计算机视觉-HW4

xTryer

2024/11/21

1 Basic Stereo Matching Algorithm

1.1 Disparity Map Computation

思路：先对两张图片根据 window 的大小进行 padding(mode=edge), 防止边缘像素出现值溢出的情况。然后开始遍历扫描每一行，对 ref_img 对应 row 上每个像素的 window 在 disparity_range 范围内与 sec_img 对应 row 上的每个像素 (如果存在) 进行匹配, 计算 cost，找出对于 ref_img 正在计算的像素的最佳 disparity。

对于 matching_function，SSD 利用 L2 距离，SAD 利用 abs 距离，normalized_correlation 利用类似“相关系数”的值。

计算 time_cost 使用 python 的 time 库，在函数开头和结尾打点计时，输出时间差。

```python
def task1_compute_disparity_map_simple(
        ref_img: np.ndarray,    # shape (H, W)
        sec_img: np.ndarray,    # shape (H, W)
        window_size: int,
        disparity_range: Tuple[int, int], # (min_disparity, max_disparity)
        matching_function: str  # can be 'SSD', 'SAD', 'normalized_correlation'
    ):
    task1_time_start = time.time()
    img_H,img_W=np.shape(ref_img)
    min_disparity,max_disparity = disparity_range

    # pad the img to scan
    padding_ = window_size//2
    pad_ref_img =
        np.pad(ref_img,((padding_,padding_),(padding_,padding_)),'edge').astype(np.float64)
    pad_sec_img =
        np.pad(sec_img,((padding_,padding_),(padding_,padding_)),'edge').astype(np.float64)

    disparity_map = np.zeros(np.shape(ref_img),dtype=np.float32)
    for row_ in range(padding_,padding_+img_H):
        # scan each row's all piexls
```

```
        for col_ in range(padding_,padding_+img_W):
            # generate a window
            pixel_window_ref =
                pad_ref_img[row_-padding_:row_+padding_,col_-padding_:col_+padding_]
            cur_disparity=min_disparity
            cur_match_result=0

            for d_ in range(min_disparity,max_disparity+1):
                if col_-d_<padding_ or col_-d_>img_W:
                    continue
                # get window of sec_img
                pixel_window_sec =
                    pad_sec_img[row_-padding_:row_+padding_,col_-d_-padding_:col_-d_+padding_]

                # print(pixel_window_ref)
                # print(pixel_window_sec)
                # print("#")

                tmp_match_result = 0
                if matching_function=='SSD':
                    tmp_window_distance = pixel_window_ref-pixel_window_sec
                    tmp_match_result=np.sum((tmp_window_distance**2))
                elif matching_function=='SAD':
                    tmp_window_distance = pixel_window_ref-pixel_window_sec
                    tmp_match_result=np.sum(np.abs(tmp_window_distance))
                elif matching_function=='normalized_correlation':
                    ref_window_mean = np.mean(pixel_window_ref)
                    sec_window_mean = np.mean(pixel_window_sec)

                    ref_window_sub_mean = pixel_window_ref-ref_window_mean
                    sec_window_sub_mean = pixel_window_sec-sec_window_mean

                    tmp_match_result=np.sum(ref_window_sub_mean*sec_window_sub_mean)/(np.sqrt(np.sum(r
                else:
                    raise ValueError("Invalid matching_function")


                if matching_function=='normalized_correlation':
                    if tmp_match_result>cur_match_result or d_==min_disparity:
                        cur_match_result=tmp_match_result
                        cur_disparity=d_
                else:
                    if tmp_match_result<cur_match_result or d_==min_disparity:
```

```
                    cur_match_result=tmp_match_result
                    cur_disparity=d_

            disparity_map[row_-padding_,col_-padding_]=cur_disparity
        # print(disparity_map[row_-padding_])

    task1_time_end=time.time()
    output_str= f"Task1 time
        cost(window_size:{window_size},disparity_range:{disparity_range},matching:function:{matching
    print(output_str)
    with open("log.txt",'a',encoding='utf-8') as write_file:
        write_file.write(output_str+'\n')
    return disparity_map
```

1.2 Hyperparameter Settings and Report

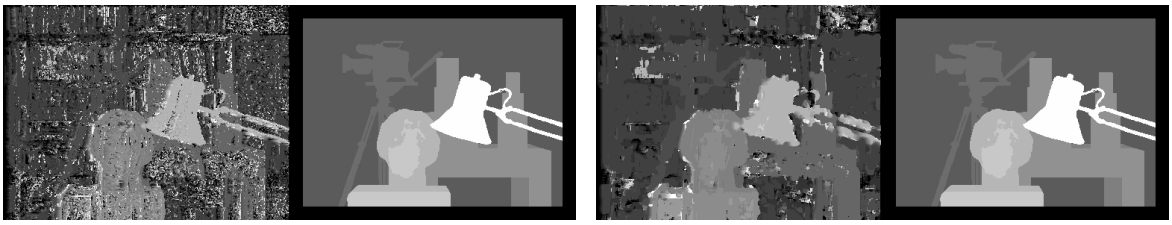
1.2.1 Task1-Q1:How does the running time depend on window size, disparity range, and matching function?

