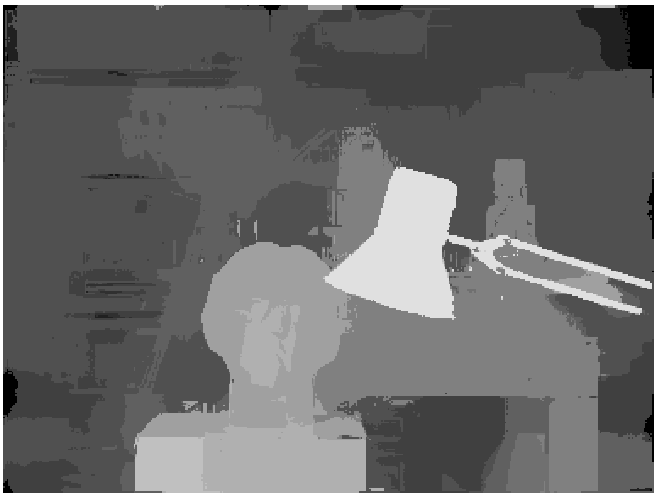
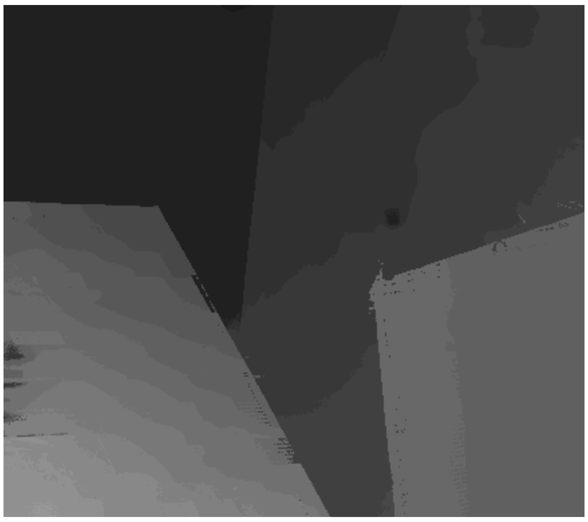
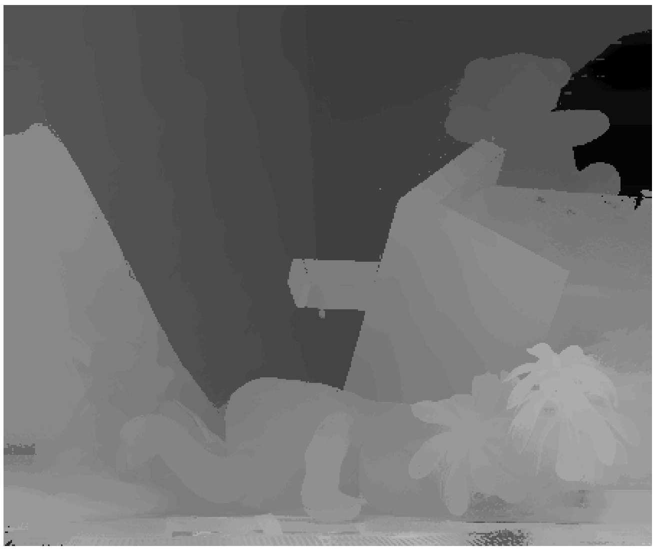
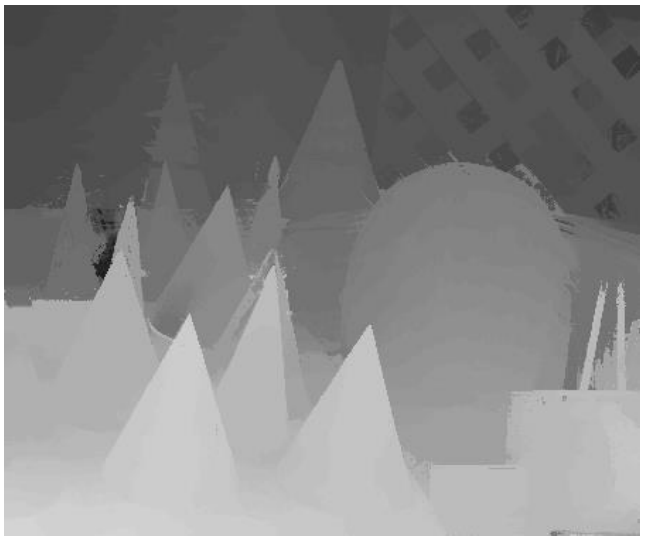
Computer Vision HW4 Report

