# Disparity Map

*Li Kaiyu (S220048)*
*Chen Yueqi (G2202490B)*
*Chang Lo-Wei (G2203324E)*

Assignment 02
Computer Vision
School of Computer Science and Engineering
Nanyang Technological University

2022

# Table of Contents


**1 Introduction of Disparity** **1**

**2 Procedure of Disparity Map** **3**
2.1 Disparity Map Algorithm - Block Matching . . . . . 3
2.2 Discussion of Cost Function . . . . . 4
2.3 Discussion of Block Length and Search Width . . . . . 5

**3 Observations and Improvements** **6**
3.1 Pre-processing . . . . . 6
3.2 Post-processing . . . . . 7
3.2.1 Left-Right Consistency Test . . . . . 7
3.2.2 Uniqueness Ratio Test . . . . . 9

**4 Advanced Approaches** **10**
4.1 Spatial Regularization . . . . . 10
4.1.1 Global Matching - Stereo MRF . . . . . 10
4.1.2 Semi-Global Block Matching - SGM/SGBM . . . . . 11
4.2 Deep Learning Approaches . . . . . 13
4.2.1 Bench Mark . . . . . 13
4.2.2 PSMNet . . . . . 13

**Bibliography** **15**

# Chapter 1

# Introduction of Disparity

If you alternate between using your left and right eyes while holding your index finger close to your eyes. You will notice that your finger jumps a lot in your perspective when compared to other distant objects. This example provides two hints: (1) There are differences in the observation of an object in the real world between the left eye and the right eye; (2) the distance between an object and eyes will influence the extent of difference. Our brain uses such differences to enable us to recognize the distance of the object. Inspired by binocular vision, taking pictures of the same scene from two different positions will contribute to computer stereo vision.

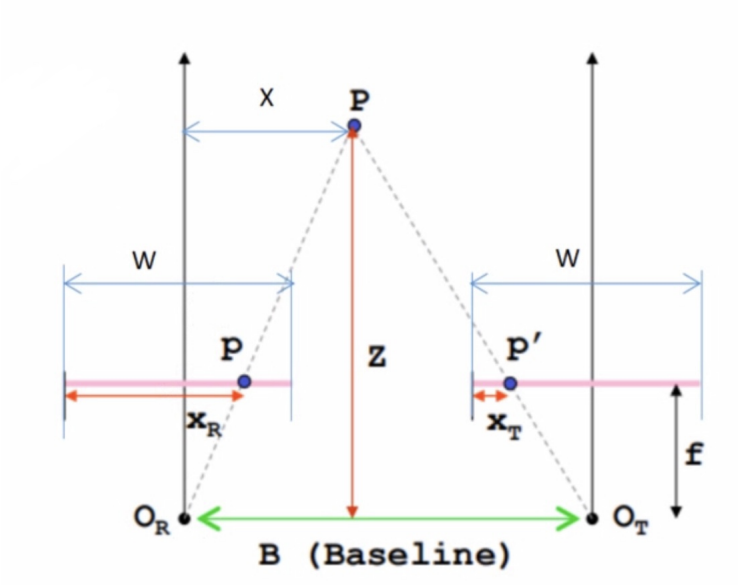



Figure 1.1: simplified stereo setup diagram

Figure 1.1 shows the cameras arrangement from above. In this diagram:

- $O_R$ and $O_T$ are two camera centers and $B$ is the distance between them;
- $f$ is the focal length
- $w$ is the length of the sensor plane;

- $P$ is the point which we want to estimate its distance ($Z$) to the camera baseline;
- $p$ and $p'$ are the projections of $P$ on two sensor planes;
- $X_R$ and $X_T$ are the respective x-coordinates of $p$ and $p'$.

That is, we are interested in estimating the unknown variable $Z$ with known variables $X_T$, $X_R$, $f$, $B$, and $w$. Then we can derive equation (1.1) by using similar triangles $\triangle Ppp' \sim \triangle PO_RO_T$:

$$\begin{aligned} \frac{B}{Z} &= \frac{pp'}{Z-f} \\ &= \frac{B-(X_R-\frac{w}{2})-(\frac{w}{2}-X_T)}{Z-f} \\ &= \frac{B+X_T-X_R}{Z-f} \end{aligned} \tag{1.1}$$

Solving equation (1.1) for $Z$, we can obtain equation (1.2) and define $d = X_R - X_T$ is the disparity. Since all the variables on the right-hand side of the equation are known, $Z$ is solvable. This equation also proves that disparity is inversely proportional to distance. In other words, larger disparity results in shorter depth and vice versa.

$$Z = \frac{B \cdot f}{d}, \text{ where } d = X_R - X_T \tag{1.2}$$

# Chapter 2

# Procedure of Disparity Map

## 2.1 Disparity Map Algorithm - Block Matching

Given a pair of rectified images of the same scene captured from two different viewpoints, *imgL* and *imgR*. We will first set two configurable parameters: **BlockLength** and **SearchWidth**.