- For window size:windowsize 越大，计算量越大，运行时间略长一些, 但是差距不是特别明显
  - Task1 time cost(window_size 2,disparity_range (0, 10),SSD):4.066013336181641
  - Task1 time cost(window_size 20,disparity_range (0, 10),SSD):4.487921714782715
  - Task1 time cost(window_size 2,disparity_range (0, 10),ZNCC):19.215134382247925
  - Task1 time cost(window_size 11,disparity_range (0, 10),ZNCC):21.040132761001587
- For disparity range:disparity range 范围越大，运行时间越长，基本上成正比
  - Task1 time cost(window_size 11,disparity_range (0, 5),SSD):2.3007616996765137
  - Task1 time cost(window_size 11,disparity_range (0, 10),SSD):4.23575234413147
  - Task1 time cost(window_size 11,disparity_range (0, 20),SSD):7.914202451705933
  - Task1 time cost(window_size 11,disparity_range (0, 5),SAD):2.351048707962036
  - Task1 time cost(window_size 11,disparity_range (0, 10),SAD):4.169891357421875
  - Task1 time cost(window_size 11,disparity_range (0, 20),SAD):7.846660137176514
- For matching function:SAD 与 SSD 运行时间接近，normalized_correlation 运行时间明显长于前两者
  - Task1 time cost(window_size 13,disparity_range (0, 20),SSD):7.940337657928467
  - Task1 time cost(window_size 13,disparity_range (0, 20),SAD):8.02975344657898
  - Task1 time cost(window_size 13,disparity_range (0, 20),ZNCC):38.11306571960449

1.2.2 Which window size works the best for different matching functions?

对于 disparity_range 为 (0,20) 时

对于 SSD: 可以看出在 window size 较小时噪点较多但能保持较多细节，window size 较大时平滑度高，但丢失很多细节，同时物体轮廓出现明显扭曲, 综合考虑大概在 window size 为 13 时取得较好效果



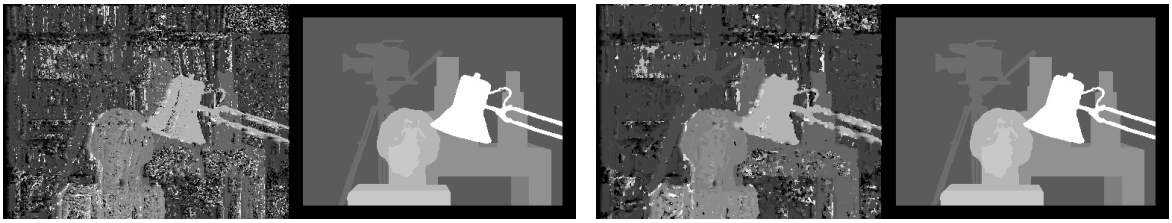
(a) window:2 range:(0,20) SSD (b) window:7 range:(0,20) SSD

图 1: Task 1-Q2:SSD

(a) window:13 range:(0,20) SSD (b) window:20 range:(0,20) SSD

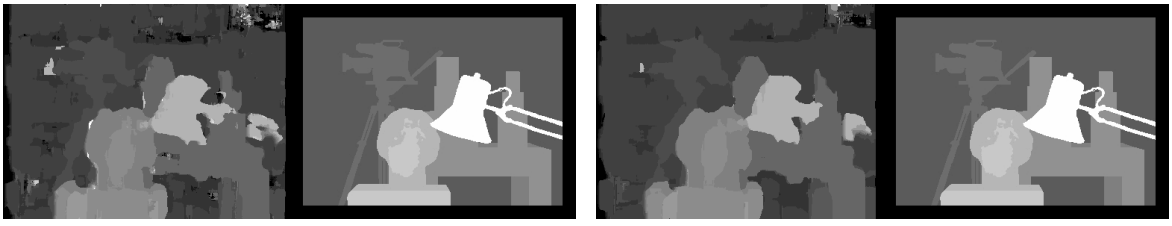
图 2: Task 1-Q2:SSD

对于 SAD：可以看出在 window size 较小时能保持一些细节但噪点较多，与 SSD 的情况类似 window size 调大后平滑度高，但丢失很多细节，综合考虑再 window size 为 13 时取得较好效果