Student ID: R10522811	
Name: 簡永展	

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	4.11%
Teddy	13.51%

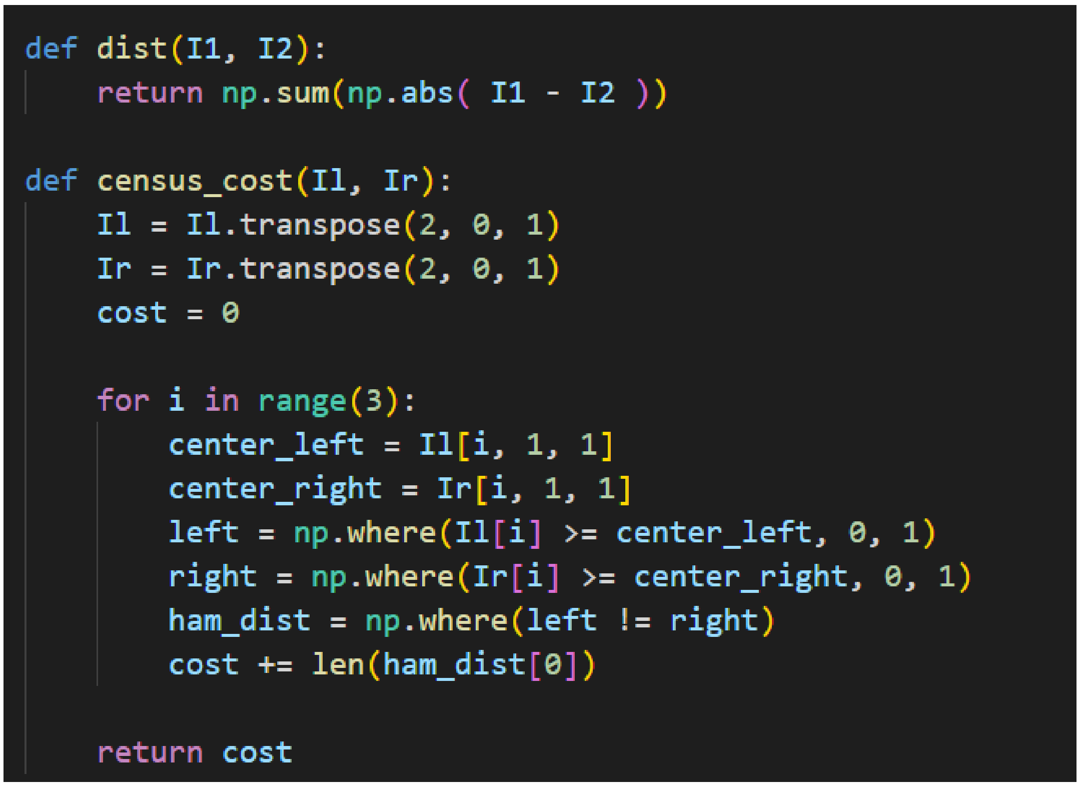
Describe your algorithm in terms of 4-step pipeline.

First, I use absolute difference to compute my cost. Second, I use the built-in module joint bilateral filter to smooth my raw costs. Then, I use winner-take-all to optimize my disparity. Last, I divided disparity refinement into three parts, left-right consistency check, hole filling and weighted median filtering separately.

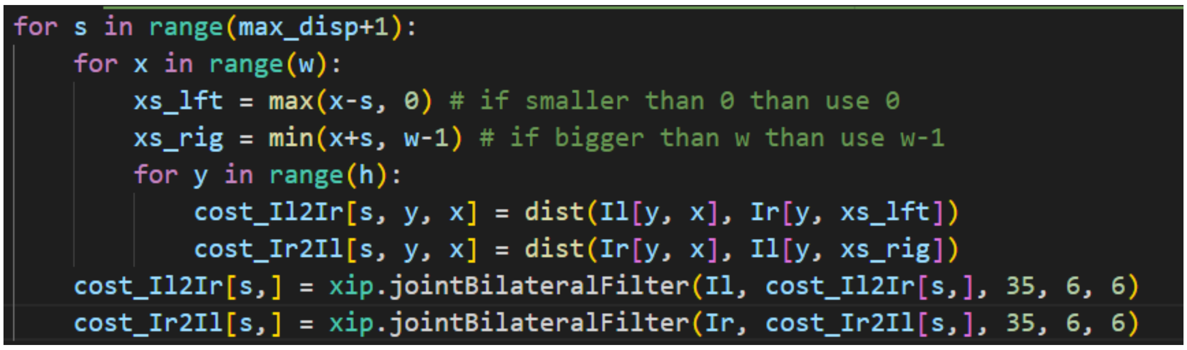
Besides, I have tried census cost; however, it takes over 10 minutes. Therefore, I choose absolute difference to compute the cost.

