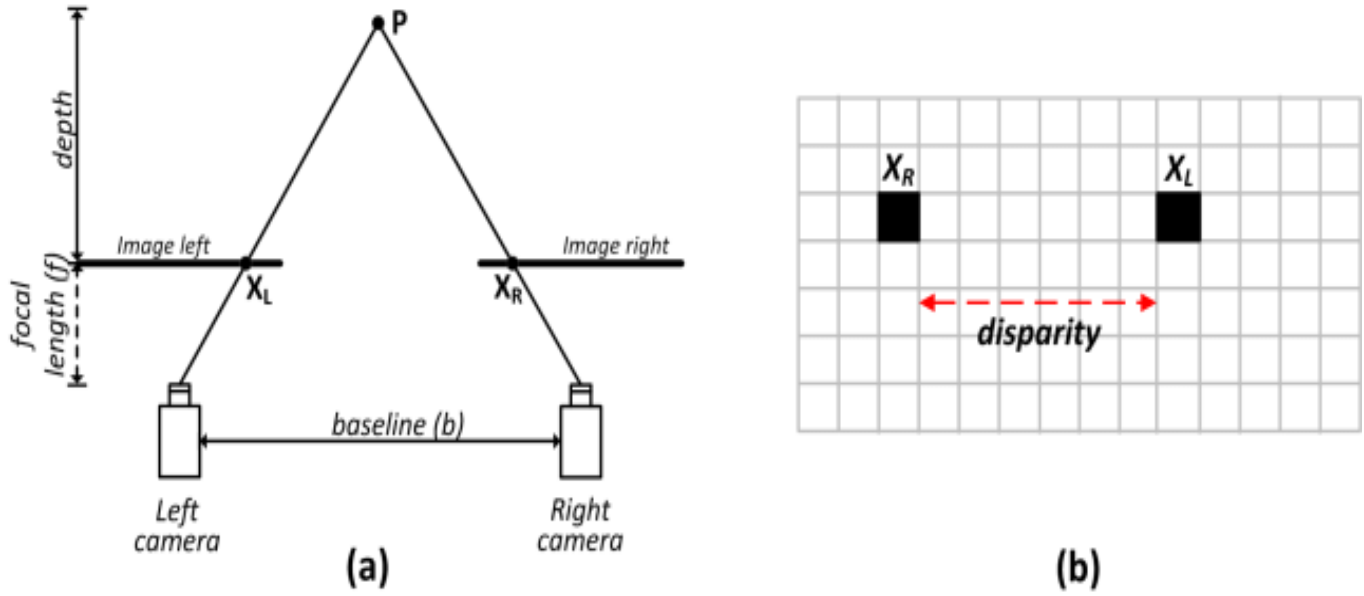
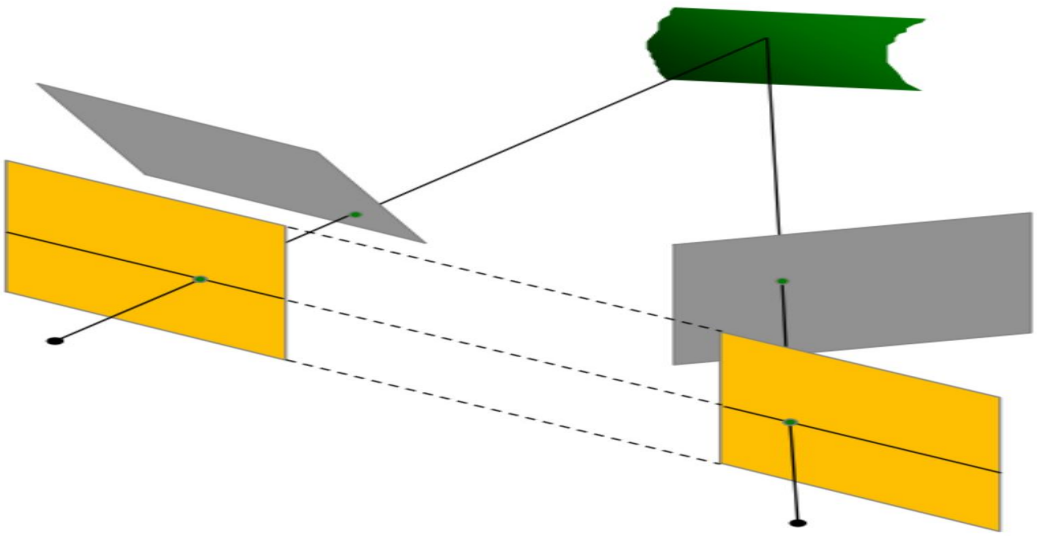
Stereo matching or disparity estimation

is the process of finding the pixels in the multiscopic views that correspond to the same 3D point in the scene. The rectified epipolar geometry simplifies this process of finding correspondences on the same epipolar line.



Algorithm

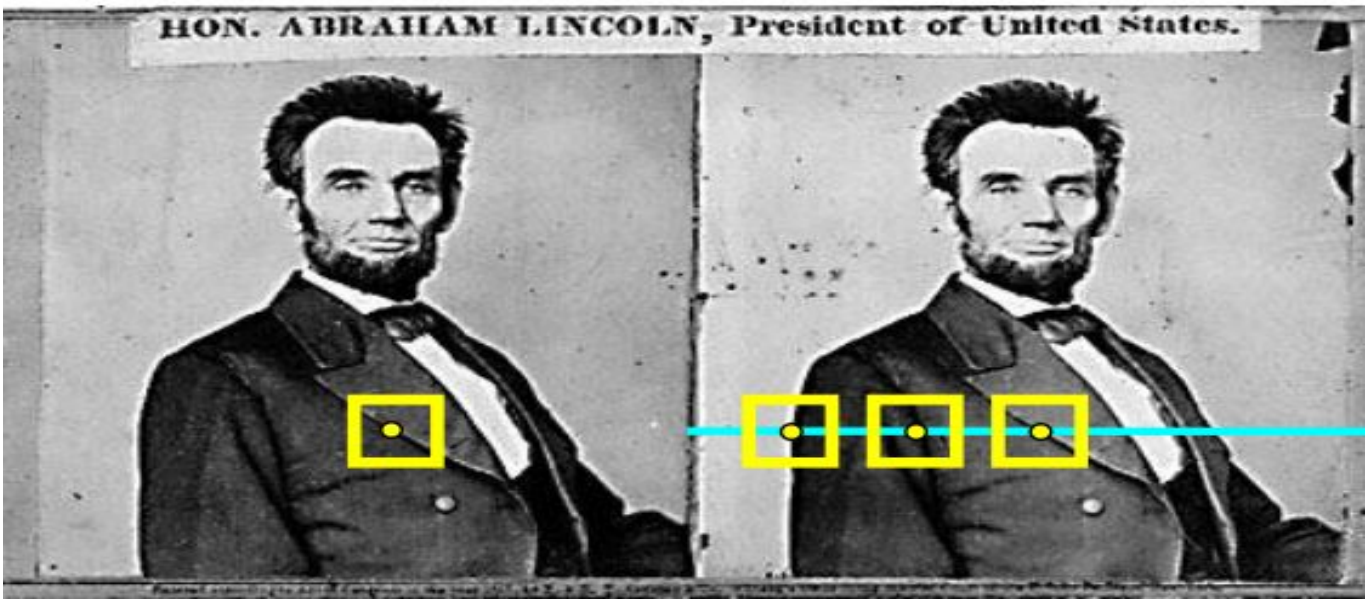
1. Rectify images (make epipolar lines horizontal)