(a) window:2 range:(0,20) SAD (b) window:5 range:(0,20) SAD

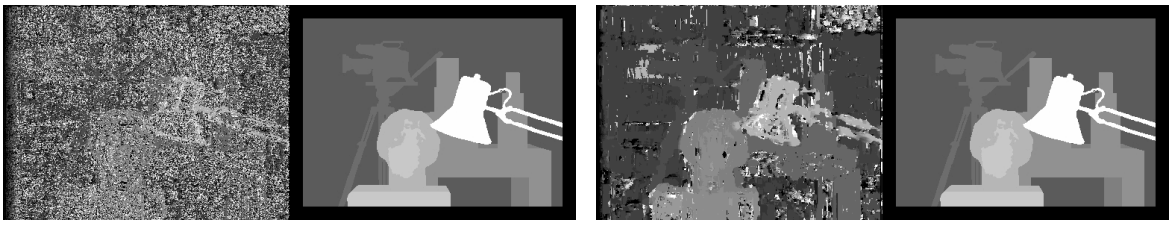
图 3: Task 1-Q2:SAD



(a) window:13 range:(0,20) SAD (b) window:20 range:(0,20) SAD

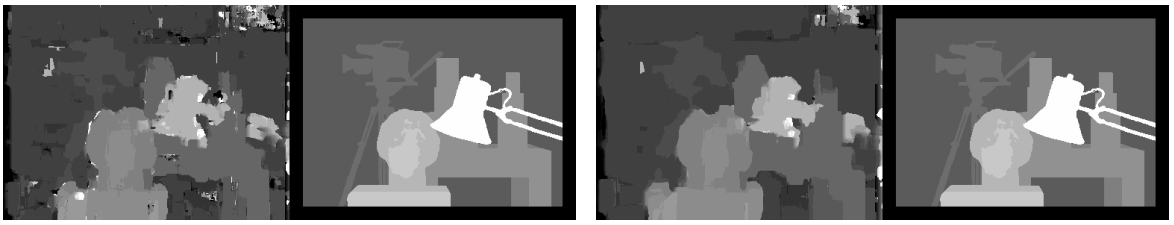
图 4: Task 1-Q2:SAD

对于 normalized_correlation 可以看出在 window size 较小时噪点非常多，似雪花状，调大 window size 后轮廓边缘被“模糊”化，平滑度较高，但也损失了相关细节, 综合考虑在 window_size 为 20 时坏点较少，比较合适。



(a) window:2 range:(0,20) ZNCC (b) window:7 range:(0,20) ZNCC

图 5: Task 1-Q2:ZNCC

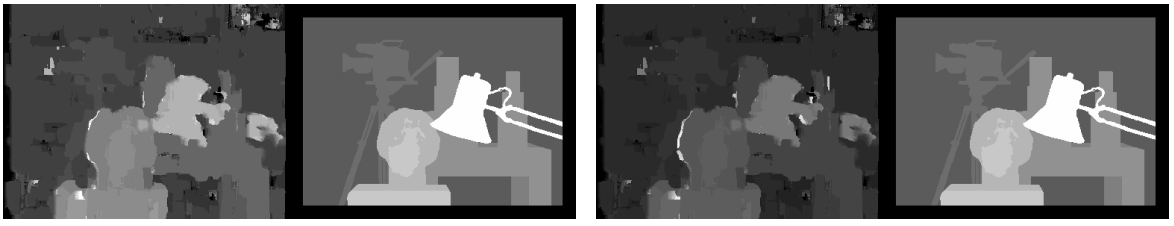


(a) window:13 range:(0,20) ZNCC (b) window:20 range:(0,20) ZNCC

图 6: Task 1-Q2:ZNCC

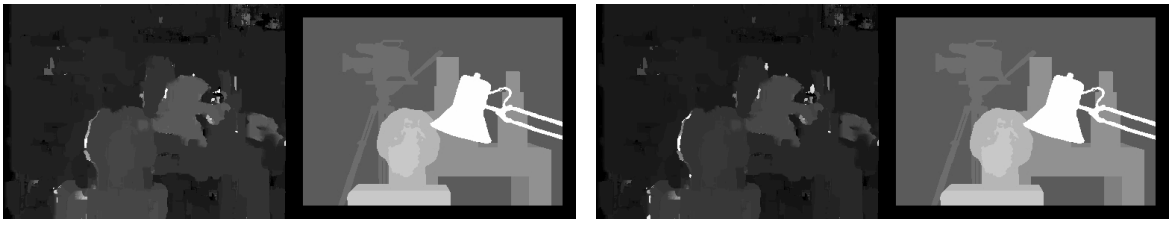
1.2.3 What is the maximum disparity range that makes sense for the given stereo pair?

