Computer Vision Final Project
Depth Map Generation on More Realistic Scenes

A. Student IDs and names

組員一: 駱奕霖　　　F06943176

組員二: 陳柏劭　　　R07943129
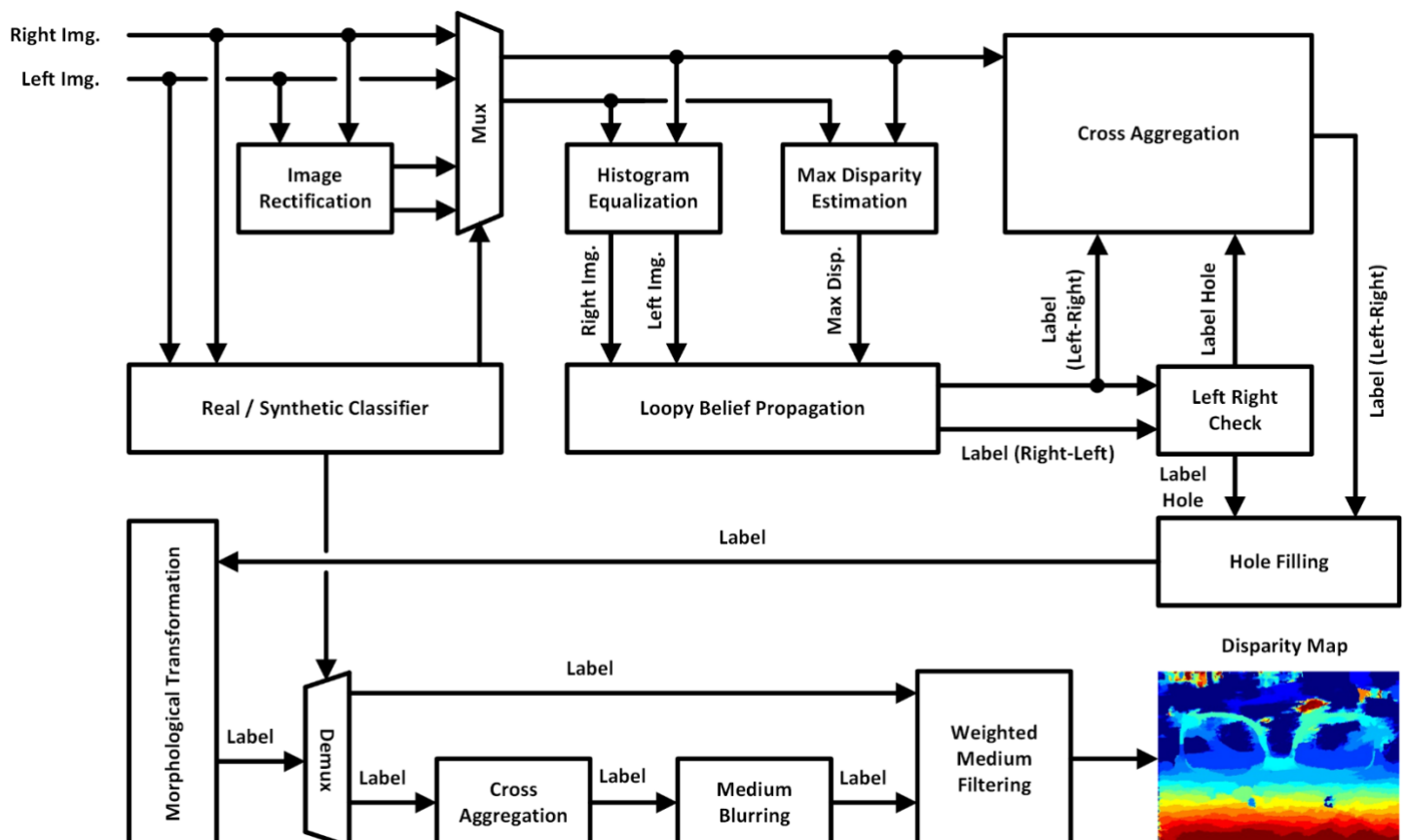
組員三: 楊仲萱　　　F07943023

組員四: 蔡宇軒　　　R07943171

B. Machine spec

CPU: Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz
RAM: 32 GB

C. Algorithm

Overview of algorithm

Synthetic/Real data classification

We distinguish the data according to the movement of the feature points in the image. We find that almost all of the objects of synthetic images move left. But some of the objects in real images have right moving directions. So we first find the feature points based on SIFT, and match these points by brute force match. Then, we minus the positions of these matching points. If the number of feature points whose moving direction are right are larger than threshold, we determine that this image is real image.

