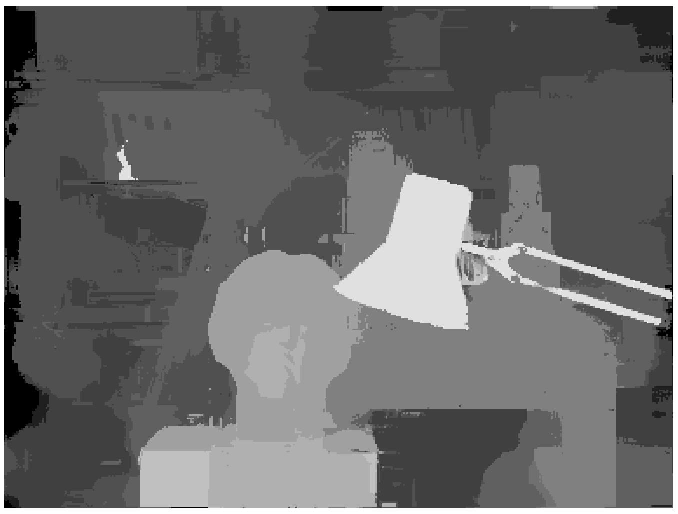
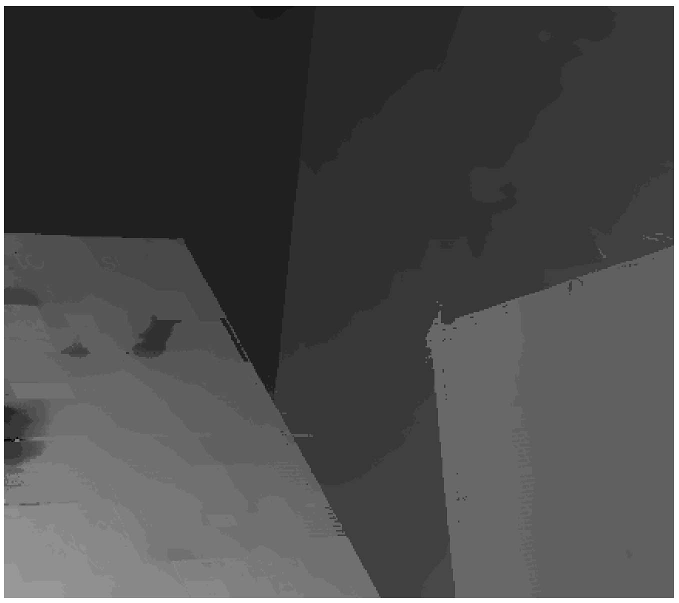
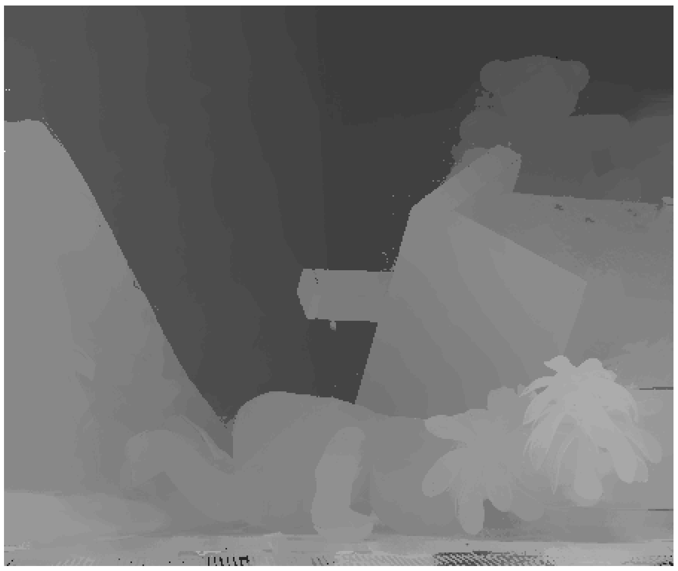
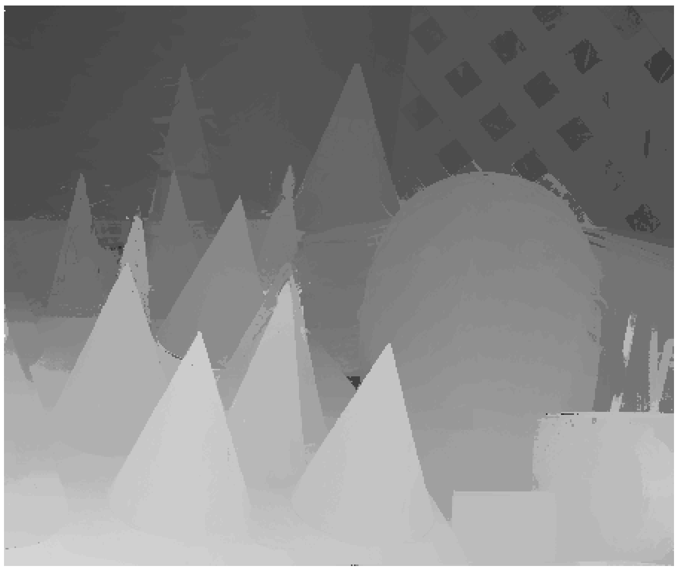
Computer Vision HW4 Report