考虑 window size 为 13，使用 SSD 作为 matching function: 可以看出随着 range 增大, 图像错匹越来越严重，图片整体越来越暗 (disparity_map 整体增大)，有意义的 range 最大值大概在 50 左右



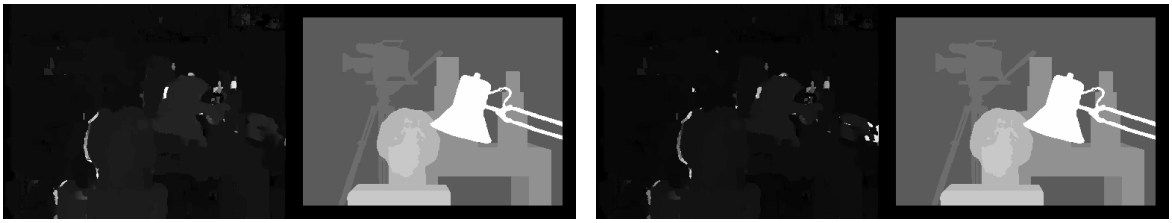
(a) window:13 range:(0,20) SSD (b) window:13 range:(0,30) SSD

图 7: Task 1-Q3



(a) window:13 range:(0,40) SSD (b) window:13 range:(0,50) SSD

图 8: Task 1-Q3



(a) window:13 range:(0,100) SSD (b) window:13 range:(0,150) SSD

图 9: Task 1-Q3

1.2.4 Which matching function may work better for the given stereo pair?

综合上述结果来看，matching function 中 SSD 在给定图片的双目匹配中效果最佳。

1.3 Discuss the trade-offs between different hyperparameters on quality and time.

window size 对时间开销影响较小，对于图片生成质量来说增大 window size 可以提高图片的平滑性，类似一个“滤波”，但是会丢失图片的细节信息。disparity range 与时间开销基本成正比，较小的 disparity range 在局部匹配上效果更好，disparity range 应该保持在一个较小但足以实现正确匹配的范围。对于 matching function，SSD 和 SAD 时间开销远小于 normalized_correlation, 在效果上 SSD 匹配效果最好。

1.4 Choose the best hyperparameters and show the corresponding disparity map.

选择 window size:15,disparity range:(0,15),matching function:SSD

图 10: Task1-window:15,disparity range:(0,15),SSD

1.5 Compare the best disparity map with the ground truth map, discuss the differences and limitations of basic stereo matching.

相比于 ground truth map，我的 disparity map 在图片上存在较多的噪点，物体边缘轮廓线不够平滑，图片的部分细节丢失较严重。

局限：basic stereo matching 只考虑到了局部最优，对于每个点求其在 disparity range 范围内最佳的匹配点，未考虑部分如匹配顺序 (从左往右匹配)，近邻原则 (左边图相近的两个 pixel 在右边图也相近) 等局部约束，且如果出现了重复的图样，错匹的概率就大大增加了。

2 Depth from Disparity

2.1 Pointcloud Visualization

思路：

对于 compute_depth_map, 遍历计算出的 disparity_map 的每个像素点的 disparity，然后利用公式 $depth(p) = \frac{focal\_lentgh \cdot baseline}{disparity(p)}$, 对深度进行计算，如果对应点的 $disparity(p) <= 0$, 那么直接忽略该点

```
    depth_map = np.zeros(np.shape(disparity_map))
    img_H,img_W = np.shape(disparity_map)
    for row_ in range(img_H):
        for col_ in range(img_W):
            if disparity_map[row_,col_]<=0:
                continue
            depth_map[row_,col_]=baseline*focal_length/disparity_map[row_,col_]
    return depth_map
```

然后遍历 depth_map, 将深度合法的点坐标加入到 points 中，再把对应像素的颜色加入到 colors 中 (要将 colors 归一至 [0,1] 范围), 并去除坏点，主要思路为：计算 depth map 的标准差，然后根据标准差把偏离 mean depth 较大的点的深度设为 0(忽略).

```
def task2_visualize_pointcloud(
    ref_img: np.ndarray,    # shape (H, W, 3)
    disparity_map: np.ndarray, # shape (H, W)
    save_path: str = 'output/task2_tsukuba.ply'
):
    baseline = 10
    focal_length = 10
    depth_map = task2_compute_depth_map(disparity_map, baseline, focal_length)

    new_depth_map=depth_map.copy()

    mean_depth = np.mean(depth_map)
    std_depth = np.std(depth_map)
    threshold_=1.7

    depth_distance_map = np.abs(depth_map-mean_depth)
    new_depth_map[depth_distance_map>threshold_*std_depth]=0
    # cut some outliers