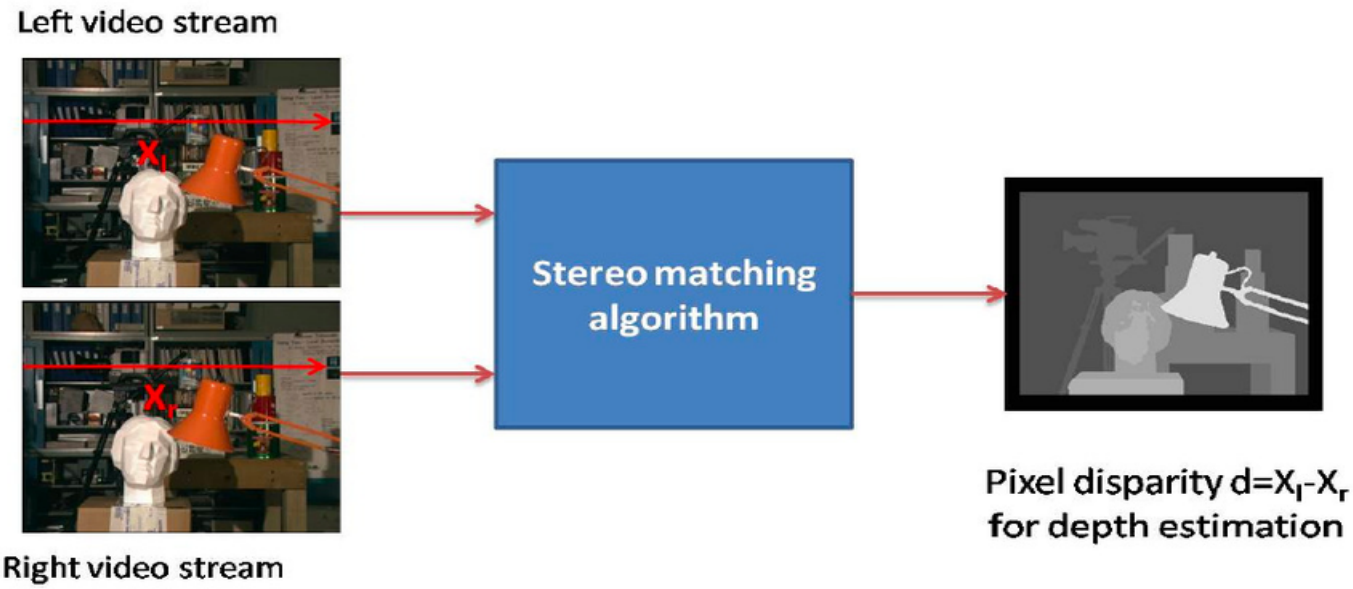
2. For each pixel

a. Find epipolar line

b. Scan line for best match



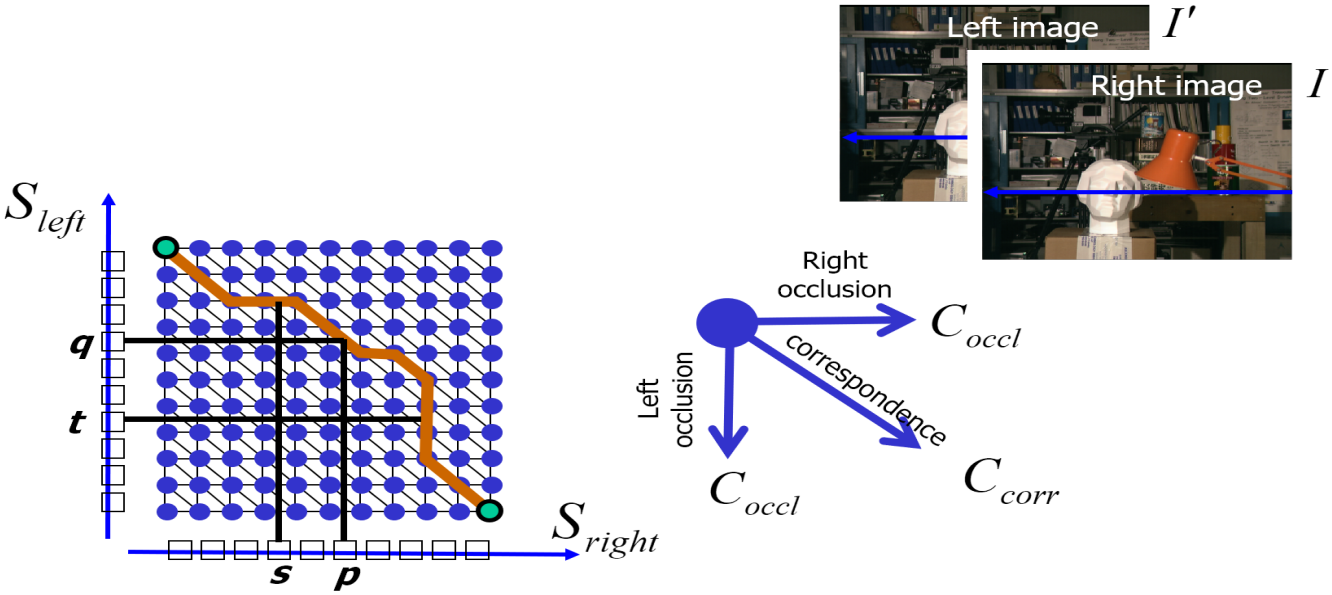
c. Compute depth from disparity



Scan line for best match

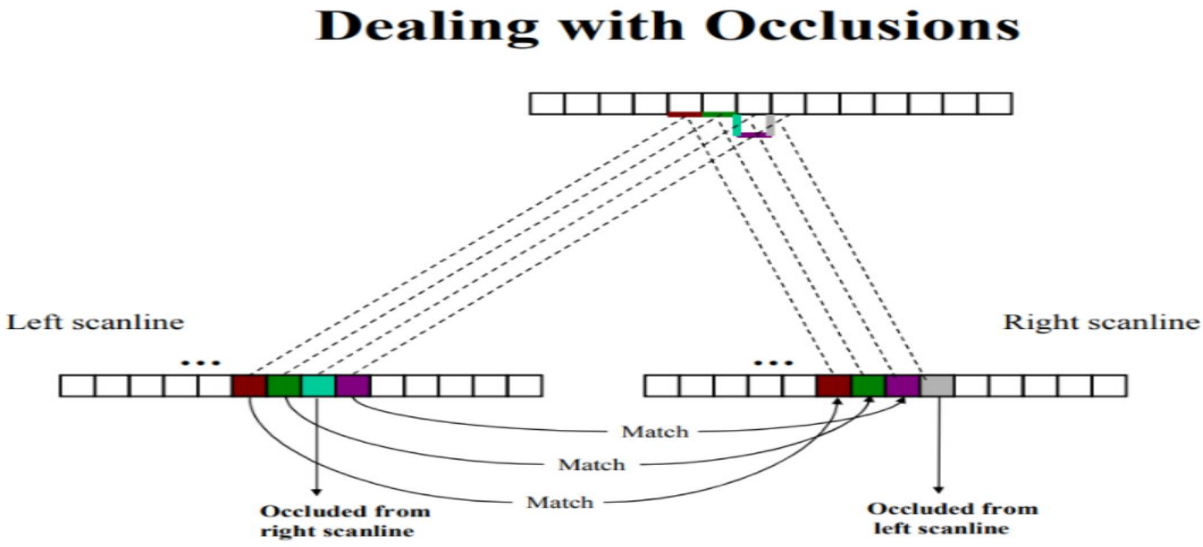
Stereo Matching with Dynamic Programming

Dynamic programming-based dense stereo matching improvement using an efficient search space reduction technique, DP algorithm which is selected in this paper has been discussed and implemented by Scharstein and Szeliski . Their goal was only to evaluate performance of the dynamic programming optimization method itself; so, they implemented the main part of the algorithm is considering other techniques such as shiftable windows and GCPs. This algorithm has been accepted as a standard 3-state dynamic programming in the literature and has been the base of implementation and improvements in Reduced search space (RSS)



Occlusions

occlusion in an image occurs when an object hides a part of another object



DP Algorithm

```
def matching_cost(I1, I2, sigma):
    return (int(I1) - int(I2)) ** 2 / sigma ** 2

def dynamic_prog_disparity(left_img, right_img, sigma, c0):
    assert left_img.shape == right_img.shape
    assert left_img.ndim == 2
```

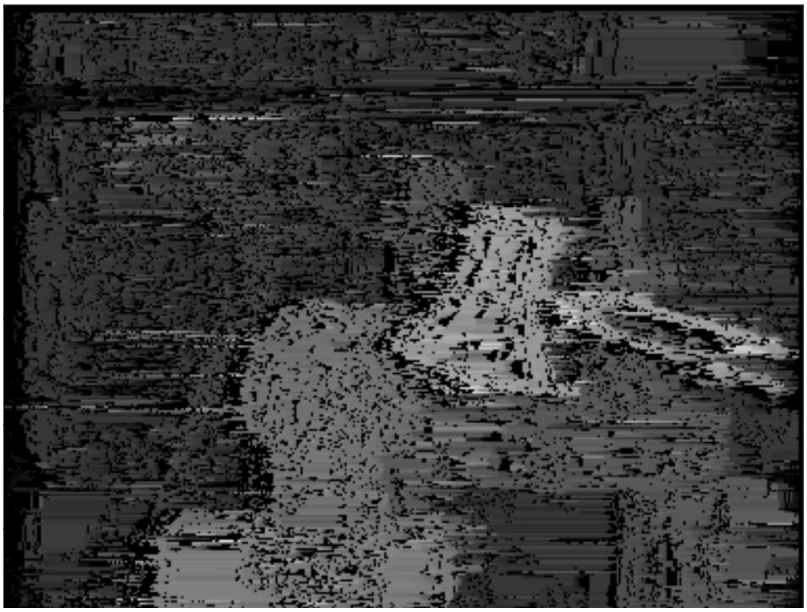
```
    assert right_img.ndim == 2
    assert sigma != 0
    assert c0 >= 0

    disparityl = np.zeros(left_img.shape)
    disparityr = np.zeros(right_img.shape)
    D_matrices = [] # it will be used in the bonus part

    # loop on the image rows
    for r in range(left_img.shape[0]):
        D = np.zeros((len(left_img[r]), len(left_img[r])), dtype=np.float32)

        # prepare the first row and column of D matrix
        for i in range(0, D.shape[0]):
            D[i, 0] = i * c0
            D[0, i] = i * c0
        D[1, 1] = matching_cost(left_img[r, 0], right_img[r, 0], sigma)

        # computing table step
        for i in range(1, D.shape[0]):
            for j in range(1, D.shape[1]):
                match_cost = matching_cost(left_img[r, i], right_img[r, j], sigma)
                D[i, j] = min(D[i - 1, j - 1] + match_cost , min(D[i - 1, j], D[i,
j - 1]) + c0)

        # backtracking step
        i = D.shape[0] - 1
        j = D.shape[1] - 1
        while (i > 0 and j > 0):
            '''
            You can construct the disparity matrix and prepare the D matrix to be
ploted in the bonus part by these steps:
            1 - choose the minimum value of D from (i - 1, j - 1), (i - 1, j), (i,
j - 1)
            2- Selecting (i - 1, j) corresponds to skipping a pixel in Il, so the
left disparity map of i is zero
            (set the pixel (i - 1, j) with the value -1 to be recognized in the
bonus part)
            3- Selecting (i, j - 1) corresponds to skipping a pixel in Ir, and the
right disparity map of j is zero
            (set the pixel (i, j - 1) with the value -1 to be recognized in the
bonus part)
            4- Selecting (i - 1, j - 1) matches pixels (i, j) and therefore both
disparity maps at this position are set to the
            absolute difference between i and j
            (set the pixels (i, j) with the value -1 to be recognized in the bonus
part)
            '''
            D[i, j] = -1
            if D[i - 1, j] < D[i - 1, j - 1] and D[i - 1, j] < D[i, j - 1]:
                i -= 1
            elif D[i, j - 1] < D[i - 1, j - 1] and D[i, j - 1] < D[i - 1, j]:
                j -= 1
            else:
```

```
                disparity = np.absolute(i - j)
                disparityl[r, i - 1] = disparity
                disparityr[r, j - 1] = disparity
                i -= 1
                j -= 1

            D[i, j] = -1
            D_matrices.append(D)

    return disparityl, disparityr, D_matrices
```