Image rectification

Because the real images do not rectify yet, we do image rectification first. In the first step, we find the feature points and match them just like what we've done in B.2. Once we have the feature points, we can find the transformation matrix and can do the image rectification. After rectification, the corresponding points of the left and right images will lie on the same epipolar lines. Therefore, we can find better disparity map. The images below shows the disparity maps of some real images w/ and w/o image rectification.
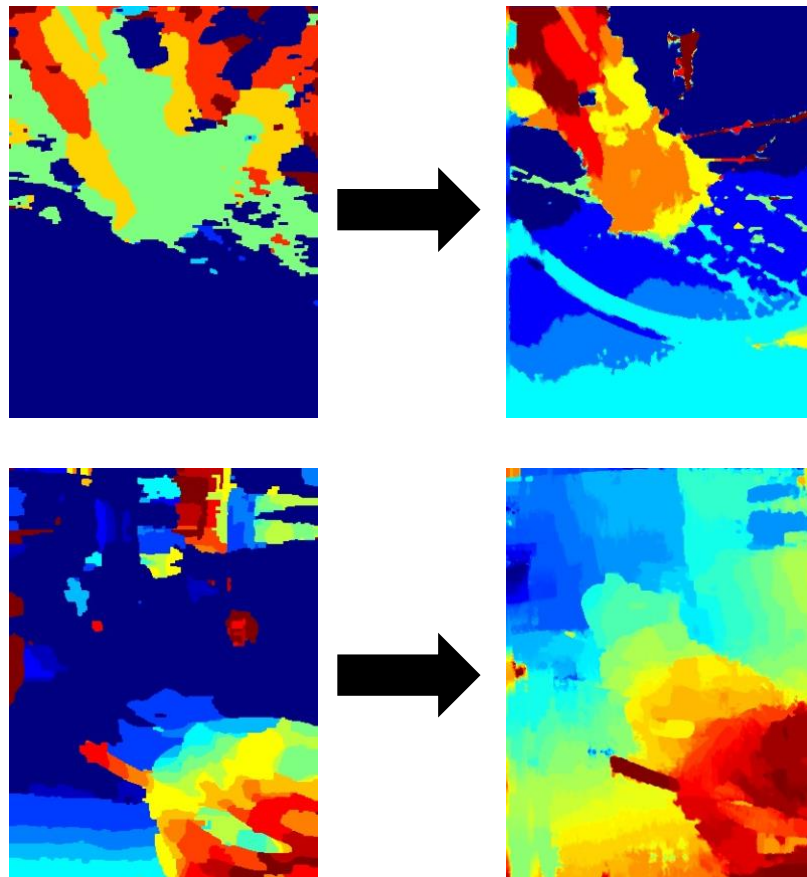
Histogram equalization

To make the costs differ greatly to enhance the stereo matching result, we modify the distribution of the pixel values to make the contrast of images greater.
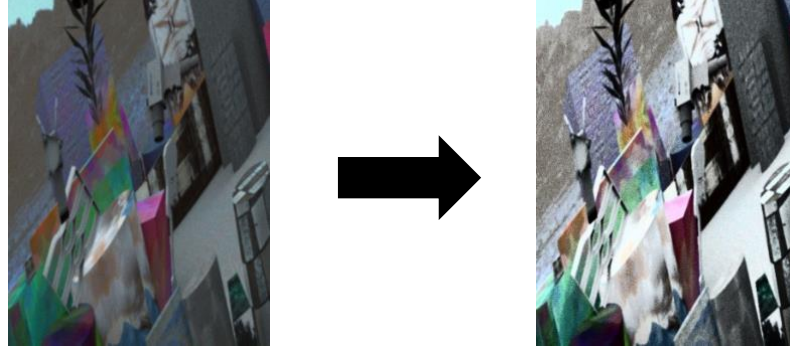
Max disparity estimation