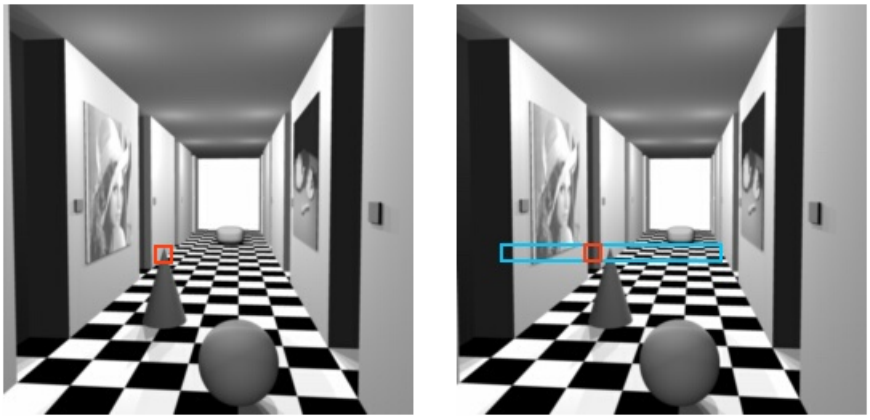


Figure 2.1: left image (*imgL*): corridor from left viewpoint; right image (*imgR*): corridor from right view point

1. For each pixel (x, y) in *imgL*, find a square neighborhood around this pixel with side length of **BlockLength** and denote this square neighborhood as *blockL*. The red square shown in the left image in Figure 2.1 visualizes one of the possible *blockL*;
2. Set a rectangular area along the same scan line of (x, y) in *imgL* with width of **SearchWidth** and height of **BlockLength** as *SearchArea* in *imgR*, which is shown as the blue rectangle in the right image in Figure 2.1;
3. Move *blockR* along *SearchArea* by one pixel at each time. Here, the red square in the right image in Figure 2.1 represents one of the possible *blockR*;

4. Calculate the cost function between each *blockR* and *blockL*, then the *blockR* with smallest value of cost function will be considered as the best matching block;
5. The center pixel (x', y') in the best matching *blockR* will be matched with (x, y) in *imgL*;
6. Then $x' - x$ will be the disparity value at pixel (x, y).
7. Repeat above steps for each pixel in *imgL* and get the best disparity values for all pixels and this is known as the winner-takes-all (WTA) strategy.
8. Visualize the best disparity values of all pixels in one figure will give the corresponding disparity map.

## 2.2 Discussion of Cost Function

Cost function is used for comparing the similarity between *blockL* and *blockR* with side length **BlockLength** (denoted as BL in the following functions). Since there are two types of frequently used cost functions, which are **sum-of-absolute difference** (SAD) and **sum-of-squared-difference** (SSD). In this section, we will discuss these two cost functions and show the reasons why we decided to use **sum-of-absolute difference** (SAD) in our algorithm mentioned in the step 4 in Chapter 2.1.

- **sum-of-absolute-difference** (SAD) computes the sum of elementwise absolute differences of two blocks (*blockL* and *blockR*). Then the function would be:

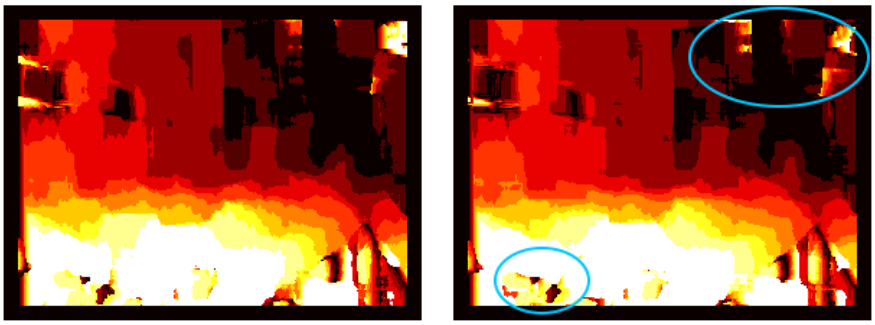
$$SAD(blockL, blockR) = \sum_{i=1}^{BL} \sum_{j=1}^{BL} |blockL[i][j] - blockR[i][j]|$$

- **sum-of-squared-difference** (SSD) computes the sum of elementwise squared differences of two blocks (*blockL* and *blockR*). Then the function would be:

$$SSD(blockL, blockR) = \sum_{i=1}^{BL} \sum_{j=1}^{BL} (blockL[i][j] - blockR[i][j])^2$$

Two equations above indicate SAD is better due to two reasons:

1. SAD is more robust to noises/outliers compared to SSD. For the same input, the square function always returns a non-smaller value than the absolute function does. Therefore, when noises or outliers are encountered, square function will magnify such errors. The blue circles in the right image of Figure 2.2 proves the magnified noises using SSD, which are not shown in the left image using SAD.
2. SAD is faster than SSD. This reason is straightforward that computing absolute function is faster than computing square function and this fact becomes more apparent if the input value is big. Table 2.1 shows the average time of using SAD and SSD to produce 6 disparity maps of each image with different parameter settings and it is shown that using SAD would be approximately 10 seconds faster than using SSD.



f

Figure 2.2: Left image: using SAD; Right image: using SSD

| | corridor.jpg | triclopsi2.jpg |
|---|---|---|
| Average time using SAD (s) | 47.96 | 53.83 |
| Average time using SSD (s) | 57.80 | 64.93 |

Table 2.1: Average time of using SAD and SSD to produce 6 disparity maps with different parameter settings on corridor.jpg and triclopsi2.jpg.