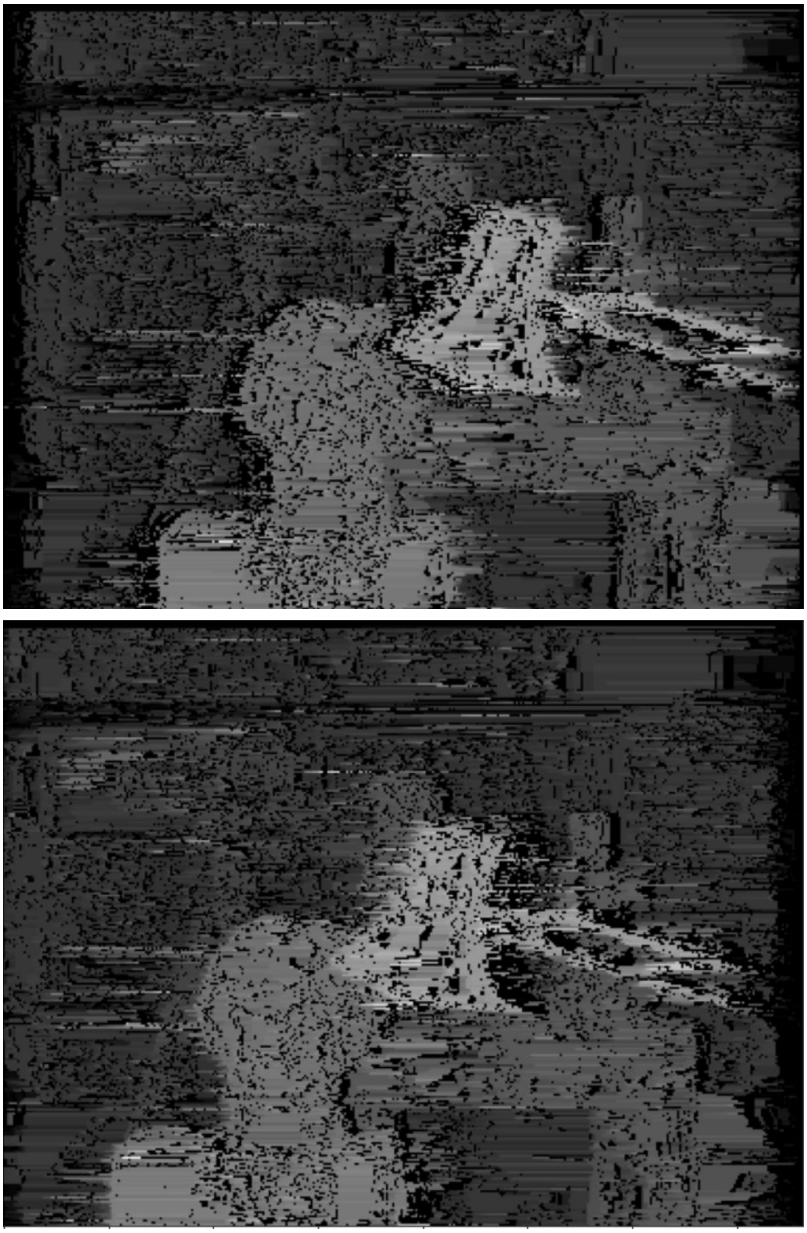
Sample Runs

First Image

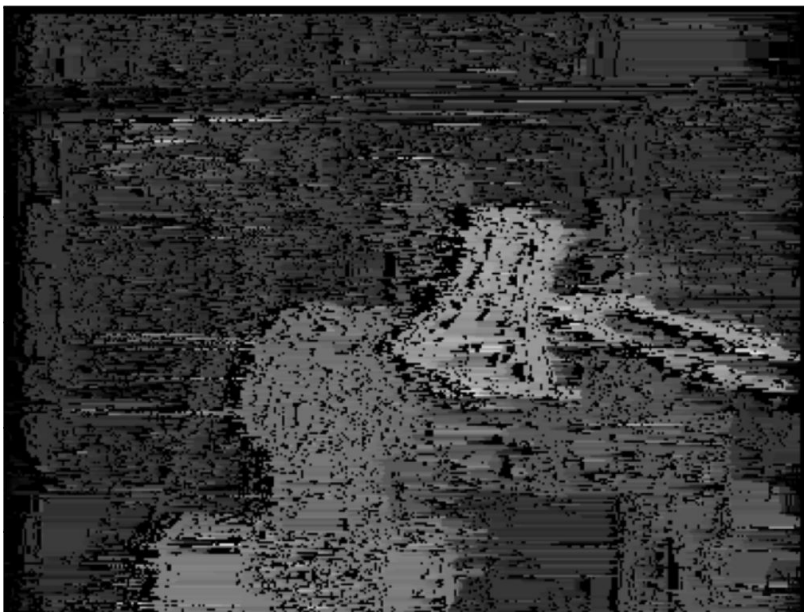
sigma = 2, c0 = 1



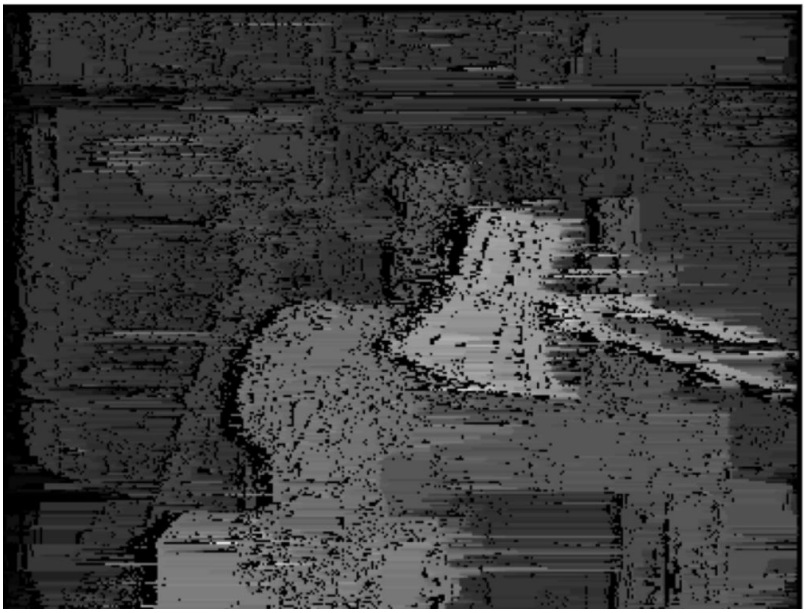
sigma = 4, c0 = 1



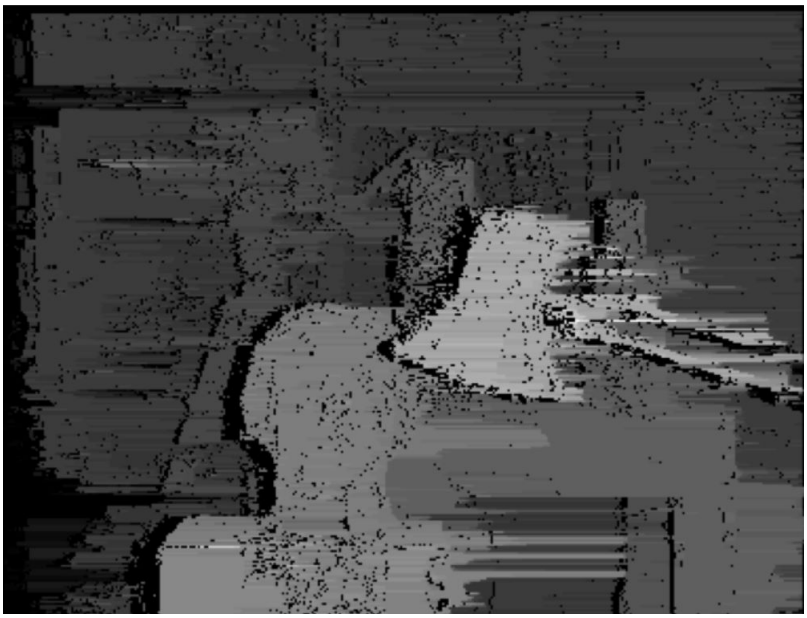
sigma = 0.5, c0 = 1



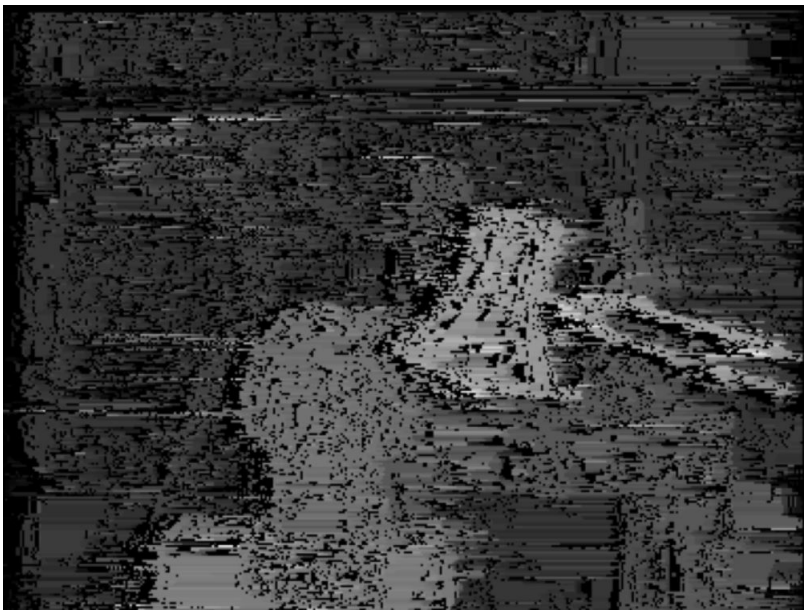
sigma = 2, c0 = 10



sigma = 4, c0 = 10



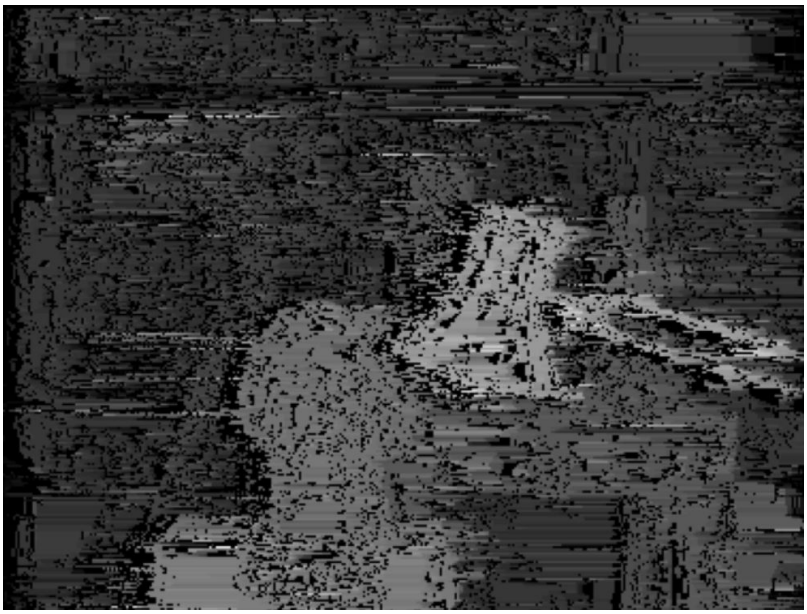
sigma = 0.5, c0 = 10



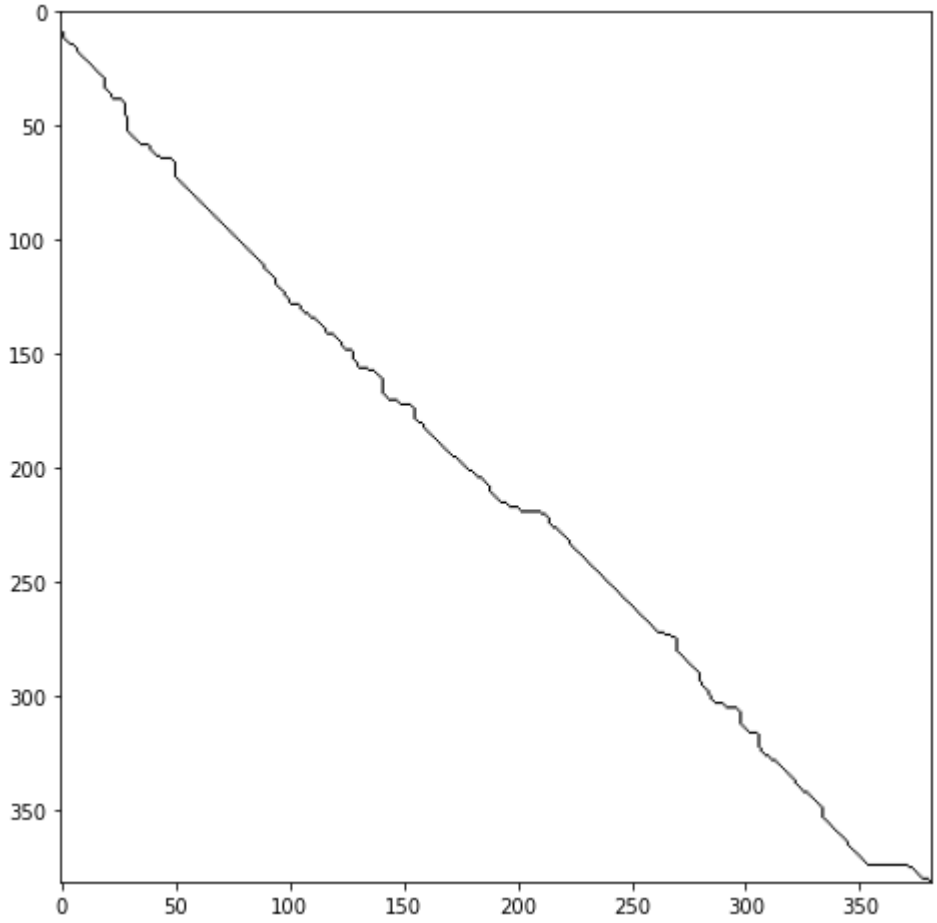
sigma = 2, c0 = 20

sigma = 4, c0 = 20

sigma = 0.5, c0 = 20

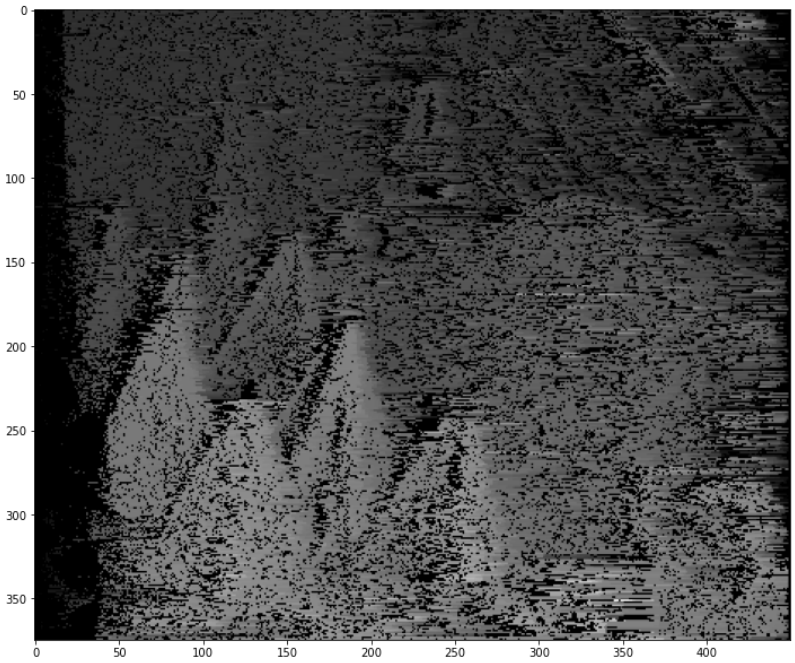