In replacement of constant max disparity, we use SIFT and brute force matcher to estimate every group of images' max disparity. With the estimated max disparity, we could get better visualization of disparity maps.
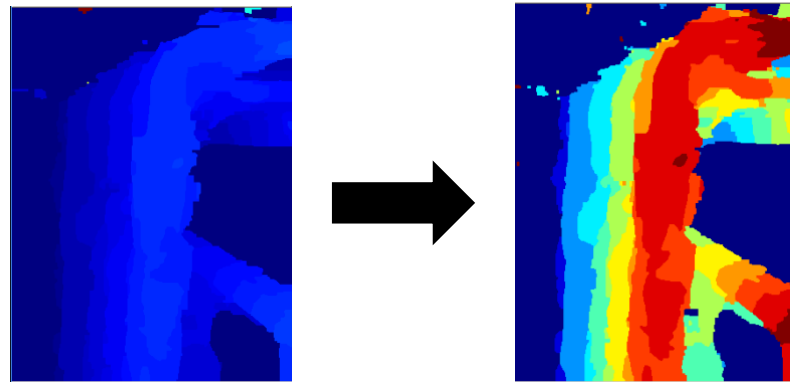
Weighted cost computation

The original cost computation uses the absolute distance (AD) of the pixel value in an area of the square window. To gain more information from the images in order to enhance the stereo matching result, we replace the AD cost with the weighted sum of the different costs, which are normalized census cost (C_{NCC}), AD of pixel value (C_{AD}), and the AD of gray scale image's gradients in x and y direction (C_{ADgx}, C_{ADgy}). To combined these costs, we have to normalize the census cost (C_{CC}). The normalized cost is calculated from C_{CC} with the following equation:

$$C_{NCC} = 1 - e^{-C_{CC}/55}$$

The weighted sum of different costs is referred to the following equation:

$$Cost = \alpha \cdot C_{NCC} + \beta \cdot C_{AD} + \gamma \cdot C_{ADgy} + (1 - \alpha - \beta - \gamma) \cdot C_{ADgx} \ , where \begin{cases} \alpha = 0.011 \\ \beta = 0.15 \\ \gamma = 0.1 \end{cases}$$

Since the computed costs are usually noisy, we use guided filter to do the cost aggregation, which can smooth the costs and enhance the disparity maps. After finishing computing the cost volume, the cost volume will be used by the loopy belief propagation.

Loopy belief propagation

When calculating the disparity map, we decided to use the global methods like loopy belief propagation and graph cut. Loopy belief propagation is selected to be our final method of constructing disparity map.

After exploiting both methods on the overall testing data, it turned out that the performance with loopy belief propagation is better. Furthermore, the reason why loopy belief propagation rather than original belief propagation is chosen is that the loopy belief propagation can deal with general graphs which contain loops.

The concept of loopy belief propagation (LBP) is to take some possible directions for passing messages into consideration making it more accurate than just using one direction and decreasing the energy gradually during iterations. The pseudo code of LBP is shown in Figure 1. At each iteration, every pixel will pass the message to four different directions (right, left, up, down) respectively. Once the iteration completes, the disparity which could minimize the calculated beliefs will be decided as the final disparity.

As for the details in our LBP algorithm, there are mainly four parts to accomplish the function which are updating message, message normalization, belief computation and disparity map construction. The steps in LBP algorithm are shown in Figure 2.

```
function LoopyBeliefPropgation
    initialise all messages

    for t iterations
        at every pixel pass messages right
        at every pixel pass messages left
        at every pixel pass messages up
        at every pixel pass messages down
    end for

    find the best label at every pixel i by
calculating belief
end
```
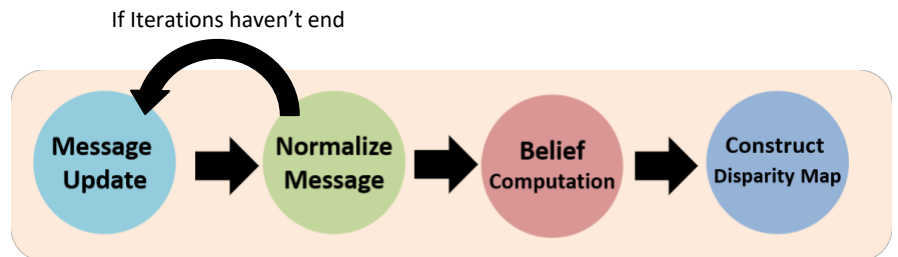
Figure. 1 Pseudo code of LBP algorithm

