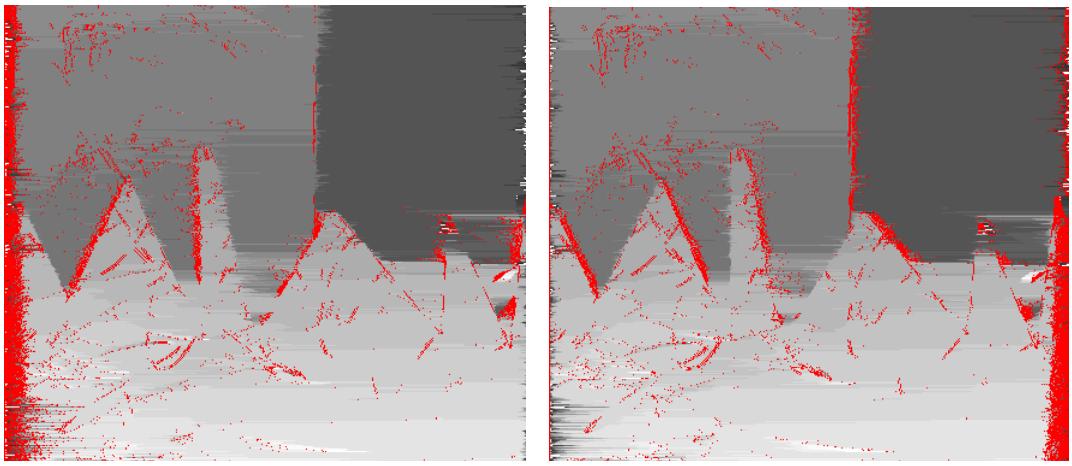
Algorithm cwk 2 by Zihan Zhu 19077928

Three pairs of stereo images:

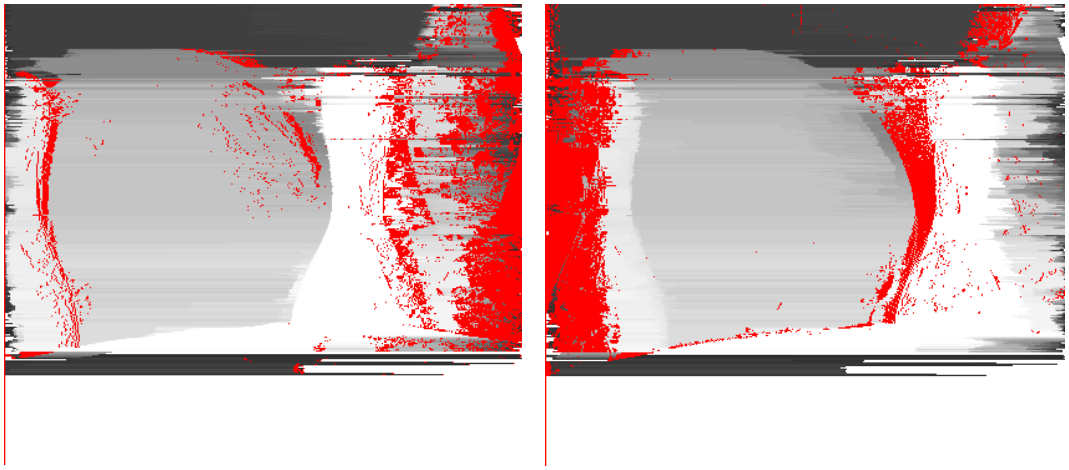
Red indicates occlusion.

The dimmer the color, the farther the object.