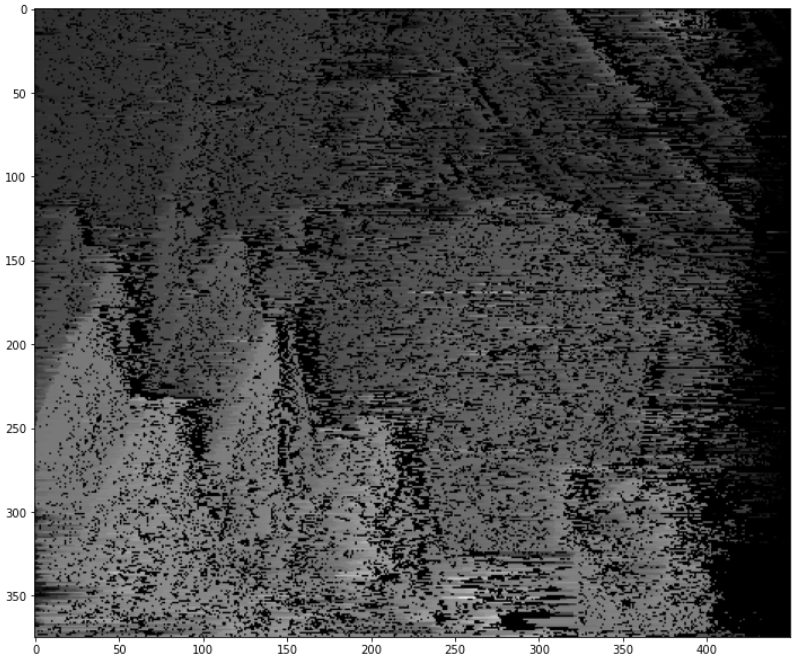
Bonus



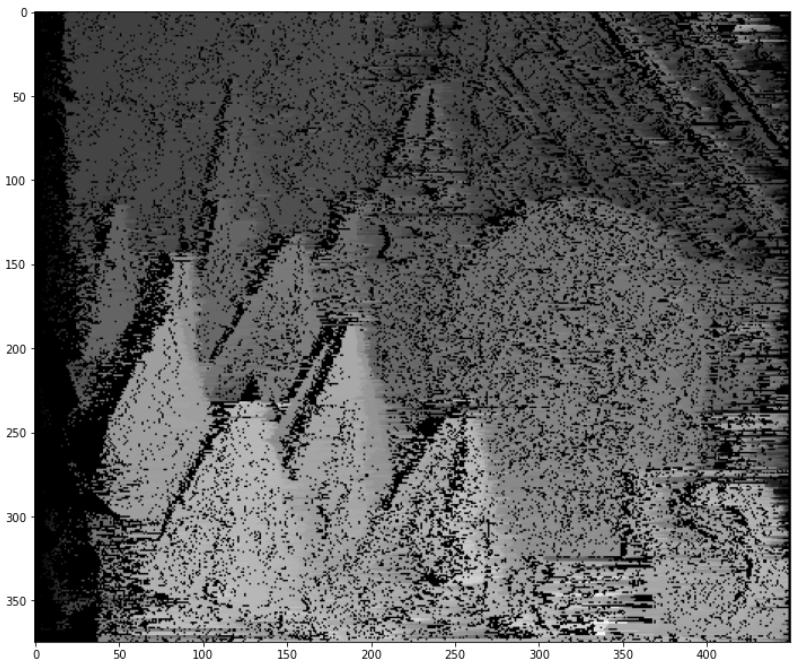
Second Image

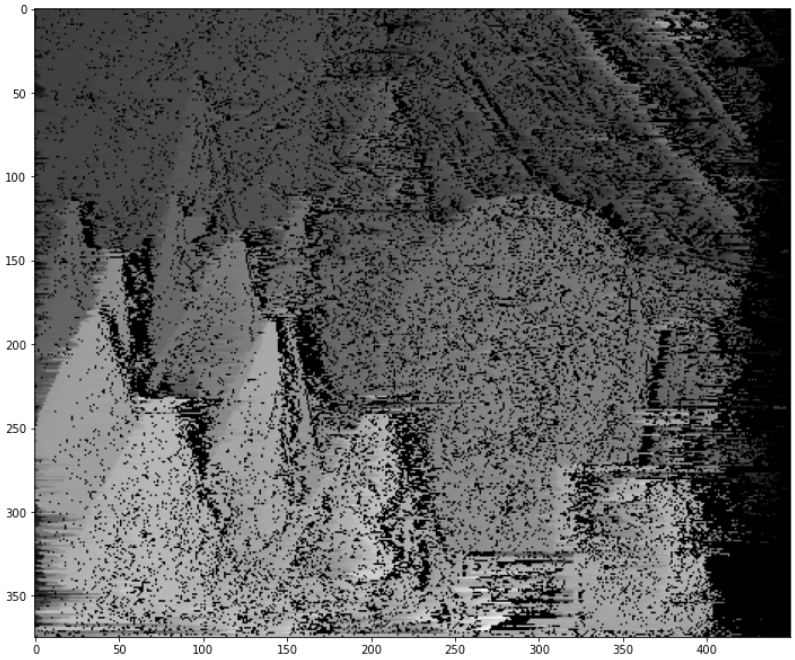
sigma = 2, c0 = 1


0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

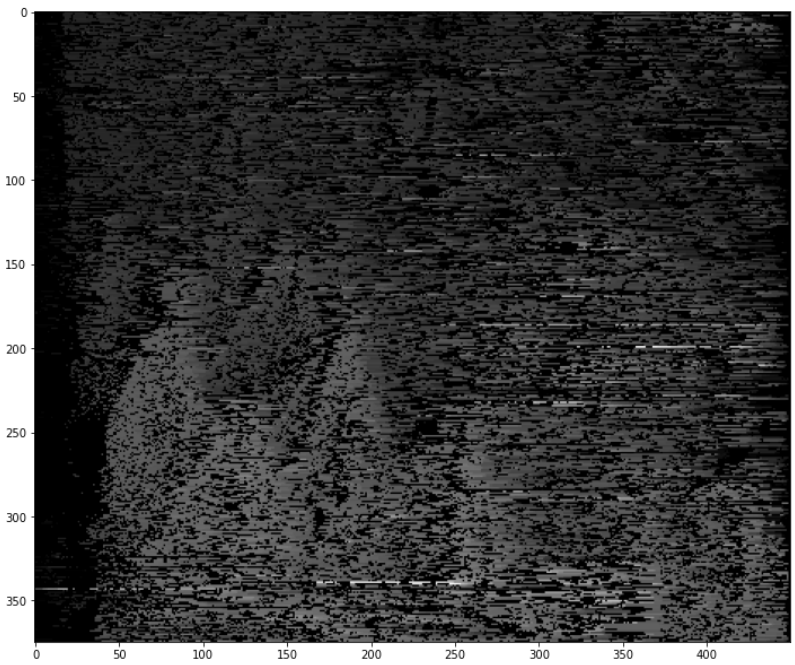

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

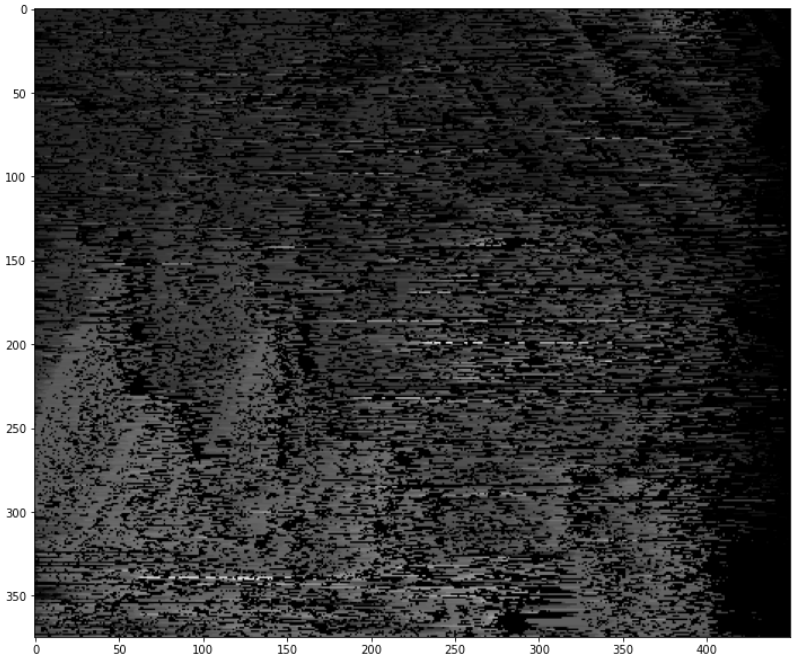
sigma = 4, c0 = 1


0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

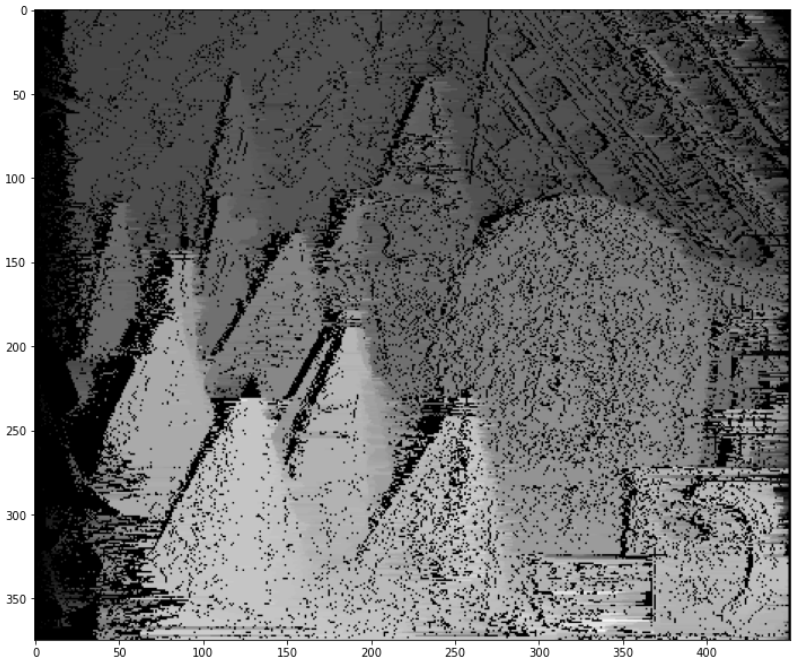

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