## 2.3 Discussion of Block Length and Search Width

As we mentioned in chapter 1.2, **SearchWidth** and **BlockLength** are configurable. To explore how these two parameters will influence the look of disparity maps, we tried combinations of different values on **SearchWidth** and **BlockLength**.

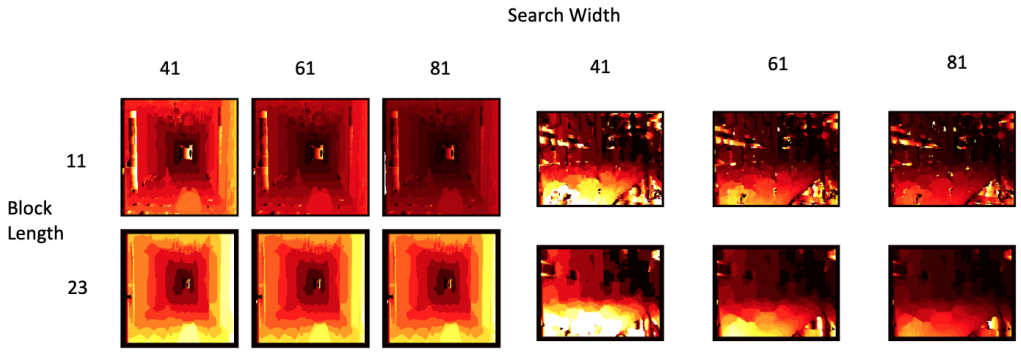



Figure 2.3: Disparity maps under different parameter settings

The disparity maps shown in Figure 2.3 resulting from different **SearchWidth** and **BlockLength**. Two observations here: (1) the results in second row is smoother than those in first row; (2) the results in the second columns with **SearchWidth** of 61 or 81 is more accurate than those of 41 **SearchWidth**. Based on these two observations, we can derive two important conclusions about parameter settings:

- larger **BlockLength** indicates a larger neighborhood around the choosen pixel and produces smoother disparity maps;
- larger **SearchWidth** indicates a larger searching area to find best matching block and produces more accurate disparity maps.

# Chapter 3

# Observations and Improvements

## 3.1 Pre-processing

Appearance-based matching detects points with similar appearances in two images and then calculates their disparity, which requires an assumption that two images have almost identical image patches. This requirement is met in both sample images since the stereo cameras have small baselines. However, the SSD-based search is not robust. Another constraint is that the matching results will be dramatically influenced by the complex transition light. Therefore, we introduce Sobel differential operator, which can enhance contrast and detect image changes.

```
def sobelOperator(img):
    sobelXY = np.zeros(img.shape)
    sobelX = np.zeros(img.shape)
    sobelY = np.zeros(img.shape)

    height, width = img.shape
    horizontalMask = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
    verticalMask = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])

    for i in range(height-2):
        for j in range(width-2):
            sobelX[i+1, j+1] = abs(np.sum(img[i:i+3, j:j+3] * horizontalMask))
            sobelY[i+1, j+1] = abs(np.sum(img[i:i+3, j:j+3] * verticalMask))
            sobelXY[i+1, j+1] = (sobelX[i+1, j+1] * sobelX[i+1,j+1] + \
                                 sobelY[i+1, j+1] * sobelY[i+1,j+1]) ** 0.5

    return sobelX, sobelY, sobelXY
```

Figure 3.1: Sobel Operator Implementation

Sobel gradient filtering compute the image gradient via two Sobel masks, which are in horizontal and vertical directions respectively. The code implementation and results are shown in Fig 3.1 and Fig 3.2.

Other than choosing Sobel filtering as the pre-processing, we can used other filters which are also capable to enhance the texture information so that improve the performances of the appearance-based matching. In Fig 3.3, we compare the final disparity maps using Sobel operator with ones using normalization as pre-filtering.

