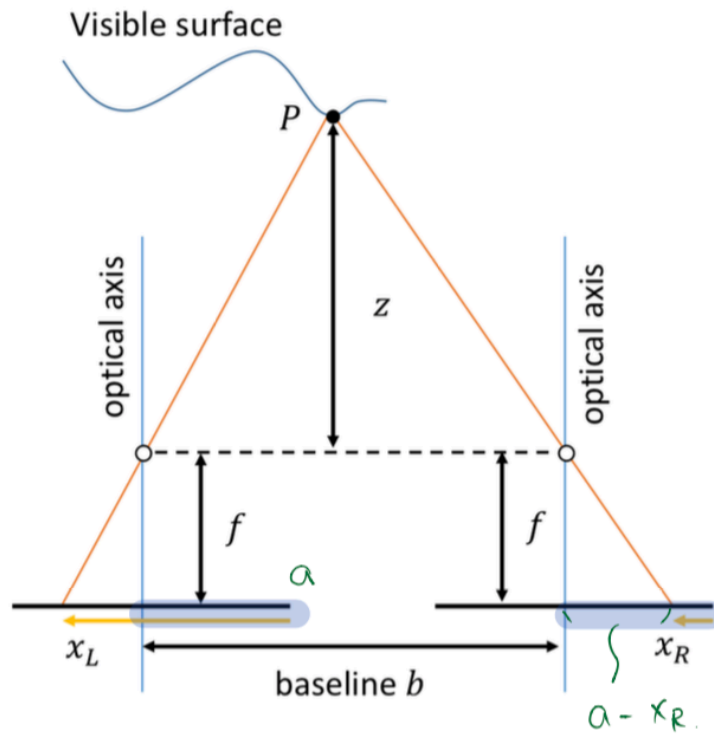## Problem 1

$$b + (x_L - a) + (a - x_R) : b = z + f : z$$

(Due to the similarity of two triangles)

$$b + x_L - x_R : b = z + f : z$$

$$b + d : b = z + f : z.$$

$$zb + bf = zb + zd \qquad d = \frac{f \cdot b}{z}$$

Problem 2

Matching cost
3-channeled census cost

```python
def census_cost(mat_1, mat_2):
    '''
    The function is used to calculate the census cost with the type np.array((k, k), dtype=boolean)

    Arg(s):
        mat_1(np.array) : BGR window with shape (k, k , 3)
        mat_2(np.array) : BGR window with shape (k, k , 3)

    Return(s):
        cost(int)       : XOR result of the two windows
    '''
    radius = mat_1.shape[0]//2
    medium_1, medium_2 = mat_1[radius, radius], mat_2[radius, radius]
    # convert teh window into boolean form
    mat_1 = mat_1 >= medium_1
    mat_2 = mat_2 >= medium_2
    mat_1, mat_2 = mat_1.reshape(-1, 1), mat_2.reshape(-1, 1)
    cost = np.sum(np.equal(mat_1, mat_2) == False)
    return cost
```

Cost aggregation
guided filter from cv2 with radius=16, epsilon=50

```python
    for disp in range(max_disp):
        disparity_left[:, :, disp] = guidedFilter(img_left.astype(np.uint8),
                                                  disparity_left[:, :, disp].astype(np.uint8),
                                                  16, 50, -1)
        disparity_right[:, :, disp] = guidedFilter(img_right.astype(np.uint8),
                                                   disparity_right[:, :, disp].astype(np.uint8),
                                                   16, 50, -1)
```