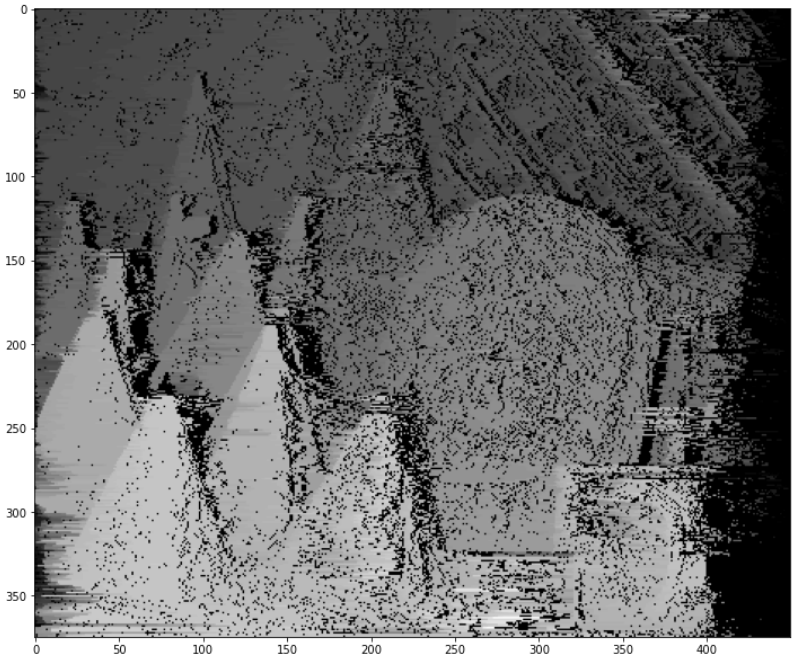
sigma = 0.5, c0 = 1


0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

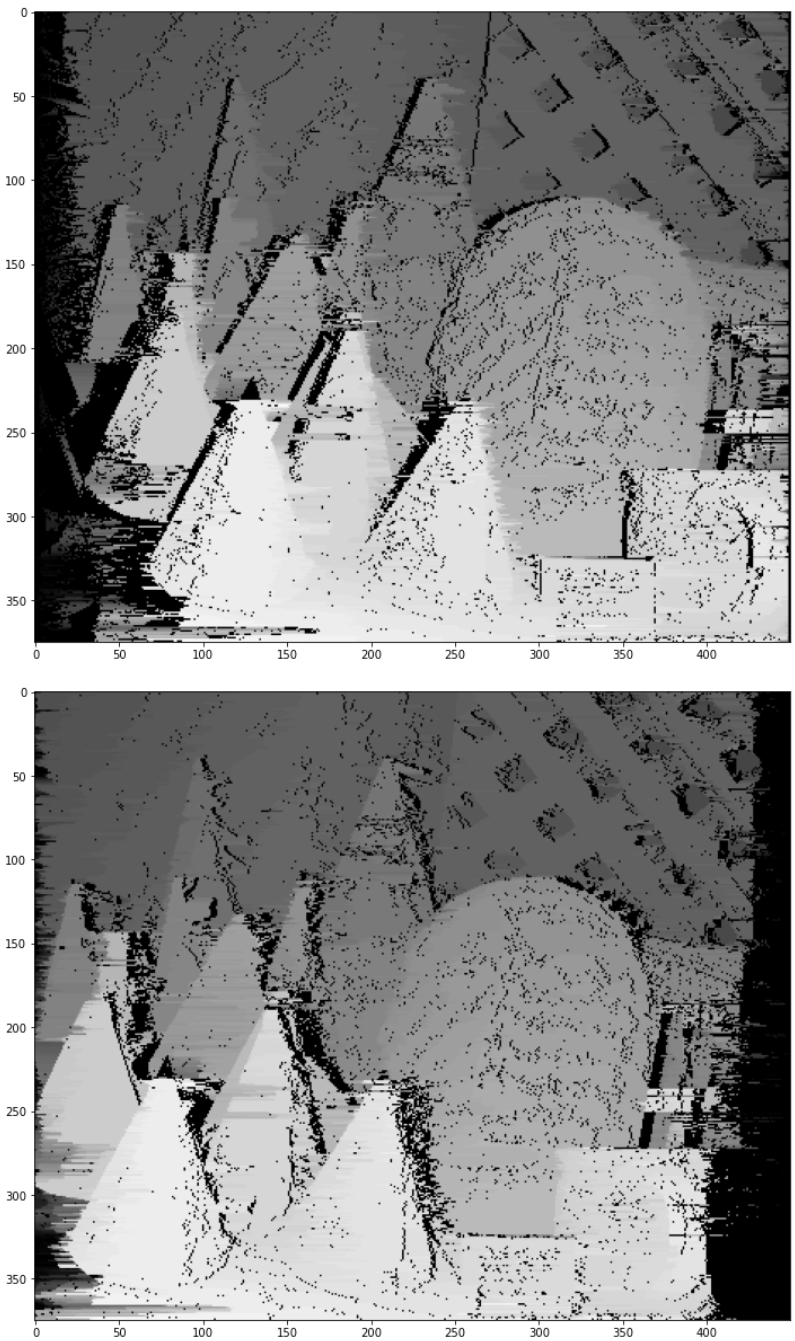

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

sigma = 2, c0 = 10

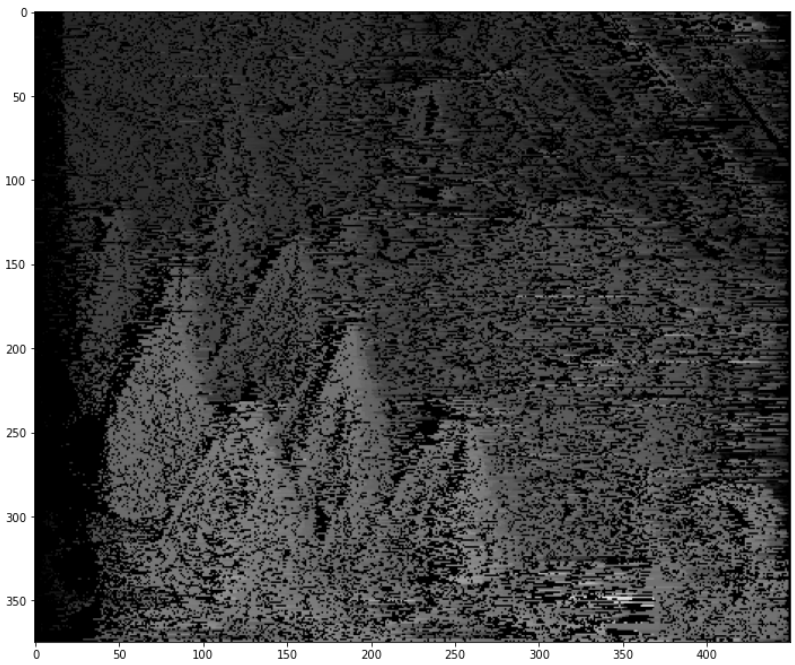

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

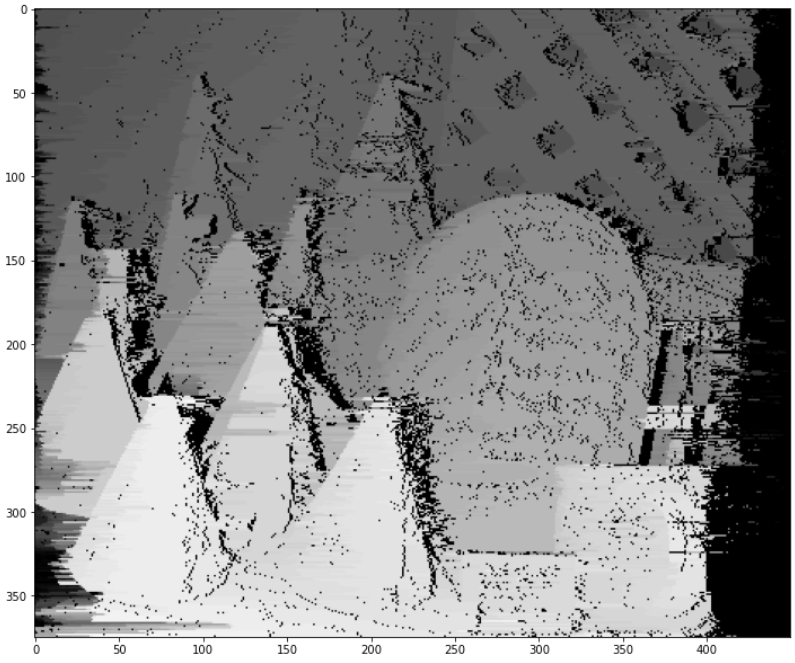

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

sigma = 4, c0 = 10

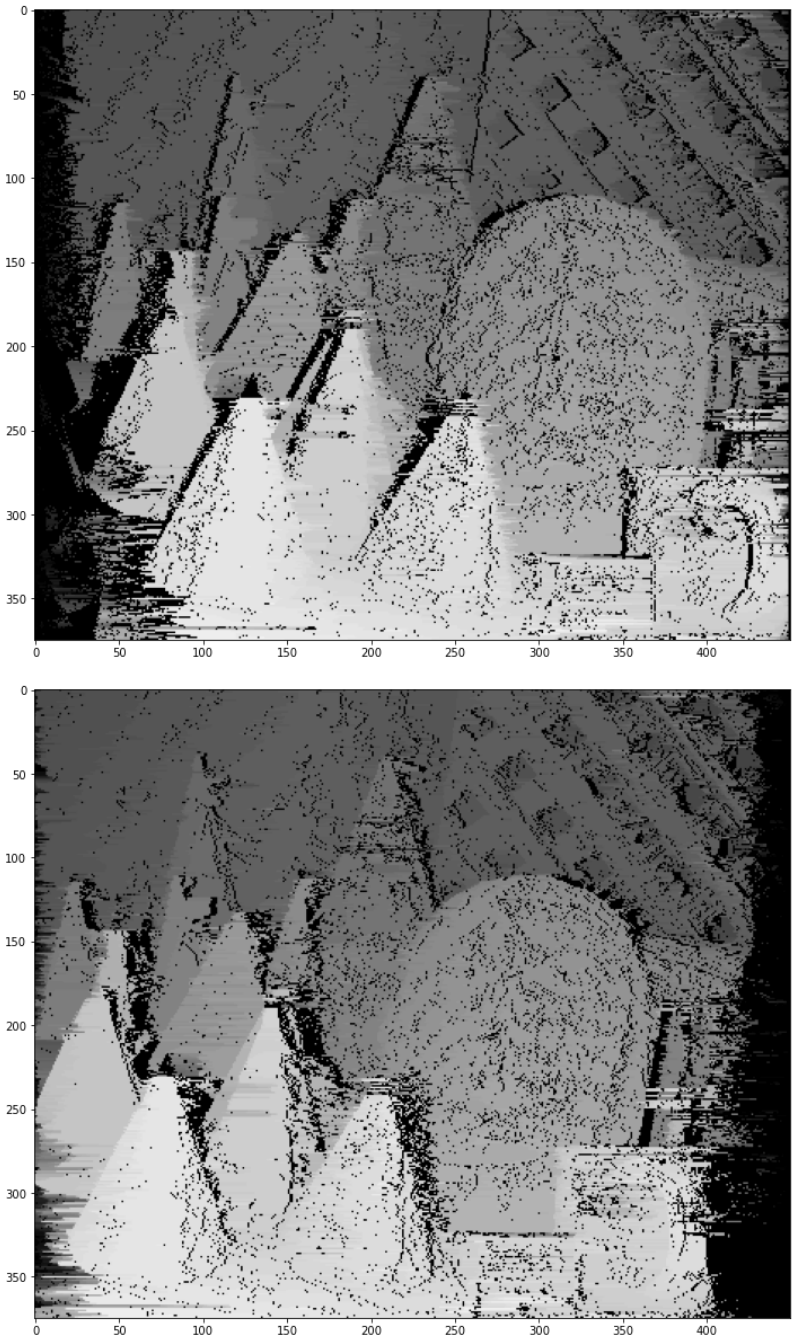


sigma = 0.5, c0 = 10