Student ID: R10521802
Name: 李睿莆

Visualize the disparity map of 4 testing images.

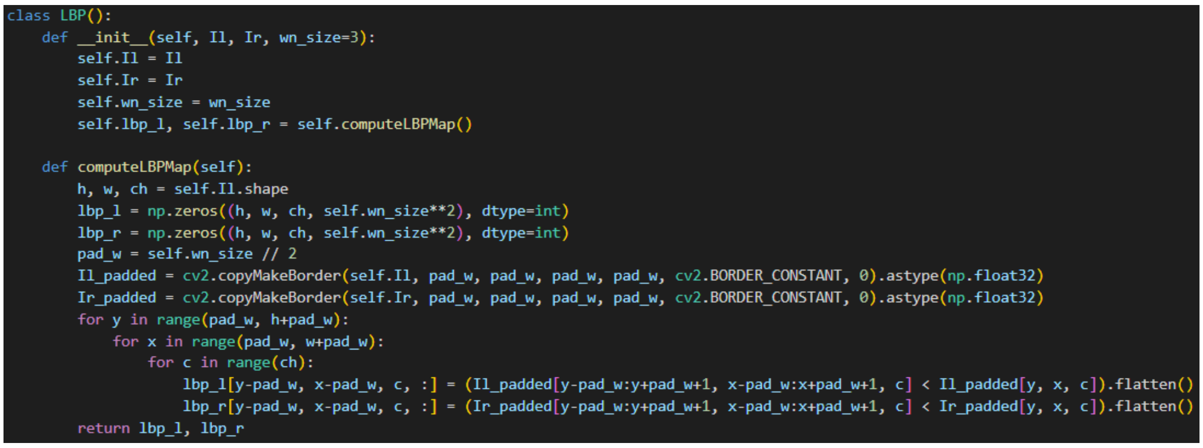
Tsukuba	Venus
	
Teddy	**Cones**
	

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

	bad pixel ratio
Tsukuba	3.94%
Teddy	10.97%

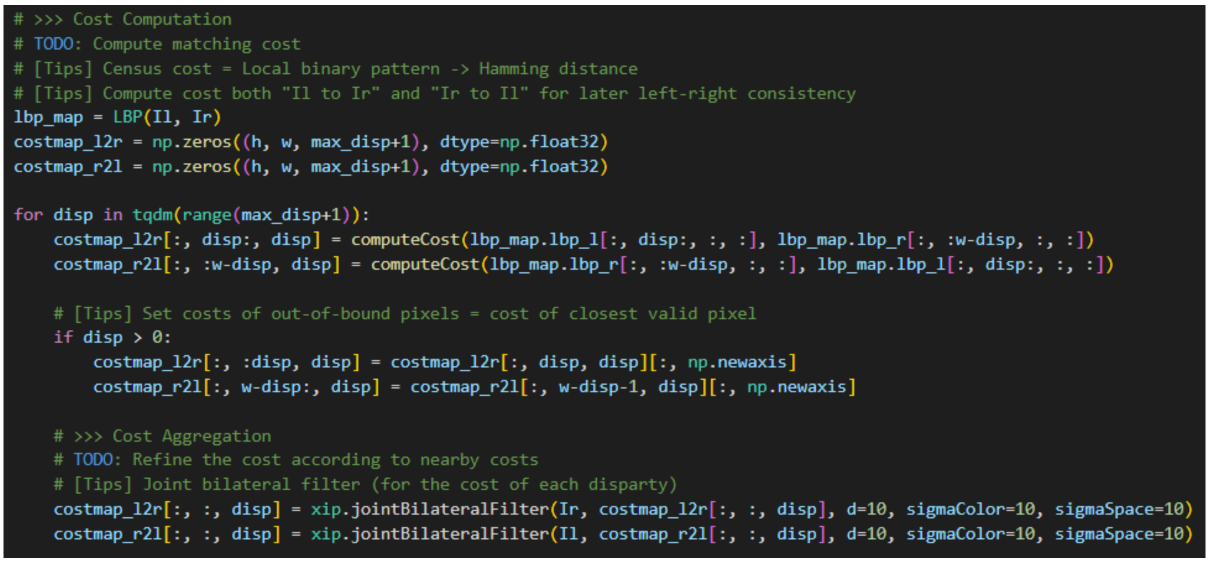
Describe your algorithm in terms of 4-step pipeline.