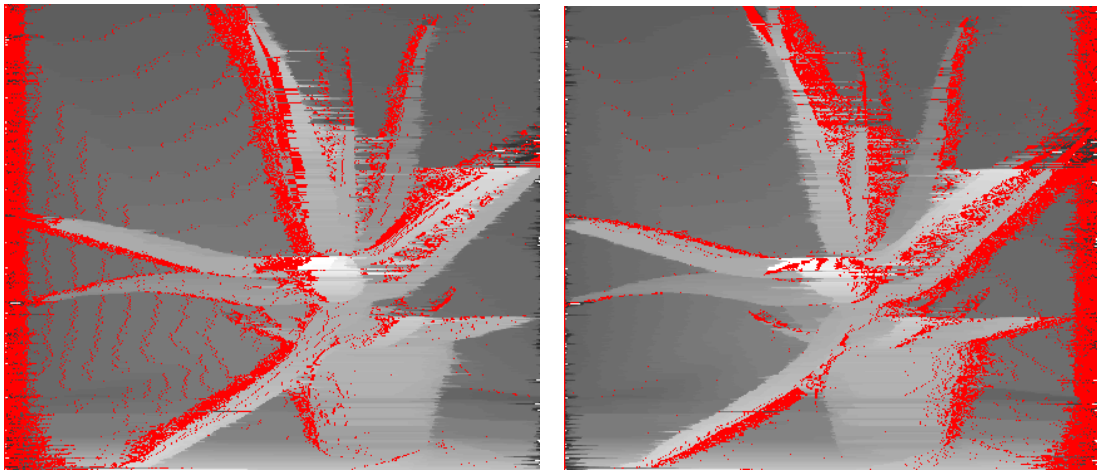
Pair1:



Pair2:

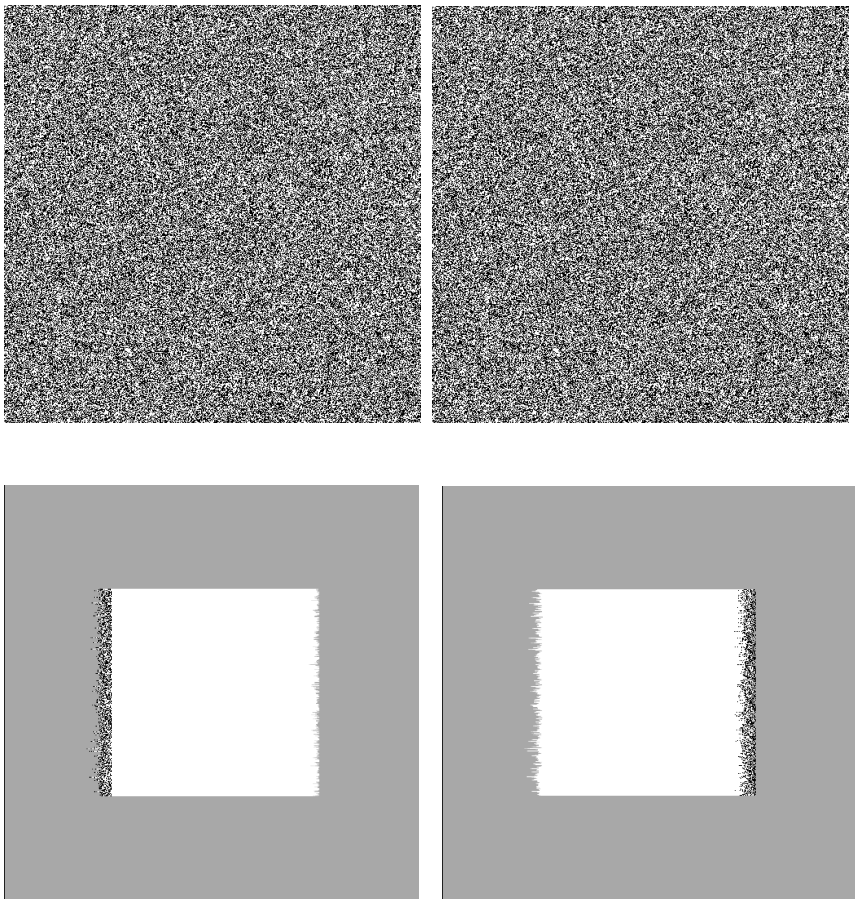


Pair3:



Random dot diagram:

Black dots indicate occlusion.



Discussion over the three questions:

- Why do matching errors occur for the binary random dot stereograms?