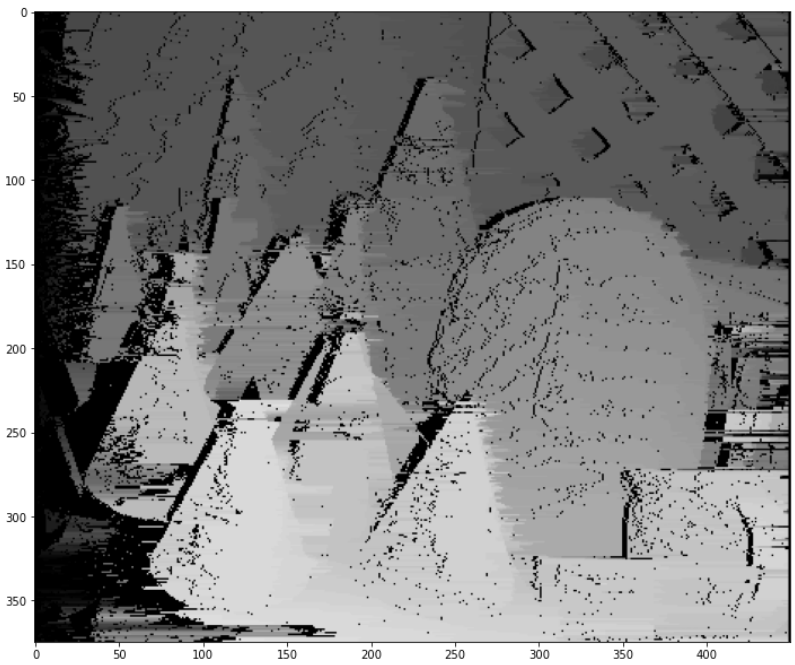

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

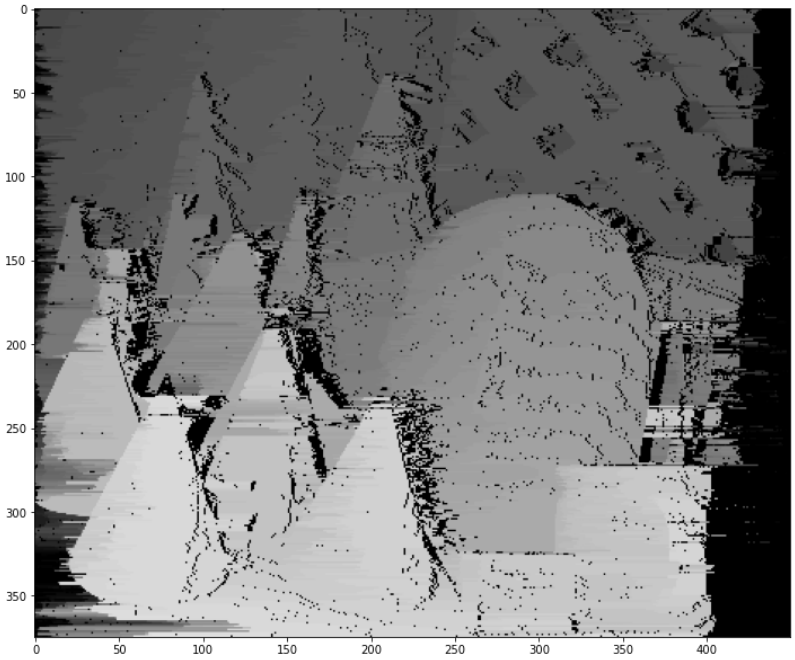

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

sigma = 2, c0 = 20

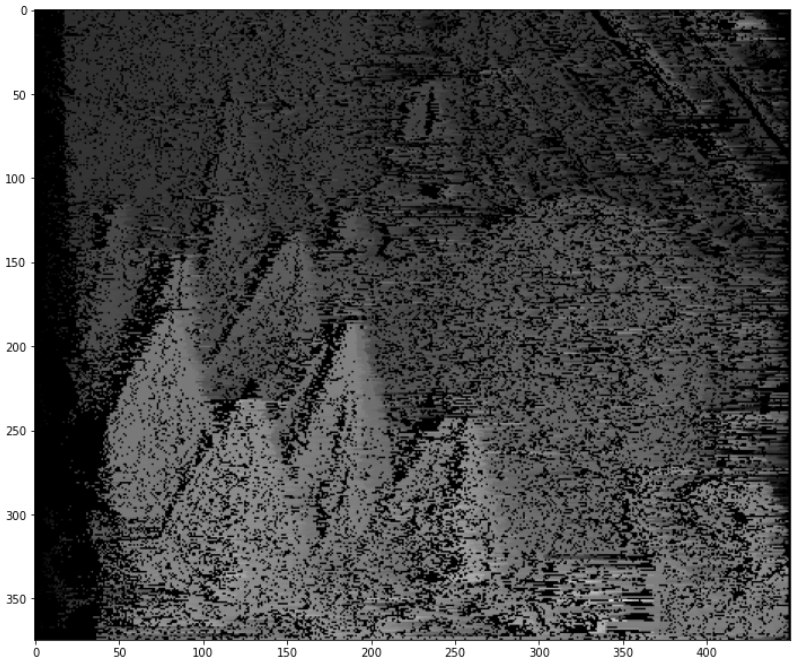


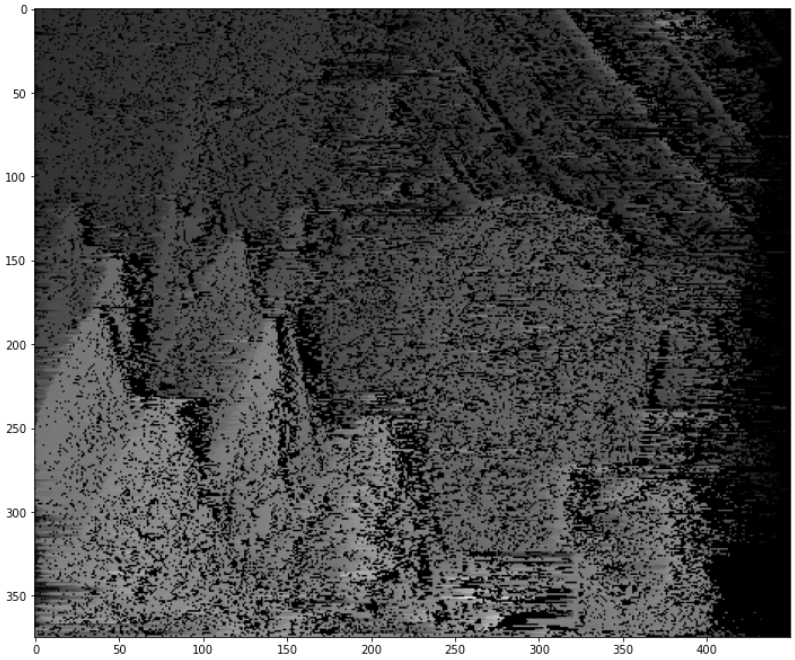
sigma = 4, c0 = 20



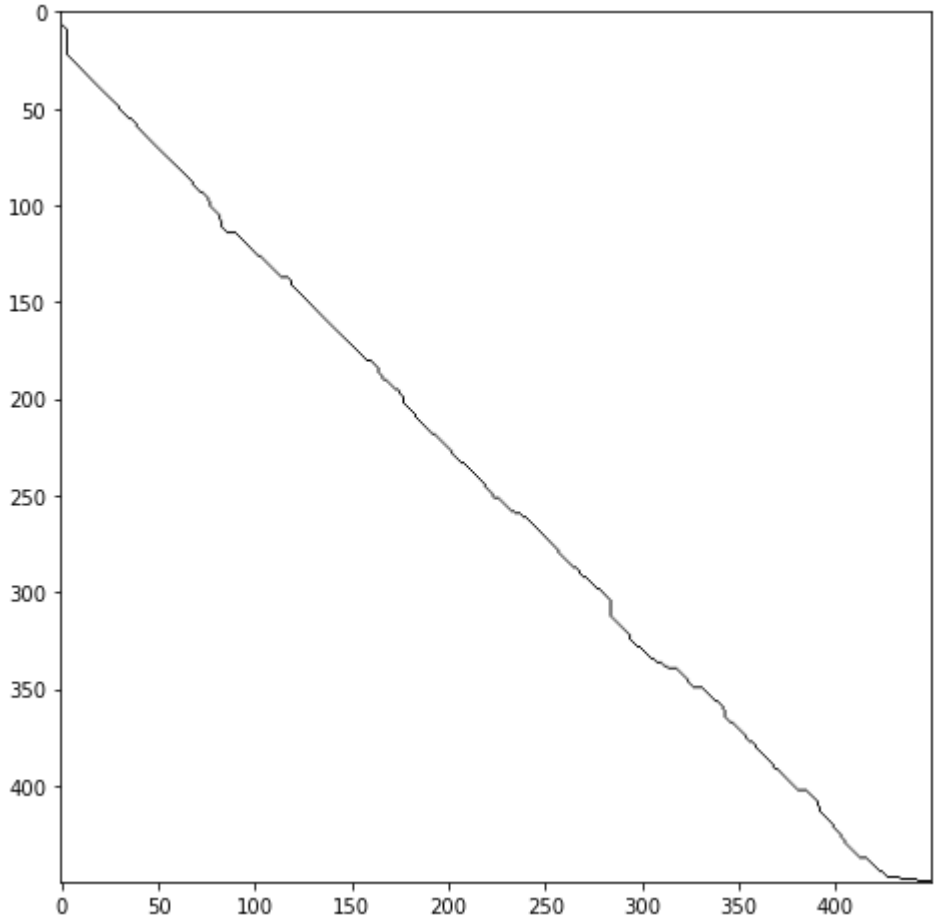


sigma = 0.5, c0 = 20


0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

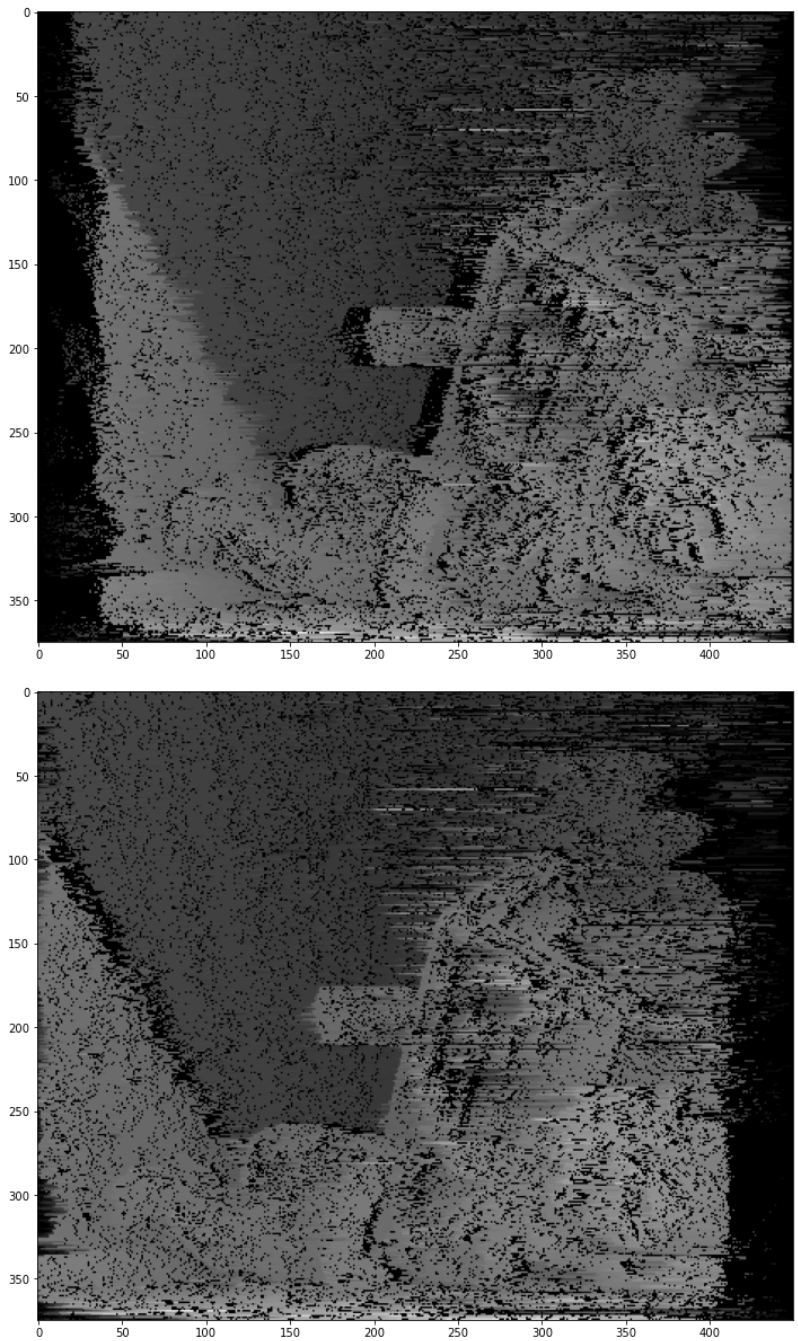

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

Bonus

