_r[:, :, shift] = xip.jointBilateralFilter(Ir_gray, cost_r[:, :,
            ↪  shift], 9, 0.9, 9)
```

Disparity Optimization

使用 Winner-take-all，每個 pixel 選擇最小的 disparity。

Listing 4: computeDisp.py (line 58-59)

```
58      labels_l = np.argmin(cost_l, axis=-1)
59      labels_r = np.argmin(cost_r, axis=-1)
```

Disparity Refinement

Step 1. Left-right consistency check

計算每個 pixel 的左右一致性，也就是檢查 $D_L(x,y) == D_R(x - D_L(x,y), y)$。

Listing 5: computeDisp.py (line 67-71)

```
67      check = np.zeros((h, w))
68      for i in range(h):
69          for j in range(w):
70              if labels_r[i, j-labels_l[i, j]] == labels_l[i, j]:
71                  check[i, j] = 1
```

Step 2. Hole filling

如果該 pixel 的 Left-right 不一致，則找出對應的 F_L、F_R，也就是往左 (右) 找第一個有效的 disparity，取小的當作該 pixel 的 disparity。

Listing 6: computeDisp.py (line 73-84)

```
73      for i in range(h):
74          for j in range(w):
75              if check[i, j] == 0:
76                  labels_l[i, j] = max_disp
77                  for k in range(j-1, -1, -1):
78                      if check[i, k] != 0:
79                          labels_l[i, j] = min(labels_l[i, j], labels_l[i, k])
80                          break
81                  for k in range(j+1, w):
82                      if check[i, k] != 0:
83                          labels_l[i, j] = min(labels_l[i, j], labels_l[i, k])
84                          break
```

Step 3. Weighted median filtering

將原圖當作 guidance image，計算 Weighted median filtering。

Listing 7: computeDisp.py (line 86)

```
86      xip.weightedMedianFilter(joint=Il.astype(np.uint8),
        ↪  src=labels_l.astype(np.float32), dst=labels, r=11, sigma=22.5)
```