Below is my code:

Compute cost

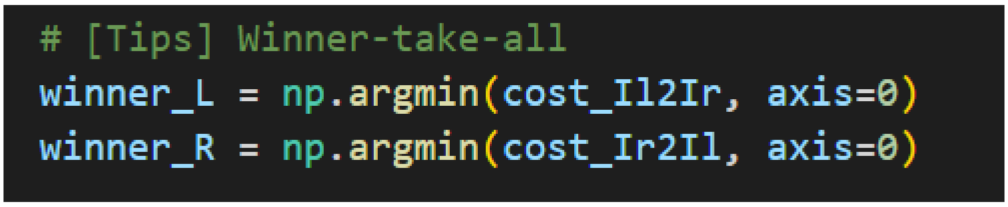


```python
def dist(I1, I2):
    return np.sum(np.abs( I1 - I2 ))

def census_cost(Il, Ir):
    Il = Il.transpose(2, 0, 1)
    Ir = Ir.transpose(2, 0, 1)
    cost = 0

    for i in range(3):
        center_left = Il[i, 1, 1]
        center_right = Ir[i, 1, 1]
        left = np.where(Il[i] >= center_left, 0, 1)
        right = np.where(Ir[i] >= center_right, 0, 1)
        ham_dist = np.where(left != right)
        cost += len(ham_dist[0])

    return cost
```



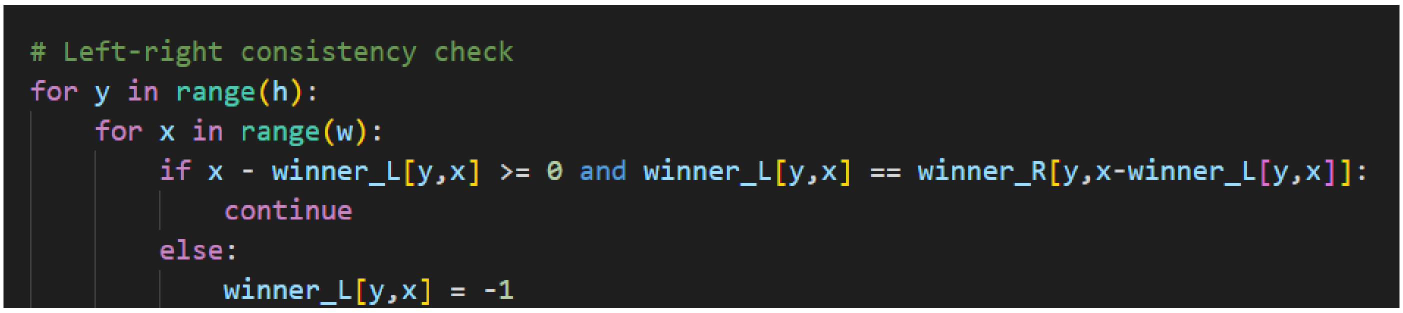
```python
for s in range(max_disp+1):
    for x in range(w):
        xs_lft = max(x-s, 0) # if smaller than 0 than use 0
        xs_rig = min(x+s, w-1) # if bigger than w than use w-1
        for y in range(h):
            cost_Il2Ir[s, y, x] = dist(Il[y, x], Ir[y, xs_lft])
            cost_Ir2Il[s, y, x] = dist(Ir[y, x], Il[y, xs_rig])
    cost_Il2Ir[s,] = xip.jointBilateralFilter(Il, cost_Il2Ir[s,], 35, 6, 6)
    cost_Ir2Il[s,] = xip.jointBilateralFilter(Ir, cost_Ir2Il[s,], 35, 6, 6)
```