Figure. 2 Steps in LBP algorithm

When updating the message for different directions, we would first calculate the new values within all disparity values and then selected the smallest new value as its updated messages along all the axis of disparity. Take the up direction as example, we could find that the new values are calculated by summing the data cost and messages from other three directions in Figure 3. Finally, the updated message value is the smaller one between original calculated value and (*spqU*+lambda) which *spqU* is the minimum value of *npqU* along depth dimension.

```python
npqU = dataCost + msg_incoming_from_L + msg_incoming_from_D + msg_incoming_from_R
for lp in range(num_disp_values):
    msgU[:, :, lp] = np.minimum(npqU[:, :, lp], Lambda + spqU)
```

Figure. 3 Software code for passing message to up direction

In message normalization, we subtracted the mean value along depth dimension (disparity) from the updated message values. After the defined iteration ending, the value of belief at each pixel is calculated by summing data cost at this pixel and all the message values from its neighbors (up, down, right, left). In the last step of constructing disparity map, we would find the best disparity for each pixel having the minimum value of beliefs in the previous step and output the resulted disparity map.

Left right check

After loopy belief propagation, the disparity map of left image and right image are acquired. We use these disparity values to check whether the left image's and right image's disparity are same. Then, a boolean map that records which pixel's disparity value is invalid is generate. This map will be applied in the following hole filling step.

Hole filling

In this step, we apply the cross aggregation and the traditional hole filling technique. We use the cross aggregation first and the traditional hole filling technique later. It is noted that the original cross aggregation is very time-consuming since repetitively calculating window arms' range of each pixel and aggregating the values to the middle pixel cost. To avoid redundant calculation, we implemented the accelerated version of the cost aggregation [3]. The steps are shown in following:
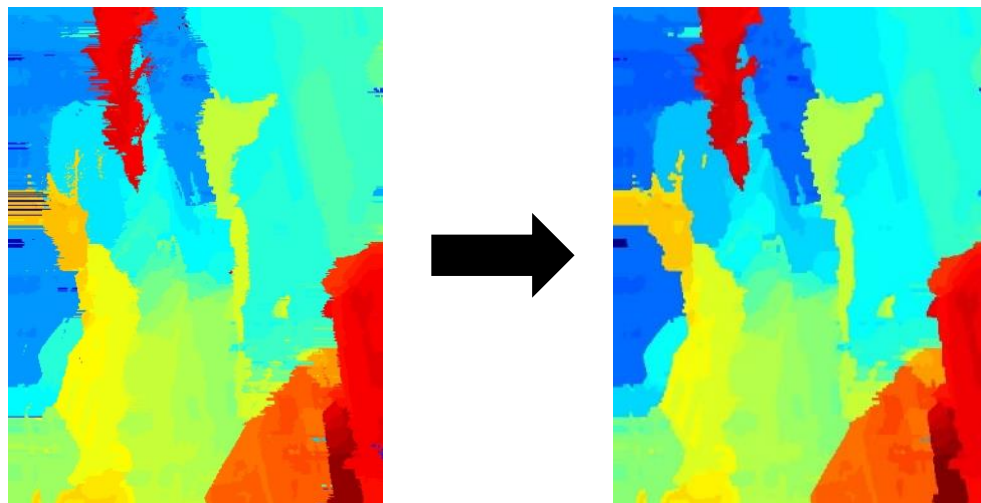
Step 1. Get each pixel's ranges of the horizontal arm and vertical arm by AD of pixel values.

Step 2. Calculate the cumulative row sum with the information acquired from step 1.

Step 3. Computing the difference between each pixel's horizontal arm's rightest value and leftest value from the cumulative row sum to get each pixel's horizontal sum.

Step 4. Calculate the cumulative column sum with the information acquired from step 1.