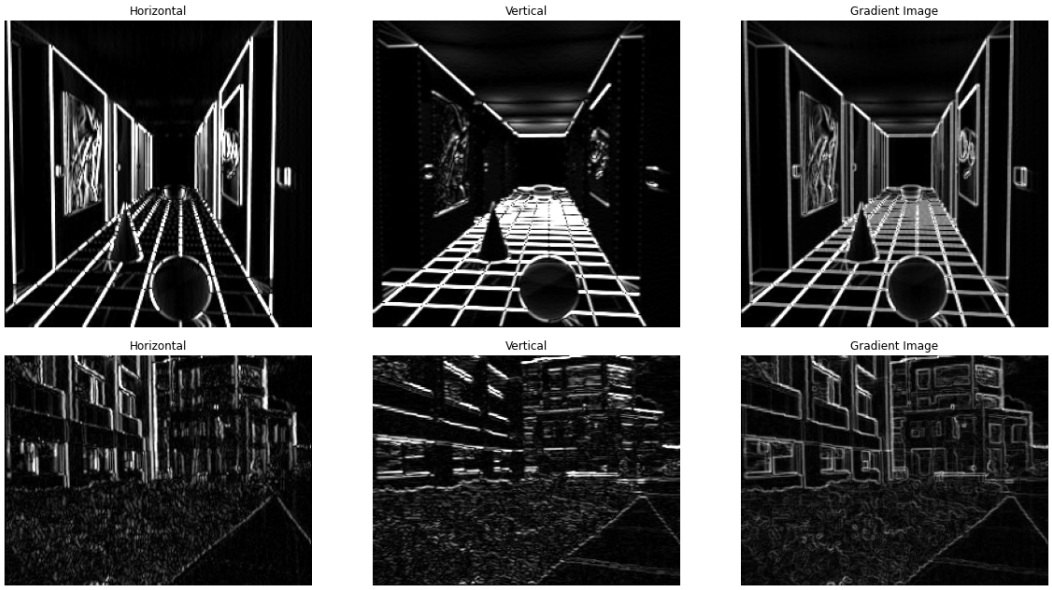



Figure 3.2: Sobel Operator Results

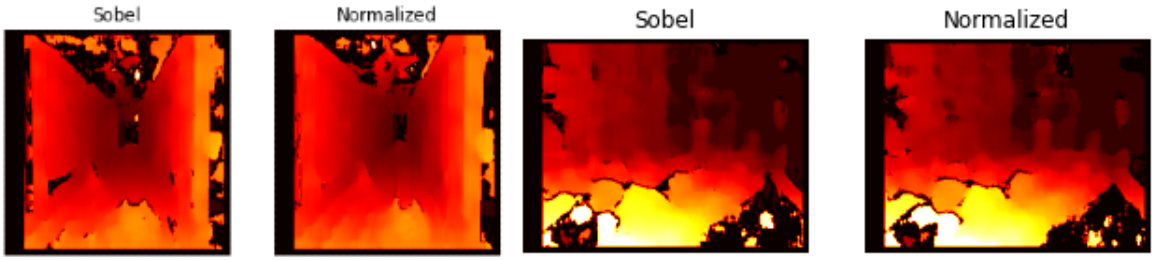



Figure 3.3: Sobel and Normalized

## 3.2 Post-processing

### 3.2.1 Left-Right Consistency Test

As the disparity algorithm in Chapter 1.2, we only consider the left images as the reference and produce the corresponding disparity maps. Nevertheless, there are some regions that are visible in one image but not visible in another image, which is so called half-occlusions. These regions are usually in the left part of the left images and the right part of the right images as shown in Fig 3.4. Since those regions are only in one image, they should not be matched anyway. To solve this problem, we need to symmetrically take both the left and right images as the reference, and only consider those pixel pairs that are consistency.

To do that, we need to do the Left-Right Consistency test. The specific steps are:

1. For every pixel in the left image, find the corresponding pixel in the right image according to the disparity we get when taking the left image as the reference.
2. Then, with the disparity at that location in the right image, which we obtain when taking the right image as the reference, we can move backwards in the left image

to find the result pixel.

3. Finally, we can check if the result pixel is the same as the starting pixel. If not, then we consider the pixel is inconsistency.

Another benefit of doing this test is that we can detect not only half-occlusions but also noises and outliers.

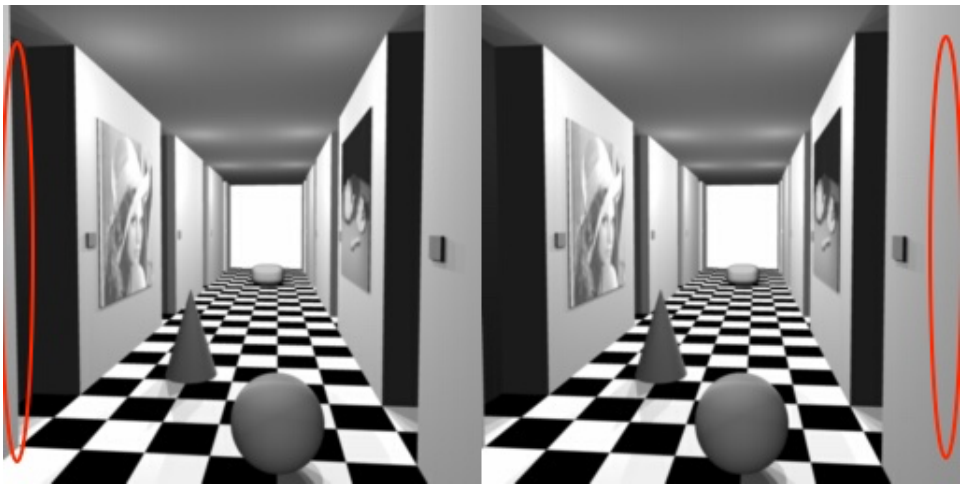


Figure 3.4: Half Occlusions

The results are shown in Fig 3.5. According to the disparity map we obtain after consistency test, the result for half-occlusions part is improved to some degree. However, we notice that there are a large number of pixels are considered as inconsistency somehow. To fix the problem, we introduce the maximum allowable difference, and if the difference exceeds the threshold, the disparity of this pixel will be set to 0.