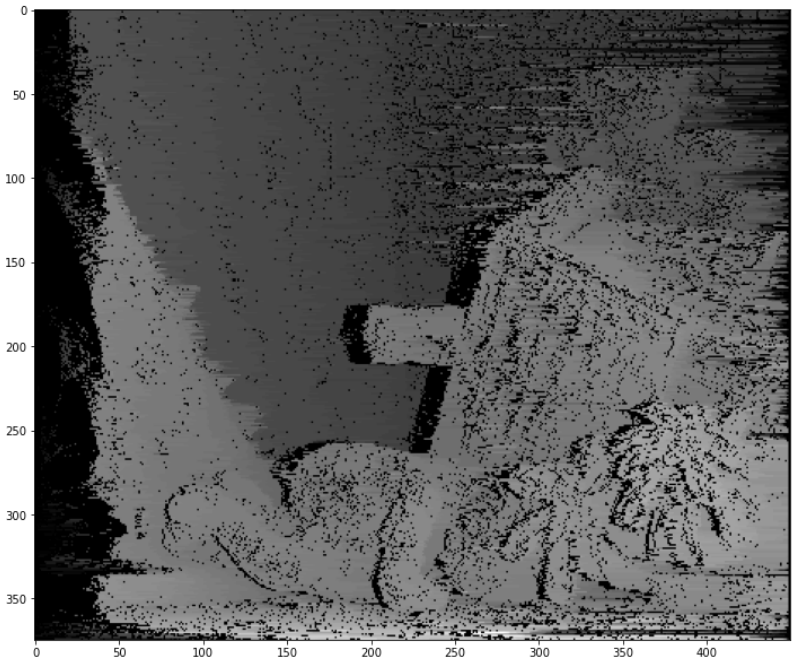


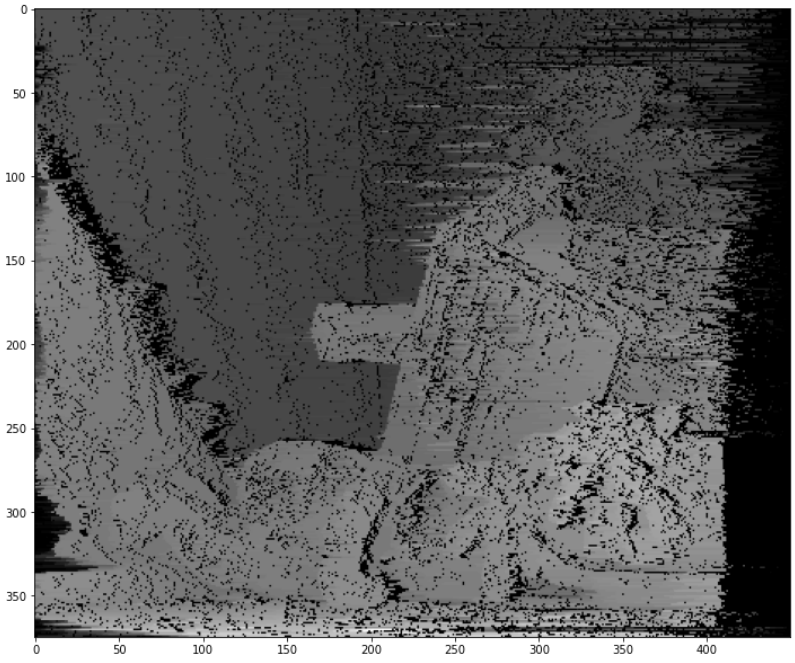
Third Image

sigma = 2, c0 = 1

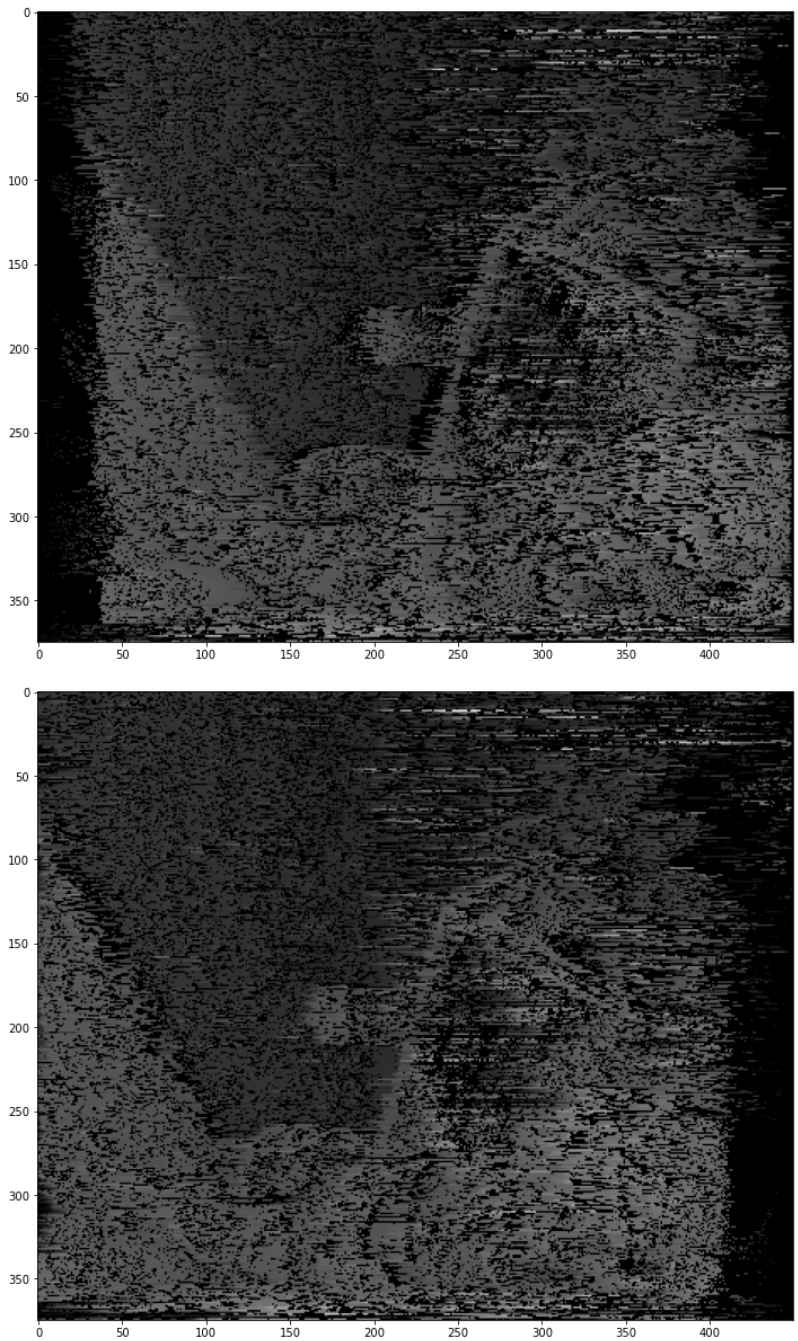


sigma = 4, c0 = 1

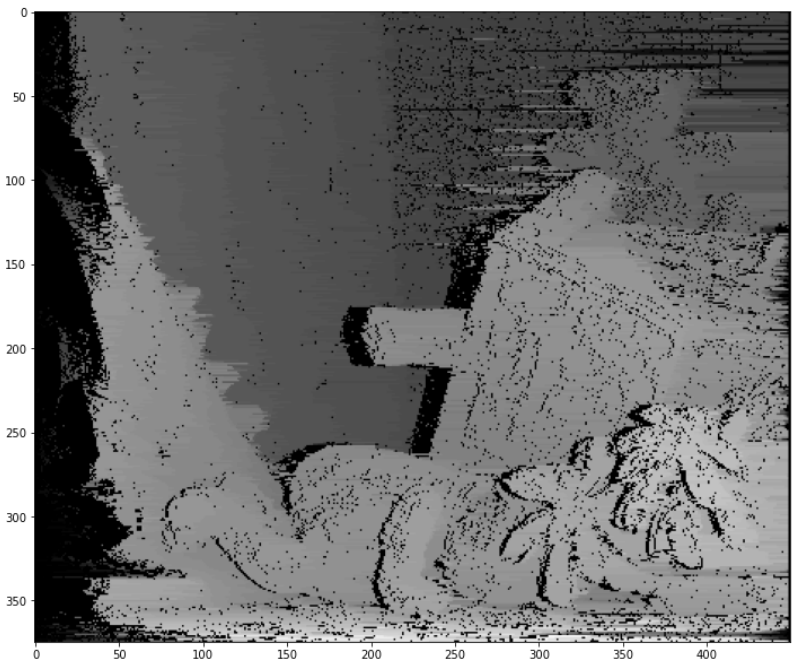

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

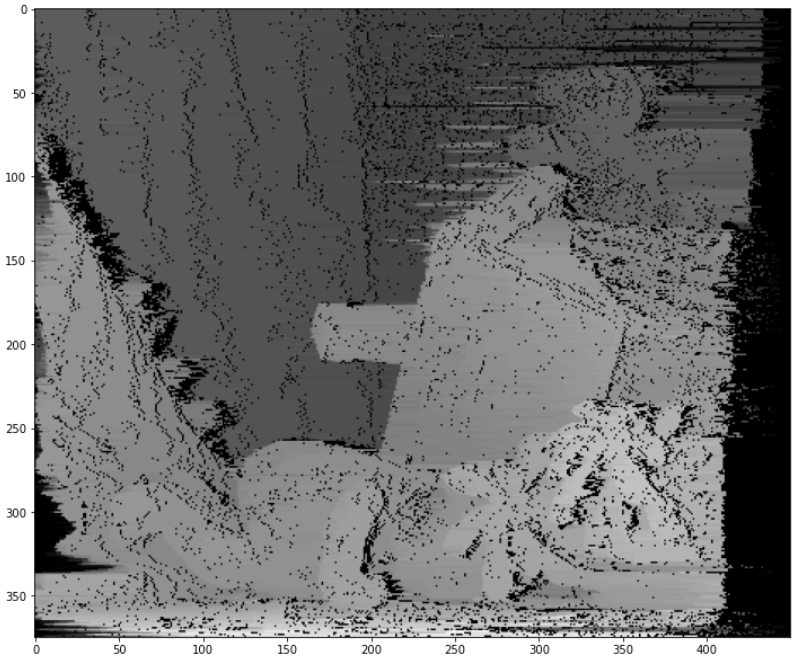

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

sigma = 0.5, c0 = 1

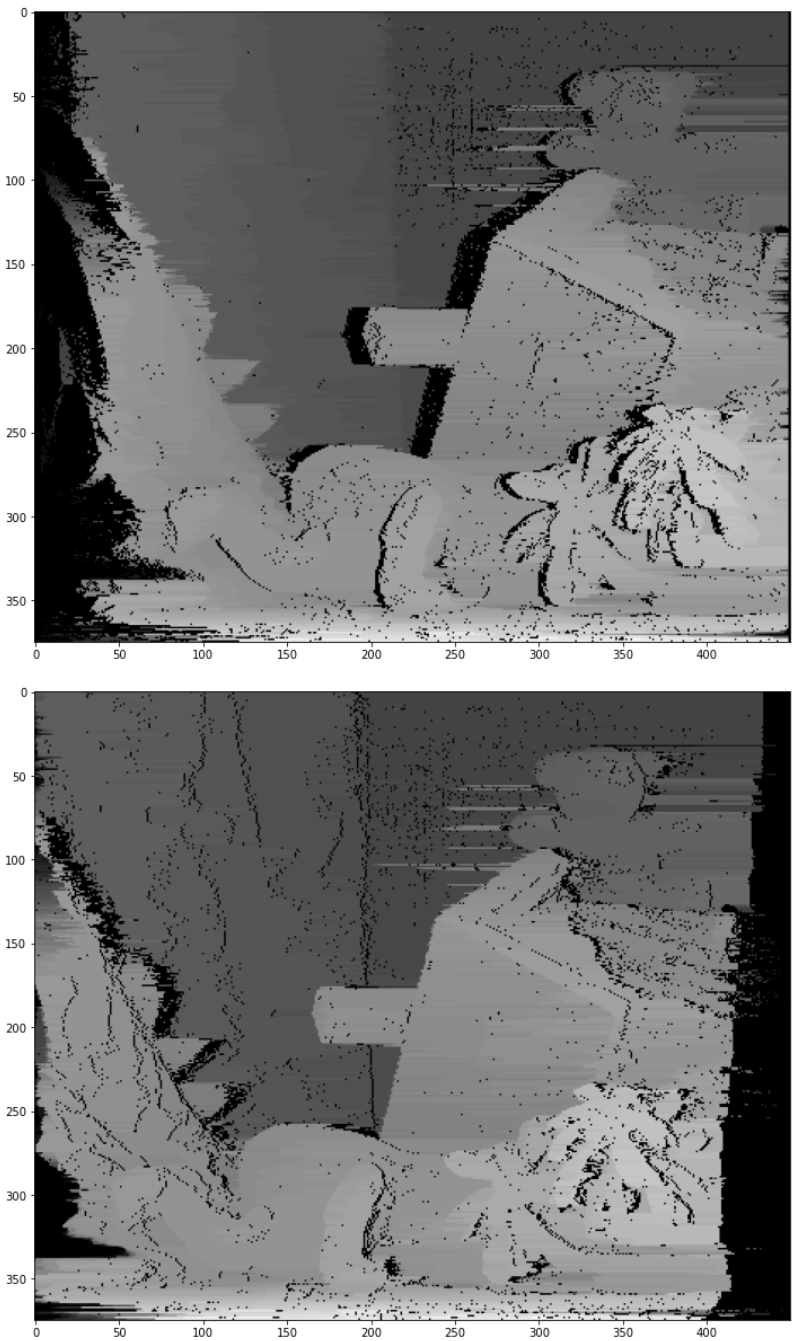

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400
0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