```

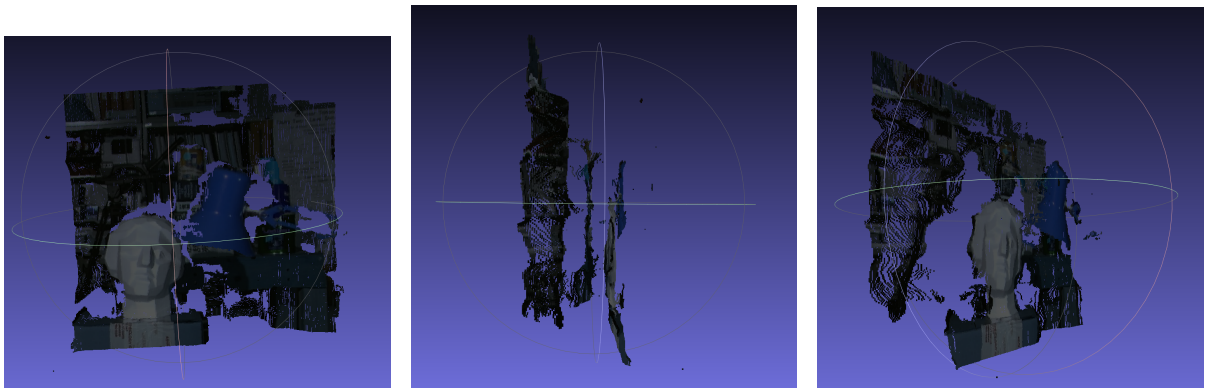
```
    img_H,img_W,_ = np.shape(ref_img)
    # Points
    new_depth_map = (new_depth_map/np.max(new_depth_map))*150 #归一化并放缩


    points=[]
    colors=[]
    for i in range(img_H):
        for j in range(img_W):
            tmp_depth=new_depth_map[i,j]
            if tmp_depth==0:
                continue
            points.append([j,i,tmp_depth])
            colors.append(ref_img[i,j]/255.0)

    # Save pointcloud to ply file
    pointcloud = trimesh.PointCloud(points, colors)
    os.makedirs(os.path.dirname(save_path), exist_ok=True)
    pointcloud.export(save_path, file_type='ply')
```

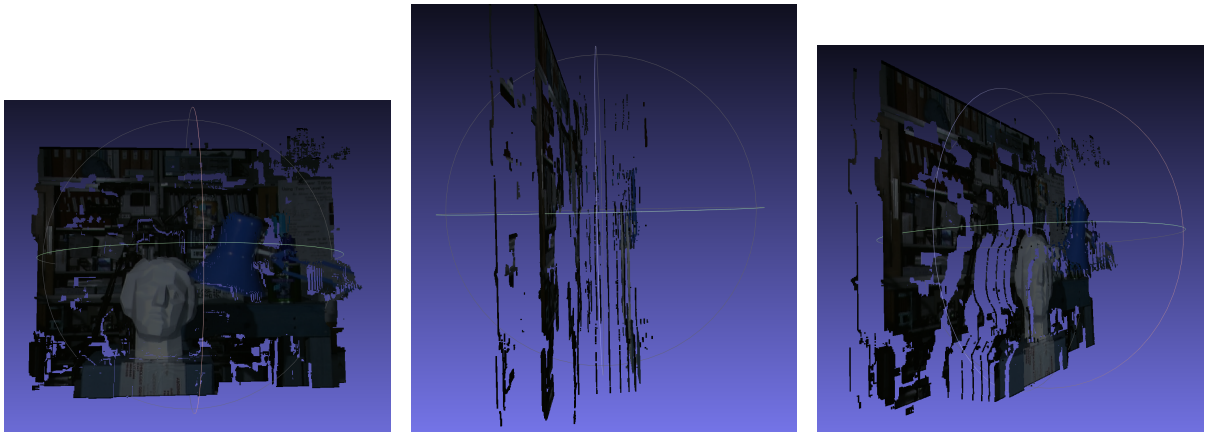
2.1.1 Report

得到结果生成的点云图如下：



(a) cv2.StereoBM view0　(b) cv2.StereoBM view1　(c) cv2.StereoBM view2

图 11: Task 2-cv2.StereoBM Views



(a) My optimal view0 (b) My optimal view1 (c) My optimal view2

图 12: Task 21-My optimal Views

Discuss:

发现与 cv2.StereoBM 相比，主要有以下几点不同：

1.cv2.StereoBM 的点云图留了很多空白，即舍弃掉了许多 depth=0 的点，而我的点云图在主视角上看则基本上严丝合缝，基本没有 depth=0 的点。

