Disparity Map Calculation on GPU

EE4750_2016_Fall_DMCG_Report
Qingwei Wu qw2208, Ruixuan Zhang rz2364
Columbia University

***Abstract*—In this paper, we calculate the depth map based on the two cameras which displaced horizontally from one another which are used to obtain two differing views on a scene, similar to human binocular vision. Techniques include local optimization algorithm of Sum of Absolute Difference (SAD) and Dynamic Programming (DP). The most challenging work is designing a matching rule to make the disparity map accurate and evacuate noise in the disparity map. Another big challenge should be reducing time complexity and optimizing memory structure on GPU and dealing with the boarder statements. It turns out that disparity map acquired by algorithm of SAD is smoother than that acquired by DP and GPU is much faster to calculate disparity with parallel computing.**

***Index Terms*—depth map, disparity calculation, parallel computing, sum of absolute difference, dynamic programming**

I. Introduction

A. Background

In traditional stereo vision, two cameras, displaced horizontally from one another are used to obtain two differing views on a scene, in a manner similar to human binocular vision. By comparing these two images, the relative depth information can be obtained in the form of a disparity map, which encodes the difference in horizontal coordinates of corresponding image points [1]. In detail, disparity map is a gray-scale image where white pixels imply that the object is closer to the camera and vice versa. With disparity map, we can easily derive the actual distance of objects in meter with some simple math calculation, see chapter 11 of [2]. In this project, we put emphasis on calculating the disparity map. What's more, the images we got from the horizontally displaced cameras have been rectified which means that we look for the best-matching pixel in the same row of the left image and right image.

Implementation of disparity map calculation is traditionally divided into four steps: matching cost computation, aggregation, disparity computation/optimization and refinement [3]. The matching cost is how you define the difference between blocks of two images respectively. Simplest art of matching cost is squared intensity difference (SD) [4] or absolute intensity difference (AD) [5] in color intensity between the corresponding pixels. Other matching cost includes truncated quadratics and contaminated Gaussians [6] and normalized cross-correlation [7]. Nevertheless, a single pixel in a picture of .jpg format could range from 0 and 255 only, which is likely to lead to a mismatch. Thus, the neighbors of this pixel should be taken into consideration. This is why aggregation is invoked. Local and window-based methods aggregate the matching cost by summing over a support region. The support region could be two-dimensional at a fixed disparity. Traditional two-dimensional aggregation has been implemented using square windows. Advanced aggregation techniques include shiftable windows [8] and windows with adaptive sizes [9] [10]. Matching cost computation and aggregation could be considered as refining our matching problem. Disparity optimization could be the main matching rule. Local methods of "winner-take-all" (WTA) and global optimization algorithm of "dynamic programming" (DP) are the main topic in this paper. We'll elaborate on these algorithms in the following sections.

Another big issue for this project should be applying parallel computing knowledge to realize the real-time calculation of the disparity map. Previous work has focused on GPU performance of disparity calculation. [11] has realized high-quality real-time stereo matching using adaptive cost aggregation and dynamic programming. Other paper like [12] also show their results using CUDA.

B. Our work

We implement "sum of absolute difference" (SAD) and "dynamic programming" (DP) without using existing libraries like "Opencv" on CPU or GPU and compare the efficiency and performance. Implemented on GPU, we try to optimize the memory structure for SAD and DP.

Section II will briefly introduce the SAD we implemented from serial processing version and parallel processing version. We'll also focus on the memory structure on GPU. Section III will introduce DP algorithm we implemented from serial processing version and parallel processing version. Our work on memory optimization for DP is also demonstrated. Section IV will show the outcome disparity map and the efficiency comparison. In section V, future work is discussed.

II. Local Optimization Algorithm

A. SAD Model

Sum of Absolute Difference (SAD) with Winner-Takes-All (WTA) is a typical local matching strategy. To illustrate the SAD model, we start from a simple example. The upper figure is a clip from the left image while the lower figure from the right image. We want to find the corresponding pixel for the pixel in the center of the black square and record the offset as shown in Fig. 1. The black square is the support window, same as support region introduced in section I in the aggregation step. Given the two rectified images, the matching block should be in the same row with the black square. Also, suppose the yellow rectangle is the searching zone since disparity can be limited.

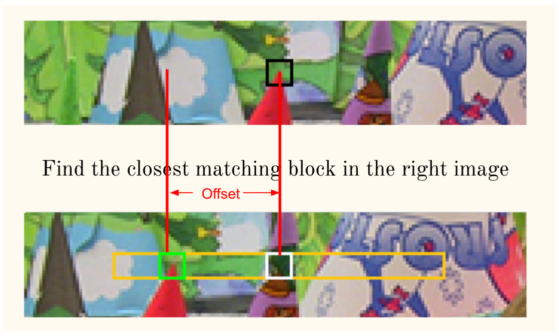

Fig. 1. Example for SAD Model

Here we define the matching cost computation of Absolute intensity Difference (AD). For instance, if we want to find the difference between the white square and the black square. The black square is represented as a matrix in (1).