Step1: Cost Computation

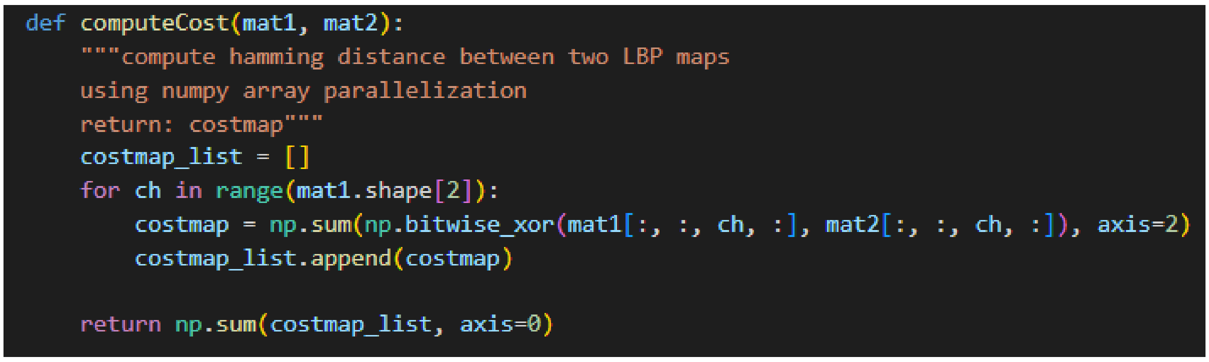


```python
class LBP():
    def __init__(self, Il, Ir, wn_size=3):
        self.Il = Il
        self.Ir = Ir
        self.wn_size = wn_size
        self.lbp_l, self.lbp_r = self.computeLBPMap()

    def computeLBPMap(self):
        h, w, ch = self.Il.shape
        lbp_l = np.zeros((h, w, ch, self.wn_size**2), dtype=int)
        lbp_r = np.zeros((h, w, ch, self.wn_size**2), dtype=int)
        pad_w = self.wn_size // 2
        Il_padded = cv2.copyMakeBorder(self.Il, pad_w, pad_w, pad_w, pad_w, cv2.BORDER_CONSTANT, 0).astype(np.float32)
        Ir_padded = cv2.copyMakeBorder(self.Ir, pad_w, pad_w, pad_w, pad_w, cv2.BORDER_CONSTANT, 0).astype(np.float32)
        for y in range(pad_w, h+pad_w):
            for x in range(pad_w, w+pad_w):
                for c in range(ch):
                    lbp_l[y-pad_w, x-pad_w, c, :] = (Il_padded[y-pad_w:y+pad_w+1, x-pad_w:x+pad_w+1, c] < Il_padded[y, x, c]).flatten()
                    lbp_r[y-pad_w, x-pad_w, c, :] = (Ir_padded[y-pad_w:y+pad_w+1, x-pad_w:x+pad_w+1, c] < Ir_padded[y, x, c]).flatten()
        return lbp_l, lbp_r
```