2.cv2.StereoBM 的点云图很“干净”，没有一些不正常的点散落在主点云区域之外，而我的点云图从侧面看上去会有很多明显的坏点散落。

3 Stereo Matching with Dynamic Programming

3.1 Algorithm Implementation

思路：

对于 ref_img 和 sec_img, 先初始化一个 shape 为 (img_H,img_W) 的 disparyty_map_dp 作为最后返回的 disparity_map。然后开始扫描 img 的每一行，对每一个 row_，取 ref_img[row_] 与 sec_img[row_] 的像素进行 dp, 初始化 dp_matrix 矩阵 (shape:img_W,img_W,initial:inf) 用于记录 cost,dp_matrix[i,j] 含义为已考虑完 ref_img[row_,0:j+1] 与 sec_img[row_,0:i+1] 的匹配后最小的 cost，再初始化 dp_record_former_matrix(shape: img_W,img_W,initial:4) 用于记录路径 (0:from dp[i-1,j-1],1:from dp[i,j-1],2:from dp[i-1,j])。然后我们需要初始化 dp_matrix 和 dp_record_former_matrix 的第一行，从左到右，表示 sec_img[row_,0] 像素无法与 ref[i] 像素匹配，直接惩罚，记录路径为 1(from left)

```
for row_ in range(img_H):
    dp_H,dp_W = img_W,img_W
    dp_matrix = np.full((dp_W,dp_W),float('inf'),dtype=np.float32)
    dp_record_former_matrix = np.full((dp_W,dp_W),4.0,dtype=np.int32)

    # initialize [0,0]
    dp_matrix[0,0]=min(np.sum(pad_ref_img[row_:row_+window_size,0:0+window_size]-
                    pad_sec_img[row_:row_+window_size,0:0+window_size]**2),occlusionConstant)

```

```
    dp_record_former_matrix[0,0]=-1

    # sec为右侧视角 正确的匹配sec在ref上的对应点应该在sec左侧
    for i in range(1,disparity_max):# initialize row0
        dp_matrix[0,i]=dp_matrix[0,i-1]+occlusionConstant
         # 横着走表示 ref[i]像素无法与sec[1]像素匹配,直接惩罚
        dp_record_former_matrix[0,i]=1
```

然后我们就可以正式开始 dp 了，我们从上往下，从左往右进行 dp 操作，对于每一个 dp_row, 我们只考虑 dp_col 处于 (dp_row,min(dp_row+disparity_max,dp_W)) 范围内的 dp_col, 这样减少了大量的不必要计算，加快了 dp 效率，对于 dp_matrix[i,j], 我们有三个来源: 来自左上，来自左，来自右。其中来自左上表示成功匹配 ref[j] 像素与 sec[i] 像素，需要加入 cost[i,j]；来自左表示跳过 ref[j], 需要被惩罚, 来自上表示跳过 sec[i], 需要被惩罚, 得到 dp 方程如下 (其中 cost 为对一个以目标像素为中心的 window 利用 L2 loss):

$$dp[i,j] = min \begin{cases} dp[i-1,j-1] + cost[i,j] \\ dp[i,j-1] + occlusionConstant \\ dp[i-1,j] + occlusionConstant \end{cases}$$

然后根据取哪个 min 来记录 dp_record_former_matrix

```
for dp_row in range(1,dp_H): # 从上往下 从左往右dp
    for dp_col in range(dp_row,min(dp_row+disparity_max,dp_W)): #
        只考虑disparity_range内的点，且对于sec[dp_row],目标ref只会出现在右侧
        # print(cal_ref_img[row_,dp_col],cal_sec_img[row_,dp_row])
        min1 = dp_matrix[dp_row-1,dp_col-1]+
                  np.sum((pad_ref_img[row_:row_+window_size,dp_col:dp_col+window_size]-
                  pad_sec_img[row_:row_+window_size,dp_row:dp_row+window_size])**2) #
                     从左上,正确匹配
        min2 = dp_matrix[dp_row,dp_col-1]+occlusionConstant # 从左 无法匹配 表示跳过ref[dp_col]
        min3 = dp_matrix[dp_row-1,dp_col]+occlusionConstant # 从上 无法匹配 表示跳过sec[dp_row]

        dp_min_lst =np.array([min1,min2,min3])
        dp_min_index = np.argmin(dp_min_lst)
        dp_matrix[dp_row,dp_col]=dp_min_lst[dp_min_index]
        dp_record_former_matrix[dp_row,dp_col]=dp_min_index
```

最后从 dp_record_former_matrix[dp_H-1,dp_W-1] 恢复路径

```
# 恢复path
cur_col = dp_W-1
cur_row = dp_H-1

dp_best_disparity=np.zeros((img_W))
```

```
while dp_record_former_matrix[cur_row,cur_col]!=-1:# when [0,0] break
    if dp_record_former_matrix[cur_row,cur_col]==0: #从左上
        dp_best_disparity[cur_col]=cur_col-cur_row
        cur_col-=1
        cur_row-=1
    elif dp_record_former_matrix[cur_row,cur_col]==1: # 从左
        dp_best_disparity[cur_col]=-1 #被跳过的ref像素
        cur_col-=1
    elif dp_record_former_matrix[cur_row,cur_col]==2: #从上
        cur_row-=1
    else:
        raise ValueError("Invalid dp_record_former_matrix value")
```