Step 5. Computing the difference between each pixel's vertical arm's bottom value and top value from the cumulative column sum to get each pixel's vertical sum.

Edge smoothing

We use morphological transformation to deal with jagged edges after the hole filling step. The followings are the original image and its different morphological transformation results:

a. Original b. Erosion c. Dilation d. Opening e. Closing
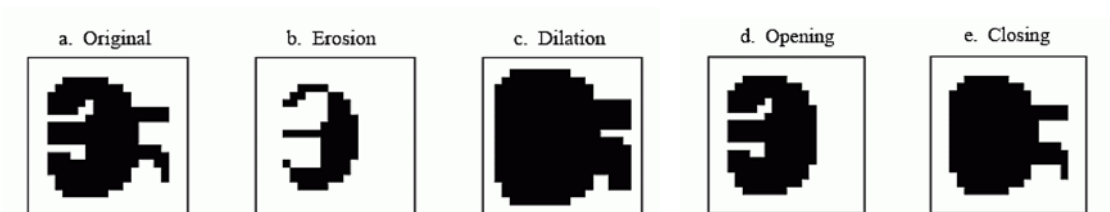
First, we use morphological opening to erase the convex parts of the jagged edges. Then, morphological closing is applied to fill up the concave parts of the jagged edges. After that, a weighted median filter is used for final edge smoothing.
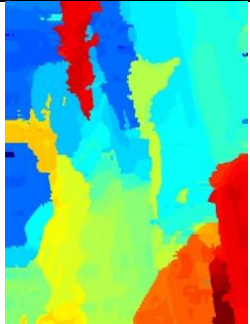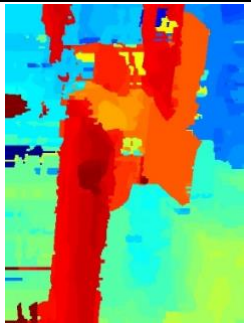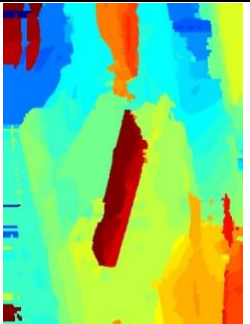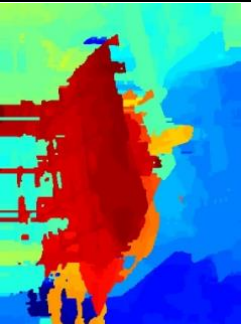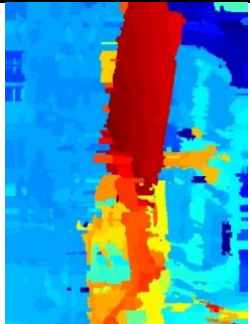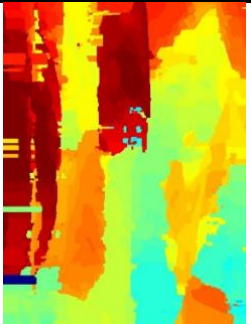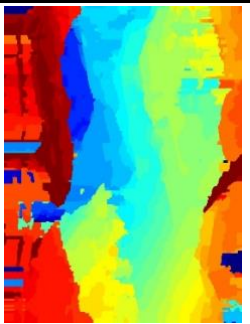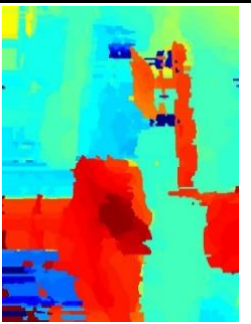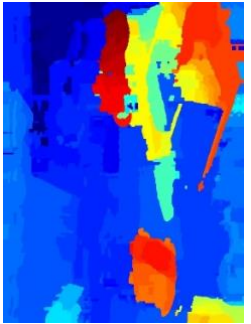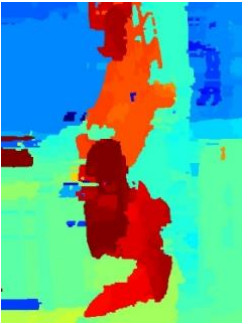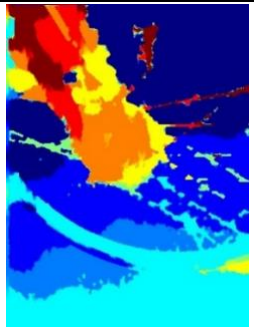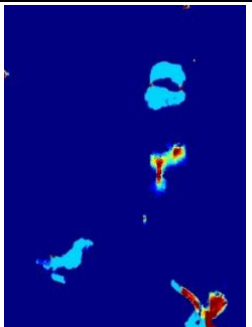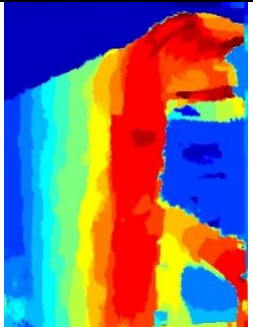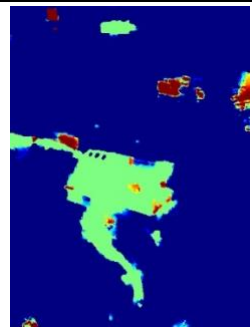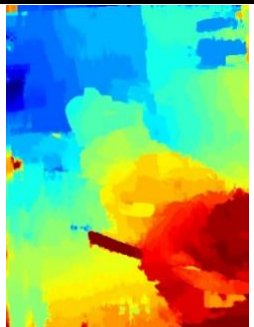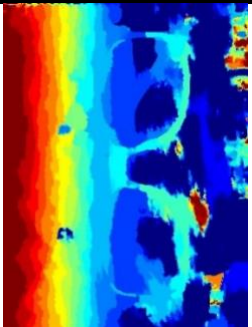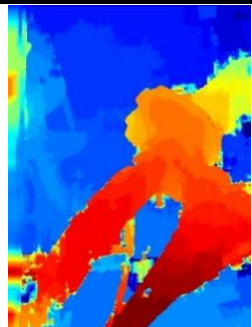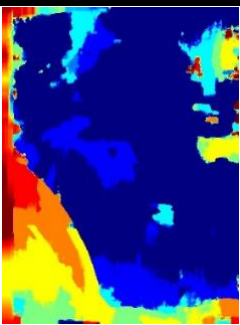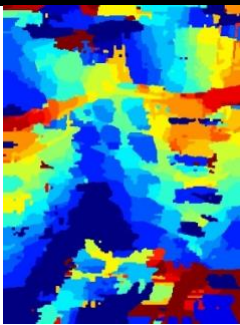
D. Results