sigma = 2, c0 = 10


0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

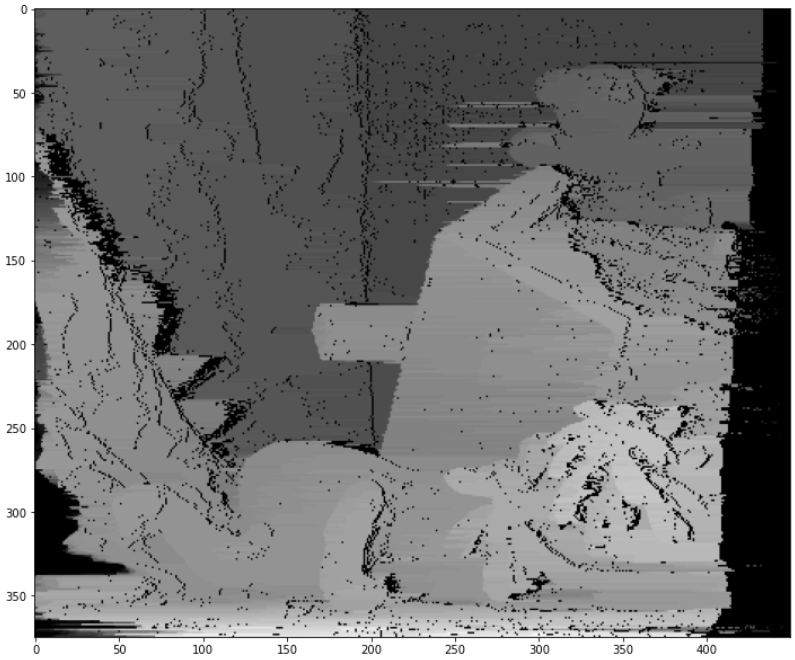

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

sigma = 4, c0 = 10

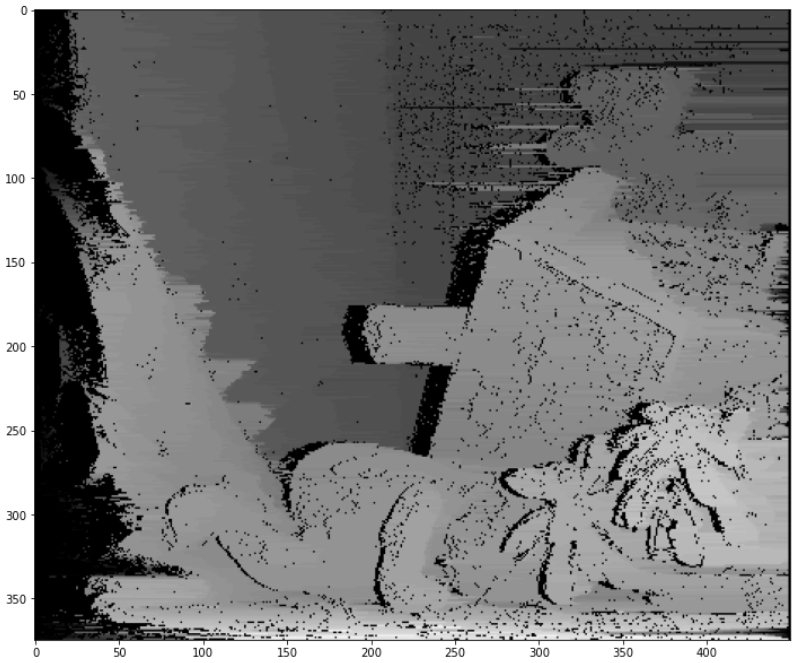


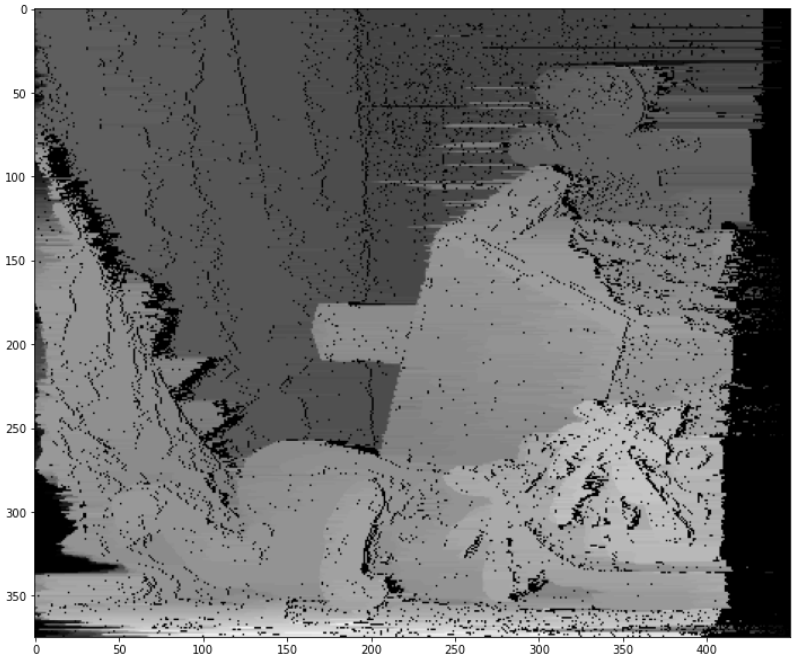
sigma = 0.5, c0 = 10


0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

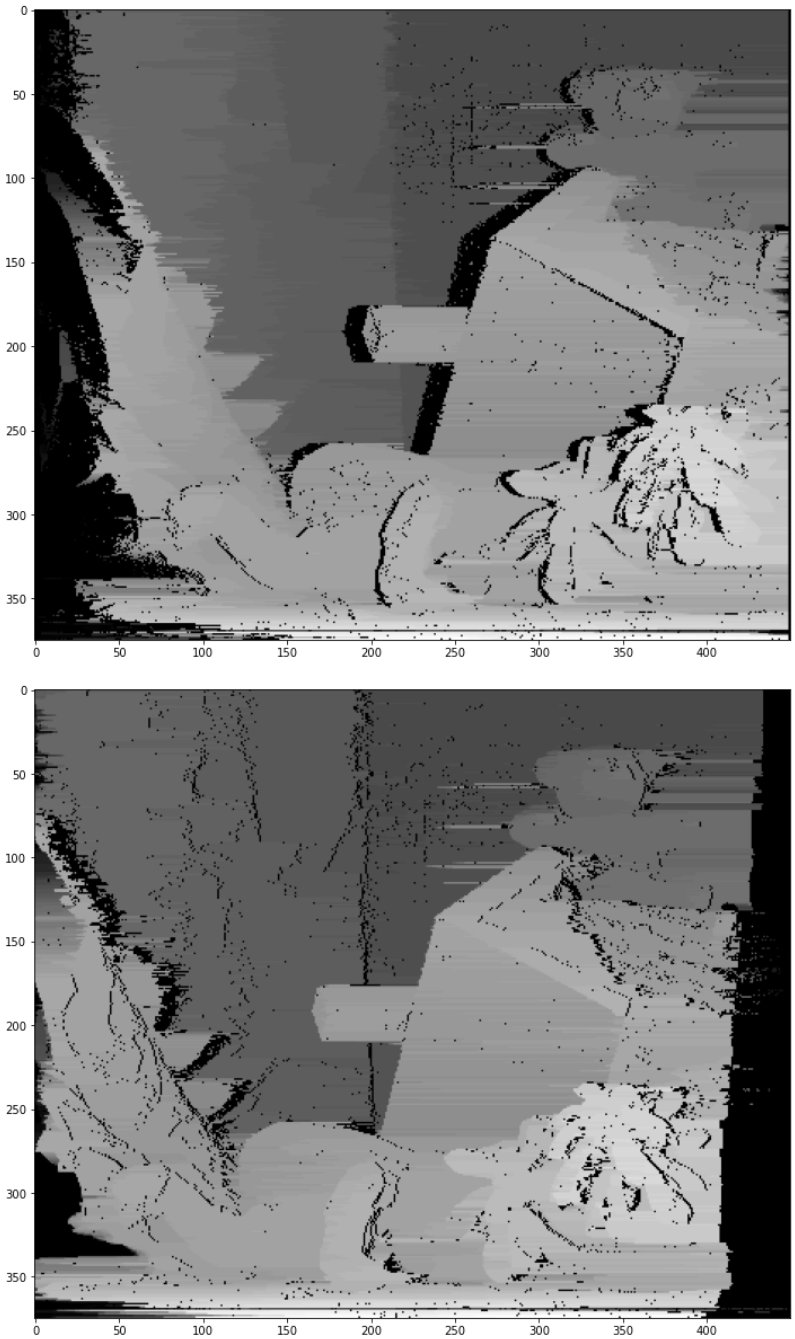

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

sigma = 2, c0 = 20

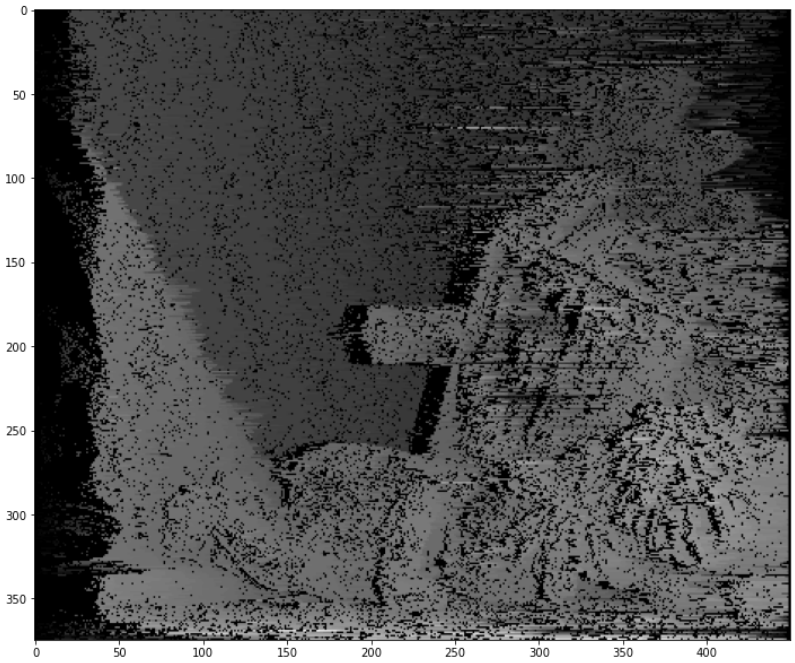

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

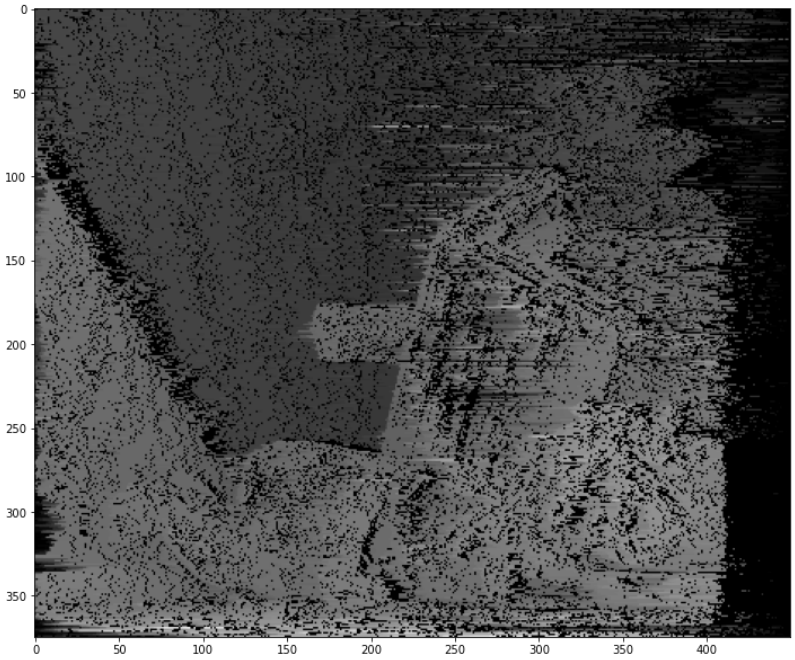

0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

sigma = 4, c0 = 20

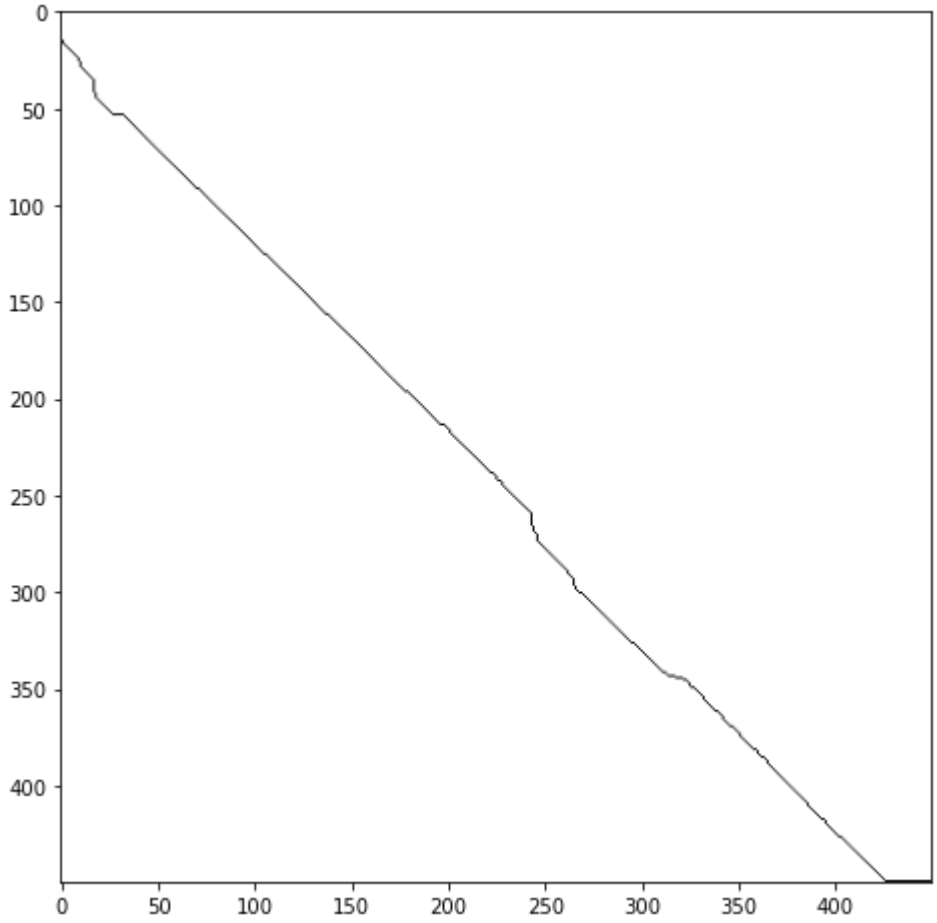


sigma = 0.5, c0 = 20


0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400


0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
350
400

Bonus


0
50
100
150
200
250
300
350
400
0
50
100
150
200
250
300
350
400