最后把被跳过的 ref 像素赋值为其左侧像素的 disparity, 增加一定的连续性, 最后把计算得到的 dp_best_disparity 赋值给对应 disparity_map_dp[row_]

```
for j in range(1,img_W):
    if dp_best_disparity[j]==-1:
        dp_best_disparity[j]=dp_best_disparity[j-1]
disparity_map_dp[row_]=dp_best_disparity
```

完整代码:

```
def task3_compute_disparity_map_dp(ref_img, sec_img):
    task3_time_begin = time.time() # record time
    print("task3 start!")
    img_H,img_W =np.shape(ref_img)

    disparity_range = (0, 14)
    disparity_max = disparity_range[1]
    occlusionConstant=200.0
    window_size = 3
    padding_=window_size//2

    cal_ref_img =np.array(ref_img).astype(np.float32)
    cal_sec_img =np.array(sec_img).astype(np.float32)

    pad_ref_img =
        np.pad(ref_img,((padding_,padding_),(padding_,padding_)),'edge').astype(np.float64)#
        padding to use window
    pad_sec_img =
        np.pad(sec_img,((padding_,padding_),(padding_,padding_)),'edge').astype(np.float64)

```

```
    disparity_map_dp = np.zeros((img_H,img_W),dtype=np.float32)

    for row_ in range(img_H):
        # calculate disparity by row

        # 动态规划: 从第一列开始走，只能往下，往右，往右下，mininize map_dp[img_W-1][img_W-1]
        # 恢复路径的方式是记录每个点的前继方式
        # map_dp[i][j]表示已经考虑过sec 前i+1个像素点 和 ref 前j+1个像素点 的匹配,记录最小的cost
        # print(np.shape(row_matrix))
        dp_H,dp_W = img_W,img_W
        dp_matrix = np.full((dp_W,dp_W),float('inf'),dtype=np.float32)
        dp_record_former_matrix = np.full((dp_W,dp_W),4.0,dtype=np.int32)

        dp_matrix[0,0]=min(np.sum(pad_ref_img[row_:row_+window_size,0:0+window_size]-pad_sec_img[row
            # initialize [0,0]
        dp_record_former_matrix[0,0]=-1

        # sec为右侧视角 正确的匹配sec在ref上的对应点应该在sec左侧
        for i in range(1,disparity_max):# initialize row0
            dp_matrix[0,i]=dp_matrix[0,i-1]+occlusionConstant # 横着走表示
                ref[i]像素无法与sec[1]像素匹配,直接惩罚
            dp_record_former_matrix[0,i]=1

        for dp_row in range(1,dp_H): # 从上往下 从左往右dp
            for dp_col in range(dp_row,min(dp_row+disparity_max,dp_W)): #
                只考虑disparity_range内的点，且对于sec[dp_row],目标ref只会出现在右侧
                # print(cal_ref_img[row_,dp_col],cal_sec_img[row_,dp_row])
                min1 =
                    dp_matrix[dp_row-1,dp_col-1]+np.sum((pad_ref_img[row_:row_+window_size,dp_col:dp_c
                    # 从左上,正确匹配
                min2 = dp_matrix[dp_row,dp_col-1]+occlusionConstant # 从左 无法匹配
                    表示跳过ref[dp_col]
                min3 = dp_matrix[dp_row-1,dp_col]+occlusionConstant # 从上 无法匹配
                    表示跳过sec[dp_row]

                dp_min_lst =np.array([min1,min2,min3])
                dp_min_index = np.argmin(dp_min_lst)
                dp_matrix[dp_row,dp_col]=dp_min_lst[dp_min_index]
                dp_record_former_matrix[dp_row,dp_col]=dp_min_index

        # 恢复path
        cur_col = dp_W-1
        cur_row = dp_H-1
```

```
54
55          dp_best_disparity=np.zeros((img_W))
56          # print("begin recover best path!")
57          while dp_record_former_matrix[cur_row,cur_col]!=-1:# when [0,0] break
58              if dp_record_former_matrix[cur_row,cur_col]==0: #从左上
59                  dp_best_disparity[cur_col]=cur_col-cur_row
60                  cur_col-=1
61                  cur_row-=1
62              elif dp_record_former_matrix[cur_row,cur_col]==1: # 从左
63                  dp_best_disparity[cur_col]=-1
64                  cur_col-=1
65              elif dp_record_former_matrix[cur_row,cur_col]==2: #从上
66                  cur_row-=1
67              else:
68                  raise ValueError("Invalid dp_record_former_matrix value")
69
70          for j in range(1,img_W):
71              if dp_best_disparity[j]==-1:
72                  dp_best_disparity[j]=dp_best_disparity[j-1]
73          disparity_map_dp[row_]=dp_best_disparity
74
75
76      task3_time_end=time.time()# record time
77      print("task3 end!")
78      print("task3 time cost:",task3_time_end-task3_time_begin)
79      return disparity_map_dp
```