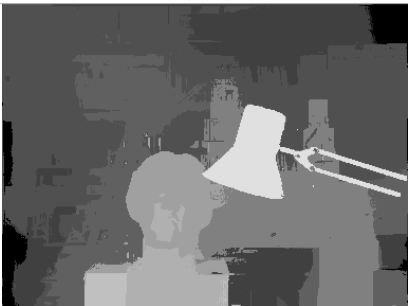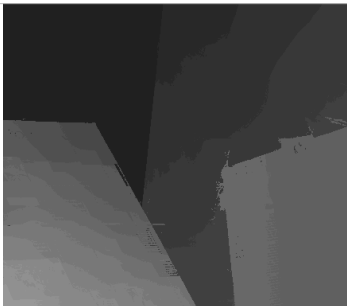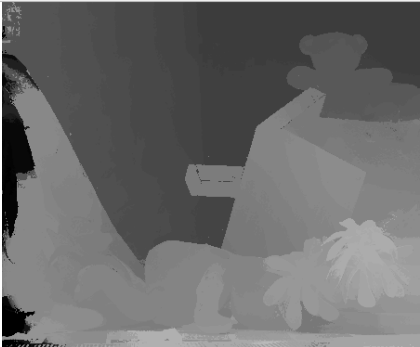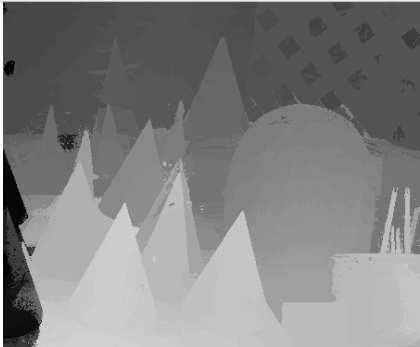## *Disparity optimization*
Winner Take All

```python
    disparity_left = np.argmin(disparity_left, axis=2).astype(np.uint8)
    disparity_right = np.argmin(disparity_right, axis=2).astype(np.uint8)
```
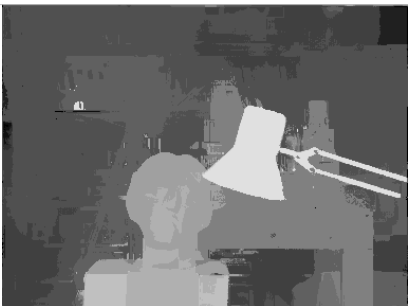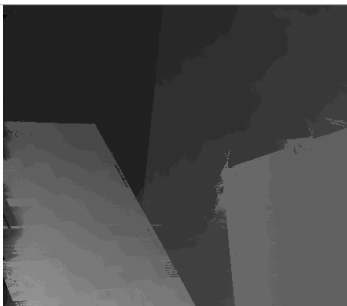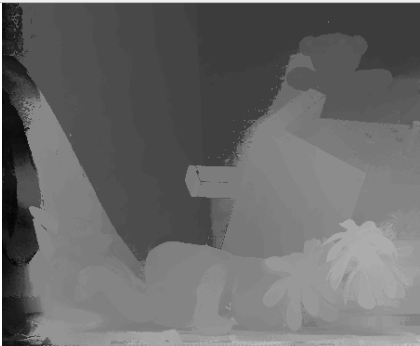
Disparity refinement
Left-Right check

```python
def consistency_check(disparity_left, disparity_right, max_disp):
    '''
    Follow the formula of the left-right checking
    Record the coordination of the holes

    Arg(s):
        disparity_left(np.array)   : the disparity map of img_left with shape (height, width)
        disparity_right(np.array)  : the disparity map of img_right with shape (height, width)
        max_dsip(int)              : the max disparity for calculation

    Return(s):
        ret(np.array)   : disparity map after left-rigth cheacking
        holes(set)      : the set of coordination of inconsistent points
    '''
    height, width = disparity_left.shape
    ret = disparity_left
    holes = set()
    for coor_y in range(height):
        for coor_x in range(max_disp, width):
            # the original check formula: Dl(x, y) == Dr(x-Dl(x, y), y)
            dis = int(disparity_left[coor_y, coor_x])
            if disparity_left[coor_y, coor_x] == disparity_right[coor_y, coor_x-dis]:
                ret[coor_y, coor_x] = disparity_left[coor_y, coor_x]
            else:
                holes.add((coor_y, coor_x))
    print('There are {:<03d} holes after left-right check'.format(len(holes)))
    return ret, holes
```

Hole-filling

```python
def hole_filling(disparity, holes):
    '''
    Find the valid point from left and right, and fill the holes with the closer point

    Arg(s):
        disparity(np.array) : disparity map after left-rigth cheacking with shape (height, width)
        holes(set)          : the set of coordination of inconsistent points

    Return(s):
        disparity(np.array)   : disparity map after hole-filling with shape (height, width)
    '''
    width = disparity.shape[1]
    for (hole_y, hole_x) in holes:
        left_point, right_point = 0, 0
        for pixel in range(1, 21):
            # if we find the valid points, stop the loop
            if left_point != 0 and right_point != 0:
                break
            # hold the boarders while searching the right side
            if hole_x + pixel >= width:
                pass
            elif (hole_y, hole_x + pixel) not in holes:
                right_point = pixel
            # hold the boarders while searching the left side
            if hole_x - pixel < 0:
                pass
            elif (hole_y, hole_x - pixel) not in holes:
                left_point = pixel
        # get the nearest valid pixel to fill the hole
        if right_point < left_point:
            disparity[hole_y, hole_x] = disparity[hole_y, hole_x + right_point]
        else:
            disparity[hole_y, hole_x] = disparity[hole_y, hole_x - left_point]
    return disparity
```

weighted median filter from cv2 with radius=5, epsilon=23

```python
disparity = weightedMedianFilter(img_left.astype(np.uint8),
                                 disparity.astype(np.uint8),
                                 23, 5, cv2.ximgproc.WMF_JAC)
```

Result

With refinement (avg: 7.72%)

error(%)	4.70	0.81
error(%)	13.63	11.74

Without refinement (avg: 9.24%)

error(%)	4.11	2.49
error(%)	17.66	12.72