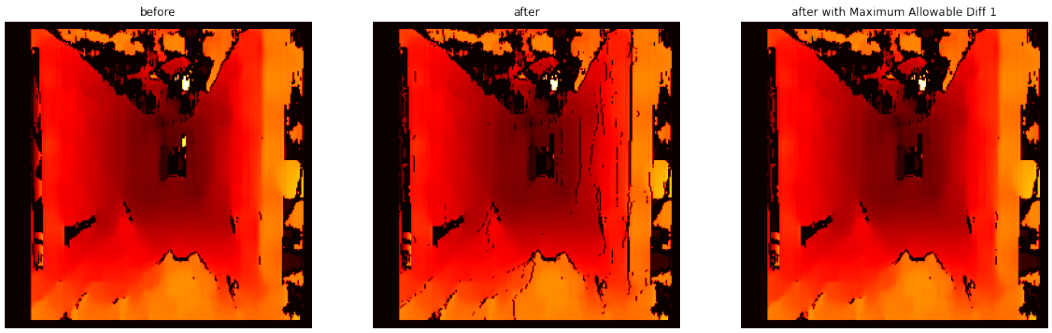



Figure 3.5: Before and After Consistency Test

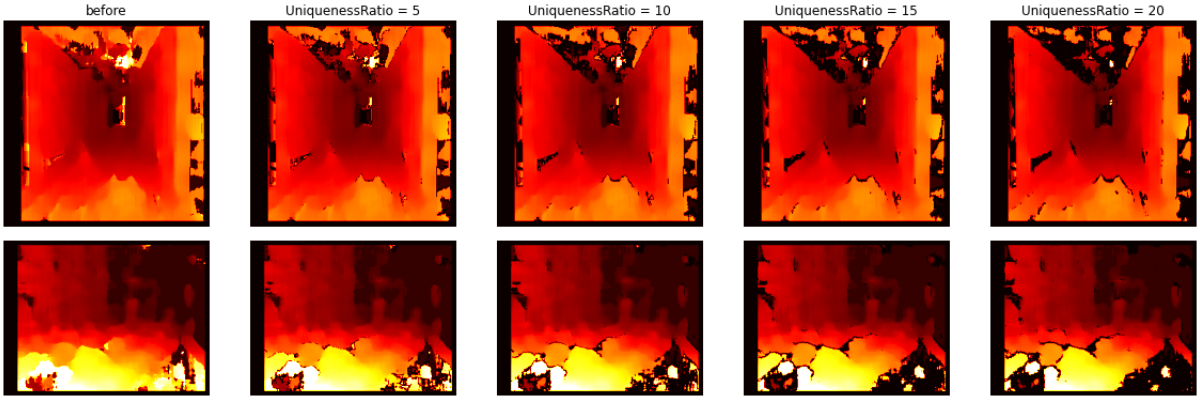



Figure 3.6: Before and After Ratio Test

### 3.2.2 Uniqueness Ratio Test

Another post-processing step is to do uniqueness ratio test. We need to compare the best matching disparity with the second-best matching disparity, and check whether

$$\frac{SAD(bestd)}{SAD(secondbestd)} \geq 1 + \frac{UniquenessRatio}{100}$$

.

If not, then the pixel is filtered out and a result of 0 will be given. In Fig 3.6, we try several different uniqueness ratios and compare them with the disparity map without uniqueness ratio test.

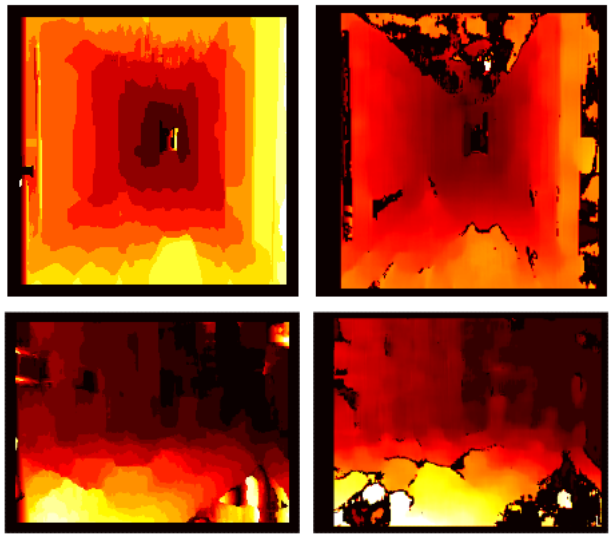


Figure 3.7: Before and After Pre- & Post-Processing

In Fig 3.7, we illustrate the disparity maps before and after pre-filtering and post-filtering. According to the results, while the performances become better, there is an obvious limit for this basic algorithm. In order to achieve further improvement, we try different algorithms in the next chapter.