首先以 3x3 window size，以及 3 個迴圈計算出左像與右像的 Local Binary Pattern Map，存到 LBP()物件，超出 window 的區域補 0。



```python
# >>> Cost Computation
# TODO: Compute matching cost
# [Tips] Census cost = Local binary pattern -> Hamming distance
# [Tips] Compute cost both "Il to Ir" and "Ir to Il" for later left-right consistency
lbp_map = LBP(Il, Ir)
costmap_l2r = np.zeros((h, w, max_disp+1), dtype=np.float32)
costmap_r2l = np.zeros((h, w, max_disp+1), dtype=np.float32)

for disp in tqdm(range(max_disp+1)):
    costmap_l2r[:, disp:, disp] = computeCost(lbp_map.lbp_l[:, disp:, :, :], lbp_map.lbp_r[:, :w-disp, :, :])
    costmap_r2l[:, :w-disp, disp] = computeCost(lbp_map.lbp_r[:, :w-disp, :, :], lbp_map.lbp_l[:, disp:, :, :])

    # [Tips] Set costs of out-of-bound pixels = cost of closest valid pixel
    if disp > 0:
        costmap_l2r[:, :disp, disp] = costmap_l2r[:, disp, disp][:, np.newaxis]
        costmap_r2l[:, w-disp:, disp] = costmap_r2l[:, w-disp-1, disp][:, np.newaxis]

    # >>> Cost Aggregation
    # TODO: Refine the cost according to nearby costs
    # [Tips] Joint bilateral filter (for the cost of each disparty)
    costmap_l2r[:, :, disp] = xip.jointBilateralFilter(Ir, costmap_l2r[:, :, disp], d=10, sigmaColor=10, sigmaSpace=10)
    costmap_r2l[:, :, disp] = xip.jointBilateralFilter(Il, costmap_r2l[:, :, disp], d=10, sigmaColor=10, sigmaSpace=10)
```



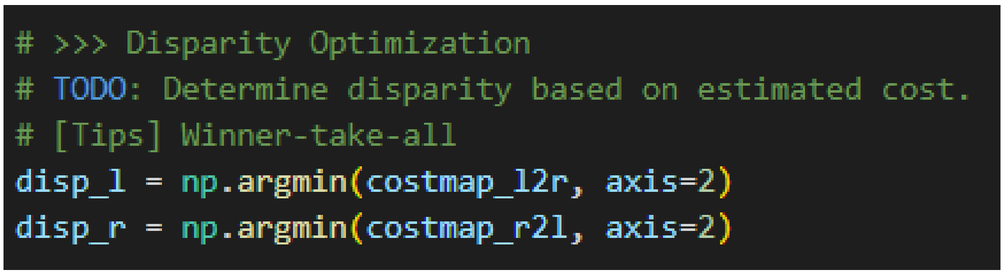
```python
def computeCost(mat1, mat2):
    """compute hamming distance between two LBP maps
    using numpy array parallelization
    return: costmap"""
    costmap_list = []
    for ch in range(mat1.shape[2]):
        costmap = np.sum(np.bitwise_xor(mat1[:, :, ch, :], mat2[:, :, ch, :]), axis=2)
        costmap_list.append(costmap)

    return np.sum(costmap_list, axis=0)
```

接著再以一個長度為 max_disp 的迴圈，計算 ImgLeft to ImgRight 與 ImgRight to ImgLeft 在每個視差下的 census cost map。原本採用的方法是以 3 個迴圈(y, x, ch)，根據各 pixel LBP 逐點計算出 cost 再拼回去 cost map，但速度有點慢(Teddy=6m…)，因此最後透過平行運算，分別對三個 channel 的 LBP map 取 Hamming distance，再加總每個 channel 的 cost map 拼回去，速度顯著提升(Teddy=15s)。前述計算 cost map 時會因視差大小使得某些 pixel 沒有對應而不能計算 cost，因此先計算有對應的 cost，再將沒有 cost 的 pixel 用最接近的 cost 補上。

Step2: Cost Aggregation

透過 xip.joinBilateralFilter()，以原圖當 guidance，針對每個視差下的 costmap 做濾波處理，以提升最後 disparity map 的精度。