Winner-take-all



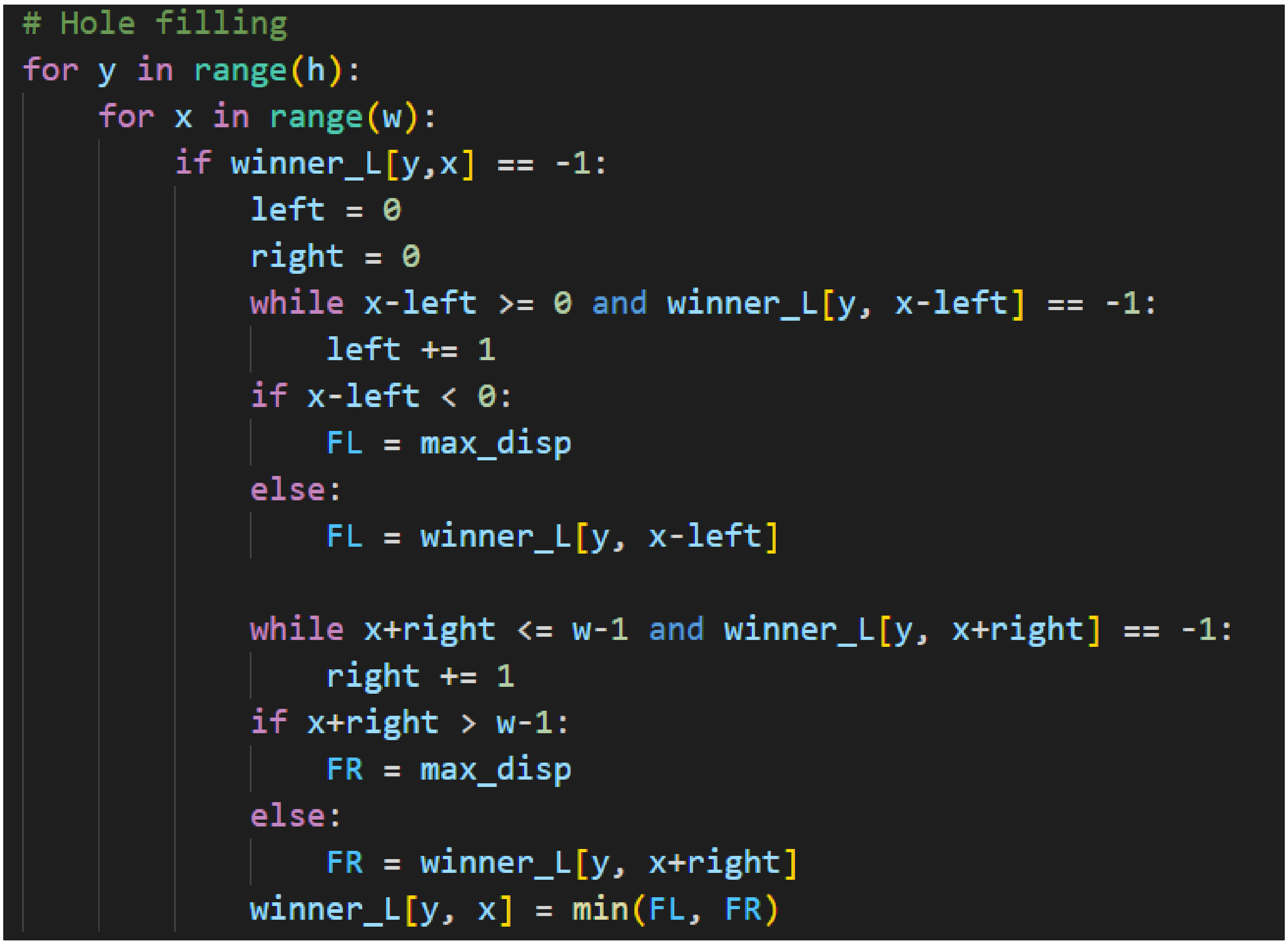
```python
# [Tips] Winner-take-all
winner_L = np.argmin(cost_Il2Ir, axis=0)
winner_R = np.argmin(cost_Ir2Il, axis=0)
```

Left-right consistency check



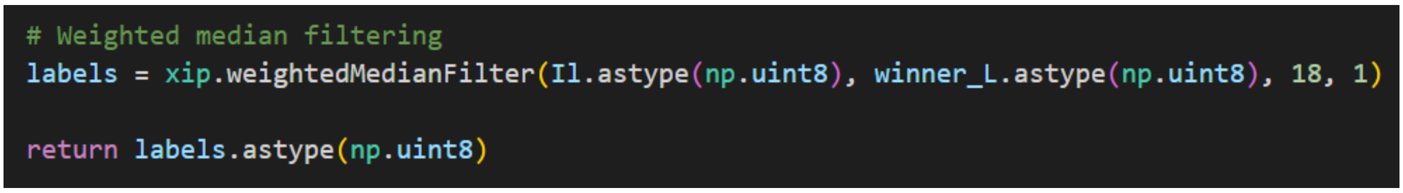
```python
# Left-right consistency check
for y in range(h):
    for x in range(w):
        if x - winner_L[y,x] >= 0 and winner_L[y,x] == winner_R[y,x-winner_L[y,x]]:
            continue
        else:
            winner_L[y,x] = -1
```

Hole filling



```python
# Hole filling
for y in range(h):
    for x in range(w):
        if winner_L[y,x] == -1:
            left = 0
            right = 0
            while x-left >= 0 and winner_L[y, x-left] == -1:
                left += 1
            if x-left < 0:
                FL = max_disp
            else:
                FL = winner_L[y, x-left]

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

Weighted median filtering



```python
# Weighted median filtering
labels = xip.weightedMedianFilter(Il.astype(np.uint8), winner_L.astype(np.uint8), 18, 1)

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