# Chapter 4

# Advanced Approaches

## 4.1 Spatial Regularization

### 4.1.1 Global Matching - Stereo MRF

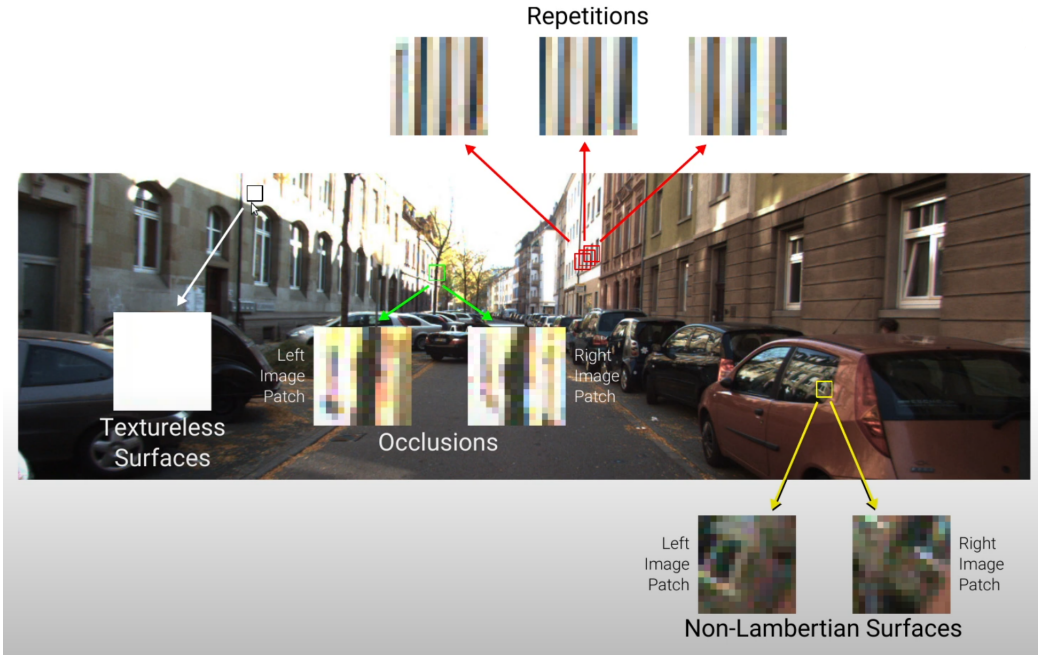



Figure 4.1: Failure cases due to local ambiguity[7]

For Block Matching algorithm demonstrated in chapter 2, the disparity value of each pixel is searched within a local rectangular region and determined based on a winner-takes-all (WTA) strategy. However, there are some cases where the local-based approach fails due to local ambiguities. See figure 4.1, from left to right, the first patch is drawn within a over-exposed area, where a part of the images is completely white and textureless. For the second patch, the left image and the right image look very different, because the tree in front of the building occludes different parts of the building. In the third to fifth patches (the red patches), similar patterns occur repeatedly along

the white building. In the last patch, the left image and the right image have different look due to specular reflection. All these cases will lead to failures of block matching algorithm and are hard to solve locally.

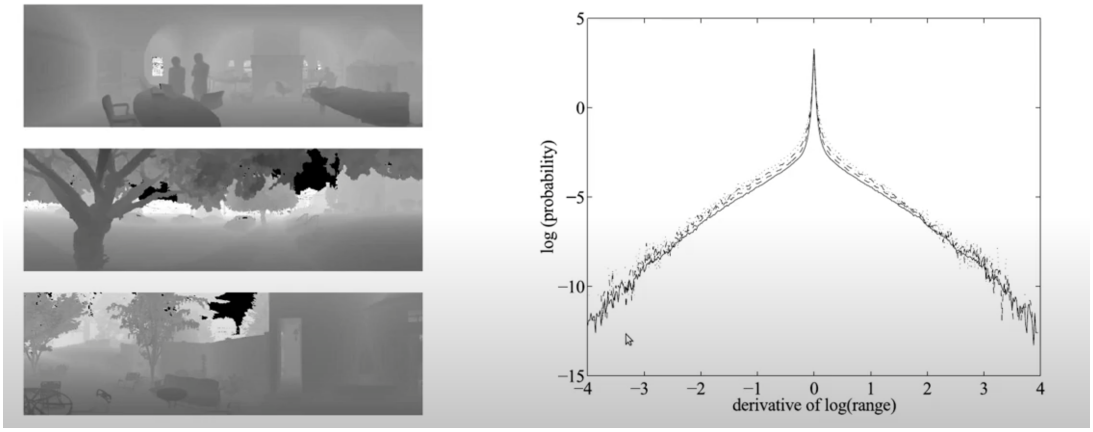



Figure 4.2: Distribution of first order derivative of pixels in disparity map of real-world images

On the other hand, real-world images have such property: Depth varies slowly and smoothly between most pixels, except at object discontinuities which are sparse. As shown in figure4.2, most pixels have small derivative close to 0, on which the depth changes slowly. We can leverage this knowledge as a global constraint and incorporate this constraint to disparity estimation. This can be done by applying a loopy Markov Random Field (MRF)[6] and solve the disparity map $D$ as an energy minimization problem:

$$p(D) \propto exp(-\sum_{i} \phi_{data}(d_i) - \lambda \sum_{i,j} \phi_{smooth}(d_i, d_j)) \tag{4.1}$$

Where unary term $phi_{data}(d_i)$ is the matching cost of each pixel. Minimizing this term solely is equivalent to using basic block matching with WTA. Pairwise term $\phi_{smooth}(d_i, d_j)$ indicates the smoothness between one pixel and its adjacent pixels $i$ and $j$, which models the prior assumption about the smoothness of adjacent pixels.