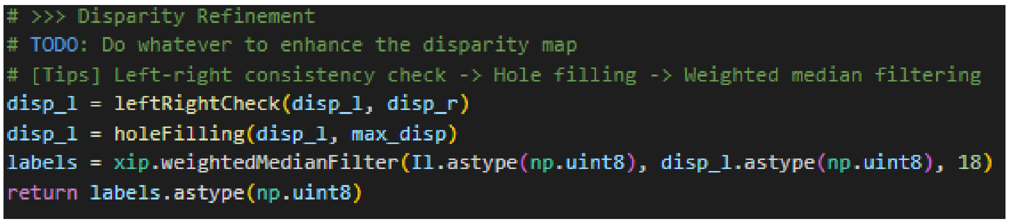
Step3: Disparity Optimization



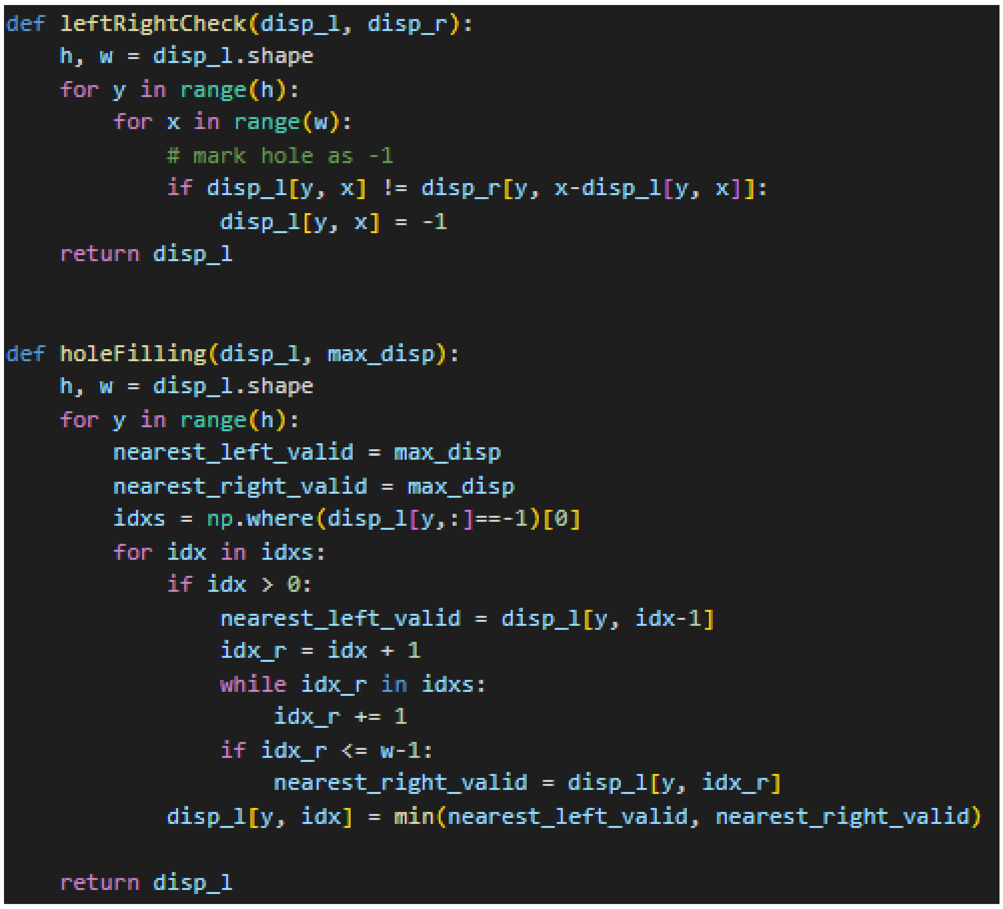
```python
# >>> Disparity Optimization
# TODO: Determine disparity based on estimated cost.
# [Tips] Winner-take-all
disp_l = np.argmin(costmap_l2r, axis=2)
disp_r = np.argmin(costmap_r2l, axis=2)
```

以 np.argmin()對找出 costmap 中 cost 最小的 index 即為視差。

Step4: Disparity Refinement



```python
# >>> Disparity Refinement
# TODO: Do whatever to enhance the disparity map
# [Tips] Left-right consistency check -> Hole filling -> Weighted median filtering
disp_l = leftRightCheck(disp_l, disp_r)
disp_l = holeFilling(disp_l, max_disp)
labels = xip.weightedMedianFilter(Il.astype(np.uint8), disp_l.astype(np.uint8), 18)
return labels.astype(np.uint8)
```



```python
def leftRightCheck(disp_l, disp_r):
    h, w = disp_l.shape
    for y in range(h):
        for x in range(w):
            # mark hole as -1
            if disp_l[y, x] != disp_r[y, x-disp_l[y, x]]:
                disp_l[y, x] = -1
    return disp_l


def holeFilling(disp_l, max_disp):
    h, w = disp_l.shape
    for y in range(h):
        nearest_left_valid = max_disp
        nearest_right_valid = max_disp
        idxs = np.where(disp_l[y,:]==-1)[0]
        for idx in idxs:
            if idx > 0:
                nearest_left_valid = disp_l[y, idx-1]
                idx_r = idx + 1
                while idx_r in idxs:
                    idx_r += 1
                if idx_r <= w-1:
                    nearest_right_valid = disp_l[y, idx_r]
            disp_l[y, idx] = min(nearest_left_valid, nearest_right_valid)

    return disp_l
```

Left-right consistency check 以兩個迴圈逐點確認，不通過的 pixel 標示為-1。Holl-filling 也是透過兩個迴圈，找出最適合的 value 填補。最後以 xip.weightedMedianFilter()對填補完的 disparity map 進行濾波處理，進一步減少 bad pixel ratio。