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,(m-1)} & x_{1,m} \\ \cdots & \cdots & x_{i,j} & \cdots & \cdots \\ x_{m,1} & x_{m,2} & \cdots & x_{m,(m-1)} & x_{mn} \end{bmatrix} \tag{1}$$

The white square is represented as (2).

$$\begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,(m-1)} & y_{1,m} \\ \cdots & \cdots & y_{i,j} & \cdots & \cdots \\ y_{m,1} & y_{m,2} & \cdots & y_{m,(m-1)} & y_{mm} \end{bmatrix} \tag{2}$$

The difference of the two squares is defined as

$$\Delta = \sum_{i=1}^{m} \sum_{j=1}^{n} |x_{i,j} - y_{i,j}| \tag{3}$$

White block is moved in the yellow zone. We use WTA computation algorithm. In other words, only the white block who has the smallest difference with the black square is chosen as the matching block and its center pixel is matched with the center pixel of the black square and the coordinate offset is recorded as the disparity value of the center pixel of the black block.

B. Serial Algorithm

Pseudocode is shown as following.

Algorithm 1: Pseudocode for SAD on CPU

```
Input: left picture matrix $L$, right picture matrix $R$
Output: disparity map $D$
1 for every element in $n * n$ matrix $L$ do
2     for $d$ in the disparity range do
3         for each pixel in $m * m$ window do
4             Update the difference value;
5         Apply WTA for choosing the matching
          pixel and record the offset;
6     Record offset in $D$;
7 return $D$;
```

Size of the searching zone is denoted as d_{max}. It's obvious that the time complexity of algorithm 1 is $O(n^2 m^2 d_{max})$ and the calculation of each element in the matrix is independent, resulting in GPU parallel computing.

C. Parallel Algorithm

Inspired by image convolution on GPU, we try to realize the parallel computing algorithm for SAD. We show the memory structure in the following figure.

Figure 2 shows how memory is managed. Elements in the thread block are matched simultaneously. The blue rectangle implies the private memory for one thread which filled with left image color intensity. Size of the blue rectangle is the window size. The red rectangle represents shared memory which filled with right image color intensity. Size of the shared memory is $(tile\_width + window\_size, disparity\_level + window\_size + tile\_width)$, where $tile\_width$ is the input block dimension. We also show the block diagram.

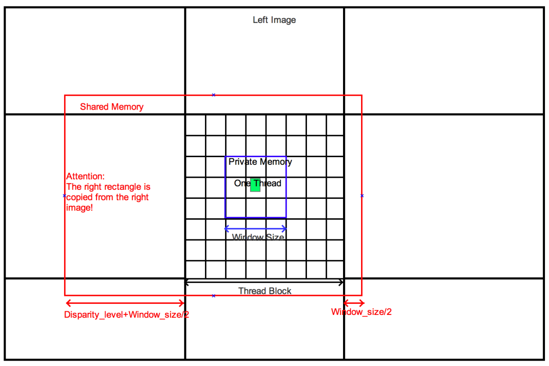

Fig. 2. Memory Structure on GPU

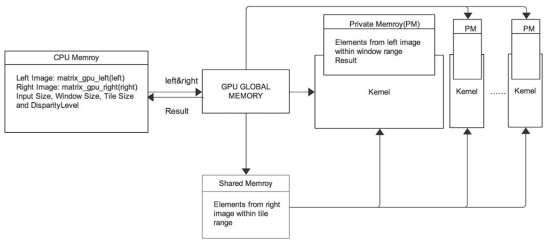
CPU Memroy
Left Image: matrix_gpu_left(left)
Right Image: matrix_gpu_right(right)
Input Size, Window Size, Tile Size and DisparityLevel
left&right
Result
GPU GLOBAL MEMORY
Private Memroy(PM)
Elements from left image within window range
Result
Kernel
PM
Kernel
......
PM
Kernel
Shared Memroy
Elements from right image within tile range

Fig. 3. Block Diagram for SAD

Step 1 for serial algorithm can be parallelized by calculating each element in a thread and the complexity is $O(m^2 d_{max})$ which reduces complexity by n^2.

Pseudocode is shown as following.

Algorithm 2: Pseudocode for SAD on GPU

```
    Input: left picture matrix $L$, right picture matrix $R$
    Output: disparity map $D$
 1  for every thread block do
 2      for pixel in support window do
 3          Save pixel intensity from $L$ in private memory;
 4      Save pixel intensity in the red rectangle from $R$ in shared memory;
 5      for $d$ in the disparity range do
 6          for each pixel in $m * m$ window do
 7              Update the difference value as (3);
 8          Apply WTA for choosing the matching pixel and record the offset;
 9      Record offset in $D$;
10  return $D$;
```

D. Sum of Squared Difference (SSD)

Squared color intensity difference is another traditional and commonly used computing matching computation. SSD is very similar to SAD. The only difference is the algorithm for cost computation. Recall equation (3) and define the new difference as following.

$$\Delta = \sum_{i=1}^{m}\sum_{j=1}^{n}(x_{i,j} - y_{i,j})^2 \tag{4}$$

III. Dynamic Programming (DP)

A. Main Idea

For implementation simplicity, we jump through the aggregation step and apply AD as matching cost computation for DP. There are two steps for dynamic programming, forward and backward.