This global based approach overcomes the difficulties we met with local block matching. However, despite many algorithms like graph cut and belief propagation for accelerating, minimizing this global energy function, being an NP-hard problem, is still very time-consuming. This issue leads to a new method - semi-global matching.

### 4.1.2 Semi-Global Block Matching - SGM/SGBM

The process of given a pair of rectified images to compute the disparity map is also called stereo matching. There is a trade-off between global stereo matching algorithm and local stereo matching algorithm. In 2005 Heiko Hirschmuller[5] come up with a balanced approach - semi-global matching(SGM), which is both more accurate than

block matching and more time-efficient than minimizing global energy function. Recall the equation (4.1), the general form of an energy function is defined as:

$$E(D) = E_{data}(D) + E_{smooth}(D) \tag{4.2}$$

Where $E_{data}$ is the similarity or matching cost of each matched pixels, and $E_{smooth}$ is the smoothness between pixels. SGM pixel-wise proposed mutual information (MI) as the matching cost $E_{data}$ for each pixel, and a two-staged hierarchical smooth constraint . Most importantly, SGM optimize $E(D)$ in multiple 1D paths instead of global 2D image. As mentioned in last section, minimizing $E(D)$ globally is an NP-hard problem, whereas minimizing $E(D)$ in a 1D path can be done in polynomial time with dynamic programming.

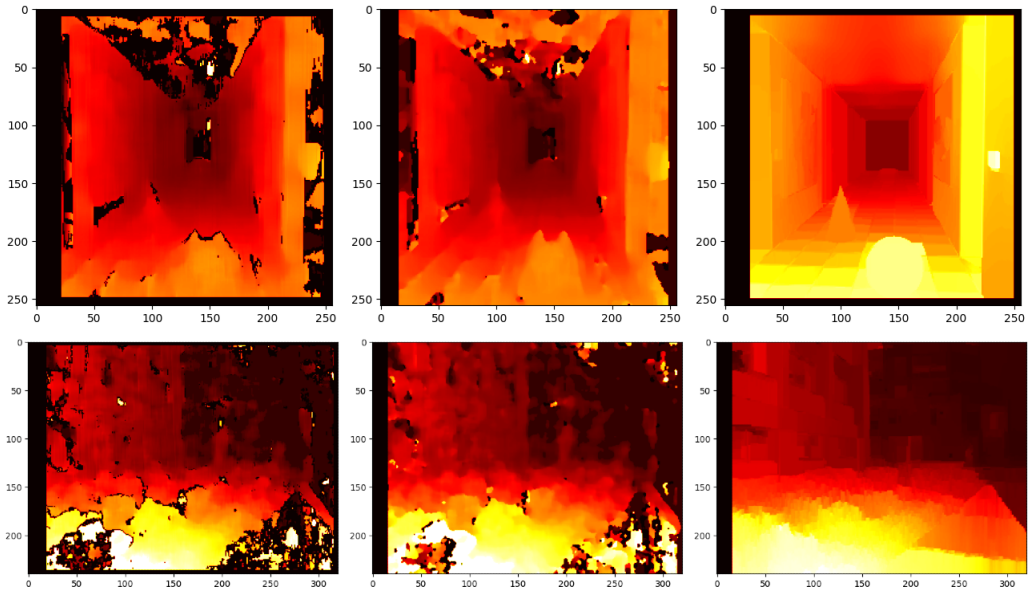


Figure 4.3: From left to right: BM, SGBM, SGBM + WLS filter

SGM algorithm is implemented in the computer vision library OpenCV[1], known as **stereoSGBM**. We leverage the openCV implementation of SGM to compare its result with our plain block matching method, as shown in figure4.3. OpenCV also contributed many post filters, such as the WLS-filter, and we append the result of SGBN plus WLS to figure4.3. Note that all three set of images are generated with same parameter configuration - $blockSize = 13$ and $numDisparities = 16$. Looking at the ceiling in the first row of images and the road in the second row of images, both BM and SGBM struggle with large homogenous textures, but SGBM gives less error. Compared with BM, SGBM shows less error overall, especially for the boundary of foreground objects, for example, the boundary of the sphere in the first image. The WLS filter filters out low-confidences regions in SGBM results and apply interpolations, which finally gives the best result.

## 4.2 Deep Learning Approaches

In previous sections we illustrate how to perform stereo matching via traditional computer vision techniques, which leverage hand crafted features to compute disparity maps. However, the hand-crafted features and similarity metrics used in those approaches do not take into consideration 3D geometries, occlusion patterns or specular reflections. Therefore, most traditional methods do not give good results on real-world images with complex scenes and light conditions. Nowadays, deep learning models are widely used instead, which are capable to automatically extract desired features and learn similarity metrics from data.

### 4.2.1 Bench Mark

Since stereo matching is one of the core technologies in computer vision, widely used in areas such as autonomous driving, many deep learning models are designed to solve this task and there are some popular datasets used as benchmarks. KITTI[4][7], is computer vision benchmark suite, which provides high-resolution labelled images captured, useful tool kits, evaluation metrics and evaluation websites. Recently, they also create virtual KITTI dataset[3] containing photo-realistic synthetic images for training. The images in KITTI datasets are mainly scenes on roads for autonomous driving, and the image resolution is considerably higher than our data (about 256x256). In the next section, we choose a SOTA model that gains a high rank on KITTI datasets to perform stereo matching on our images.