Average error: 2.845056438446045

Average time: 186.3954990 (s)
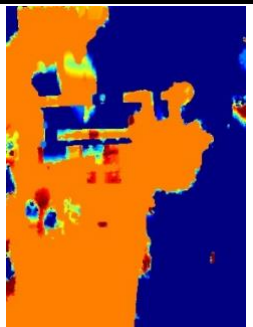
Synthetic Data			
Name	TL0	TL1	TL2	TL3
Error	1.98692608	2.63733625	1.91714478	3.84550357
Time (s)	185.967360	188.531249	147.119412	211.666707
Result				
Name	TL4	TL5	TL6	TL7
Error	3.16961193	2.49279261	2.69341207	4.78051281
Time (s)	204.984685	194.446175	133.433359	217.456425
Result				
Name	TL8		TL9	
Error	1.87624264		3.05108166	
Time (s)	169.319257		211.030361	
Result				

Average time: 56.4431055 (s)

Real Data			
Name	TL0	TL1	TL2	TL3
Time (s)	34.248015	33.191941	59.515657	32.182641
Result				
Name	TL4	TL5	TL6	TL7
Time (s)	99.116925	60.023015	36.018225	99.665310
Result				
Name	TL8		TL9	
Time (s)	43.127470		67.341856	
Result				

References

[1] C. Loop and Z. Zhang, "Computing rectifying homographies for stereo vision," Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)

[2] http://nghiaho.com/?page_id=1366

[3] K. Zhang, J. Lu, and G. Lafruit, "Cross-Based Local Stereo Matching Using Orthogonal Integral Images," IEEE Transactions on Circuits and Systems for Video Technology, vol. 19, no. 7, pp. 1073–1079, 2009.