To obtain the depth image by dynamic programming we assume that each pixel on the left image has a unique match to a pixel on the right image. However, on a binary random dot stereogram, pixels are painted either black or white, so one pixel on one image has many identical pixels to match with on the other image. This leads to multiple global minima, and therefore uncertainties in constructing cost matrix and matching matrix, and therefore false matches when doing the backward pass. The error occurs because false matches between the images from multiple sensors camouflage the true matches. True matches are correspondences between image points that have the same generative source; false matches are correspondences between similar image points that have different sources.

- Investigate how the algorithm performs on other images as the occlusion cost is varied.

Occlusion cost indicates the likelihood of one pixel to be determined as occluded in the algorithm, i.e., the higher the cost, the lower the likelihood. The cost should be deliberately calibrated. If the occlusion cost is set too high, a string of bad matches will be selected as the lower-cost path; if it is too low, some pixels that should have been matched will be treated as occlusion.

- For string matching, what is the equivalent of occlusion?

Occlusion is the equivalent to the case in string matching that for the string (or substring) and the pattern (or sub-pattern) which are currently being studied, their last elements don't match.

Code:

```
from PIL import Image, ImageDraw
import numpy as np

def getwidth(im):
    return im.width

def getheight(im):
    return im.height

class matrix():
    def __init__(self, row, col):
        self.row = row
        self.col = col
        self.value = np.empty((col, row))

    def get_value(self, row, col):
        return self.value[col, row]

    def add(self, value, row, col):
        self.value[col, row] = value

    def get_size(self):
        return (self.row, self.col)


def get_pixel(path):
    im = Image.open(path)
    grey_im = im.convert('L')
    mat = matrix(getwidth(im), getheight(im))
    for j in range(getheight(im)):
        for i in range(getwidth(im)):
            mat.add(grey_im.getpixel((i, j)), i, j)
    return mat

def cost_of_matching(im1, im2, i, j, col):
    z1 = im1.get_value(i, col)
    z2 = im2.get_value(j, col)
    z = (z1 + z2) / 2
    w = (z - z1) * (z - z2) / 16
    return abs(w)
```

```
def cost_matrix(im1, im2, col):
    a = im1.get_size()[0]
    mat = matrix(a + 1, a + 1)
    mat1 = matrix(a + 1, a + 1)
    for i in range(1, a + 1):
        mat.add(i * 3.8, i, 0)
    for i in range(1, a + 1):
        mat.add(i * 3.8, 0, i)
    for i in range(1, a + 1):
        for j in range(1, a + 1):
            l = []
            a1 = mat.get_value(i - 1, j - 1) + cost_of_matching(im1, im2, i -
1, j - 1, col)
            b = mat.get_value(i, j - 1) + 3.8
            c = mat.get_value(i - 1, j) + 3.8
            l.append(a1)
            l.append(c)
            l.append(b)
            m = min(l)
            mat.add(m, i, j)
            mat1.add(l.index(m) + 1, i, j)

    return mat1

def match(mat):
    pairs = []
    leng = mat.get_size()[0]
    a = b = leng - 1
    while (a != 0 and b != 0):
        path = mat.get_value(a, b)
        if path == 1:
            pairs.append((a, b))
            a -= 1
            b -= 1
        elif path == 2:
            a -= 1
        elif path == 3:
            b -= 1

    row = []
    a = 0
    b = 0
    pairs.reverse()
    while (a < leng):
```

```
        if b >= len(pairs):
            row.append(-1)
            a += 1
        elif (a != pairs[b][0]):
            row.append(-1)
            a += 1
        else:
            row.append(abs(pairs[b][0] - pairs[b][1]))
            a += 1
            b += 1

    return row


def point_value(im1, im2):
    points = []
    for i in range(im1.get_size()[1]):
        points.append(match(cost_matrix(im1, im2, i)))

    return points

im1 = get_pixel(path of image)
im2 = get_pixel(path of image)

points = point_value(im1, im2)

for i in range(width):
    for j in range(height):
        a = points[j][i]
        if (a == -1):
            im.putpixel((i, j), pixel value for occlusion)
        else:
            im.putpixel((i, j), f(a))
```