3.2 Report

经过不断调整 disparity range,occlusionConstant 和 window_size, 发现在 disparity range=(0,14) ,occlusionConstant=80.0,window_size=1 时效果较好

时间开销为：”task3 time cost: 7.9398932456970215”, 时间开销较少

得到 disparity map:

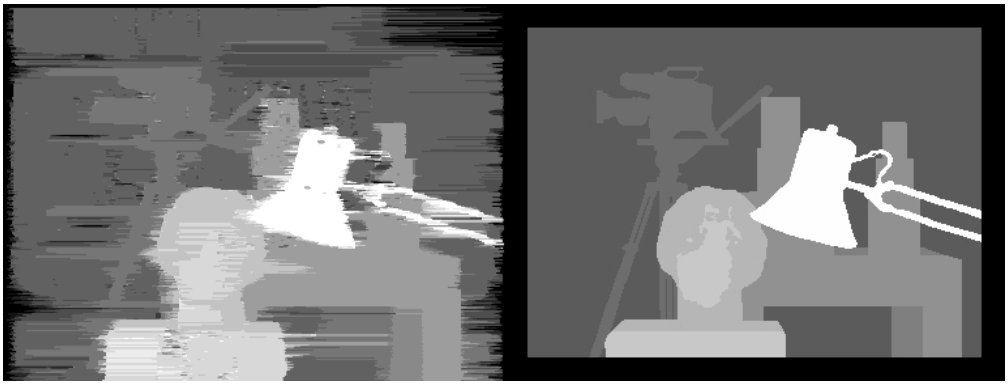
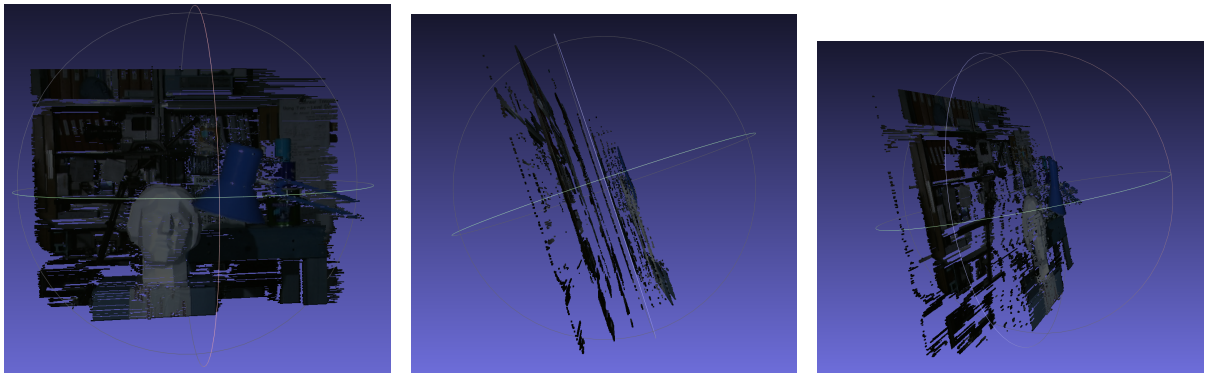


图 13: Task3-Dynamic Programming,disparity range=(0,14),occlusionConstant=80.0,window size=1

生成点云：



(a) My dp view0-better (b) My dp view1-better (c) My dp view2-better

图 14: Task 23-My dp Views-better

与 basic stereo matching algorithm 相比，通过 Stereo Matching with Dynamic Programming 得到的结果有以下特点：

- disparity map 中物体的边缘轮廓线能很好地与 ground truth 上的图像边缘线吻合，没有扭曲和”钝化”, 这是因为 dynamic Programming window size=1, 没有进行“滤波”操作，每次都是像素点与像素点的直接匹配。
- dp 得到的 disparity map 存在较多的噪点，匹配失败较多，可能是因为匹配要求设定的 occlusion-Constant 较小，匹配要求较严格
- dp 得到的 disparity map 保留了较多的细节 (如保留下了完整的台灯支撑)，深度图层次更加明显。
- dp 得到的 disparity 存在“拉丝”情况，可能是由于 disparity range 较大，造成匹配不准确。
- dp 得到的点云图比起 basic stereo matching algorithm 坏点密度较低，可能是由于得到的 disparity map 在局部上更加具有“连续性”

以上所有使用到的图片和点云文件都可以在”visualization results” 文件夹下 task1-show,task2-show,task3 下找到。