### 4.2.2 PSMNet

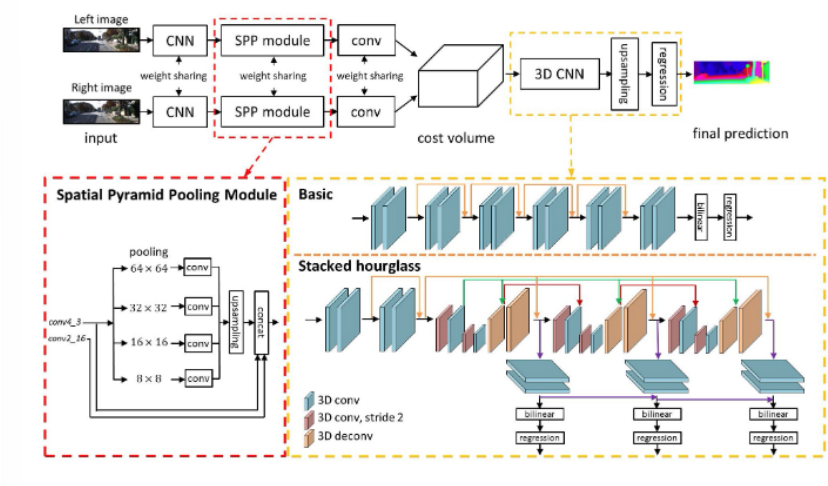



Figure 4.4: From left to right: BM, SGBM, SGBM + WLS filter

Pyramid Stereo Matching Network (PSMNet)[2] achieves SOTA performance in stereo matching using a novel architecture illustrated in figure 4.4. The network is composed of two modules: a pair of spatial pyramid pooling modules and a 3D CNN module. The spatial pyramid pooling module within the Siamese network takes advantage of the capacity of global context information by aggregating context in different scales

and locations to form a cost volume. Then, the 3D convolution network module takes the cost volume as input and learns to regularize it using stacked multiple hourglass networks with intermediate supervision. Figure4.5 shows the result of PSMNet on our images and figure4.6 show the result of PSMNet on a random image in KITTI dataset. On figure 4.5, although PSMNet gives a clean disparity map and accurately predict the foreground objects (the sphere and the cone), the result is not comparable with SGBM + WLS in figure4.3. What's more, the models fail on the second image. However, looking at figure 4.6, the network generates a much better result, and extraordinarily tackles the road surface with complex texture and shadows. We don't show the result of SGBM or BM, because they completely fail on this image. This huge performance gap is because our data is different from the images in the KITTI dataset. PSMNet , and most network for stereo matching are trained with high resolution images in RGB format. To leverage these models, we need to upscale our images, which will introduce noise and distortion. Nevertheless, figure4.6 demonstrates the power and robustness of deep learning methods compared with traditional models.

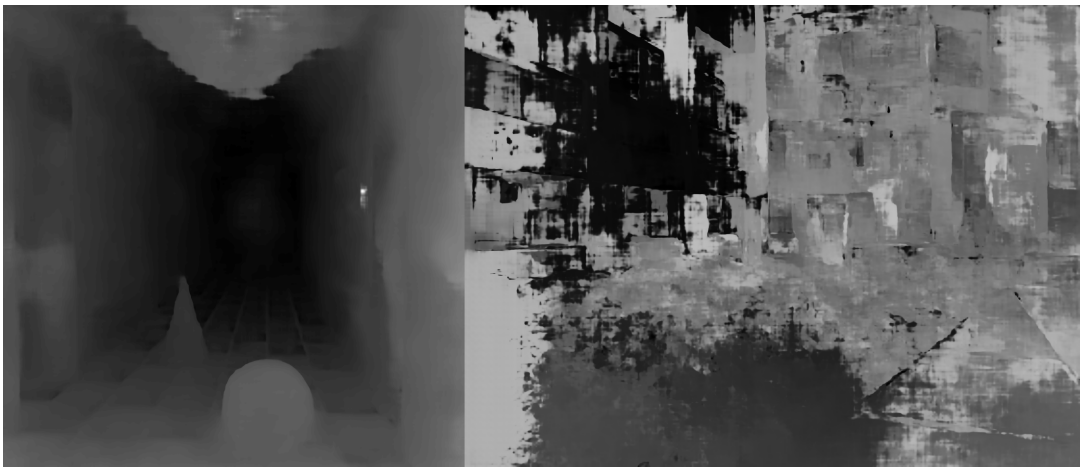


Figure 4.5: From left to right: BM, SGBM, SGBM + WLS filter

Figure 4.6: From left to right: BM, SGBM, SGBM + WLS filter

# Bibliography


[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network, 2018.

[3] A Gaidon, Q Wang, Y Cabon, and E Vig. Virtual worlds as proxy for multi-object tracking analysis. In *CVPR*, 2016.

[4] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[5] H. Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 807–814 vol. 2, 2005.

[6] Jinggang Huang, A.B. Lee, and D. Mumford. Statistics of range images. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*, volume 1, pages 324–331 vol.1, 2000.

[7] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.