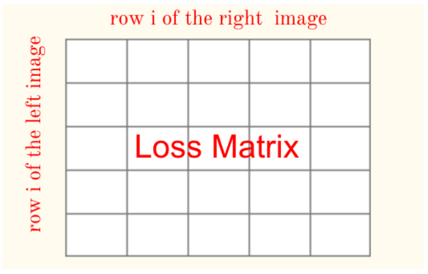

Fig. 4. Loss Matrix Δ

Suppose we have a loss matrix for matching row i of the left image and the right image. For forward step, we start from the top-left corner to the bottom-right corner of the loss matrix. Every element in the loss matrix, for instance $\Delta(p, q)$, is the value of the smallest possible total difference accumulated so far from the top-left corner if p^{th} pixel in row i of the left image is matched with q^{th} pixel in row i of the right image. More explicitly, recursive formula is written down.

$$\Delta(p,q) = \min[\Delta(p-1,q), \Delta(p,q-1), \Delta(p-1,q-1)] + |\boldsymbol{L}(i,p) - \boldsymbol{R}(i,q)| \tag{5}$$

For backward step, we try to figure out what's the matching strategy with the smallest difference. Start from the bottom-right corner and move to the top-left corner of the loss matrix according to the following rules. Suppose we are at (p, q) of the loss matrix.

- If $\Delta(p-1, q-1) <= \min\{\Delta(p-1, q), \Delta(p, q-1)\}$, move upleft.
- If $\Delta(p, q-1) <= \min\{\Delta(p-1, q), \Delta(p-1, q-1)\}$, move up.
- If $\Delta(p-1, q) <= \min\{\Delta(p-1, q-1), \Delta(p, q-1)\}$, move left.

Only when moving up-left, $\boldsymbol{L}(i, p)$ and $\boldsymbol{R}(i, q)$ is matched. When moving left, we claim the pixel in the right image is occluded. When moving up, we claim the pixel in the left image is occluded.

B. Serial Algorithm for DP

Algorithm 3: Pseudocode for DP on CPU

```
Input: left picture matrix $\boldsymbol{L}$, right picture matrix $\boldsymbol{R}$
Output: disparity map $\boldsymbol{D}$
1 for every row $i$ in $\boldsymbol{L}$ do
2     Forward Step: calculate the loss matrix;
3     Backward Step: get the optimal path and record
      the loss matrix coordinate (say $(x, y)$) when the
      next move is up-left;
4     if coordinate in line $m$ moves up-left then
5         $\boldsymbol{D}(i, x) = abs(x - y)$;
6     else
7         $\boldsymbol{D}(i, x) = 0$;
8 return $\boldsymbol{D}$;
```

Pseudocode is shown in Algorithm 3.

The code is another short review of what we've just illustrated. In practice, there might not be a coordinate who will move up-left. An exception processing is required.. An arbitrary solution is given since we claim that these exceptions won't occur too often and these pixels will be smoothened by the median filter. The present code provides a $O(n^3)$ algorithm, given n the width and height of the image. However, since the calculation of each row is independent, we're interested in the possibility of parallel computation of each row.

The for loop can be parallelized by matching each row in the left and right image in a thread.

C. Parallel Algorithm for DP

Pseudocode is demonstrated in Algorithm 4.

With parallel, the time complexity could be reduced to approximately $O(n^2)$.

D. Optimized Memory Structure

Memory efficiency is a big issue to be discussed when implementing DP on GPU. In fact, we fail to load the whole loss matrix into private memory if the dimension of the loss matrix appears to be $(image\_width, image\_width)$. However, in practice, the disparity level should be much smaller than $image\_width$. We are interested in diagonal elements of the loss matrix. Thus, the original loss matrix is converted to a dense matrix as shown in Fig.5.

The triangle should be padded with zeros. At the same time, the recursive formula is slightly altered.

Algorithm 4: Pseudocode for DP on GPU

```
Input: left picture matrix $\boldsymbol{L}$, right picture matrix $\boldsymbol{R}$
Output: disparity map $\boldsymbol{D}$
1 Deploy thread block as $(\boldsymbol{L}_{height}, 1, 1)$. Each thread
  takes the following action;
2 Forward Step: calculate the loss matrix;
3 Backward Step: get the optimal path and record the
  loss matrix coordinate (say $(x, y)$) when the next
  move is up-left;
4 if coordinate in line $m$ moves up-left then
5     $\boldsymbol{D}(i, x) = abs(x - y)$;
6 else
7     $\boldsymbol{D}(i, x) = 0$;
8 return $\boldsymbol{D}$;
```

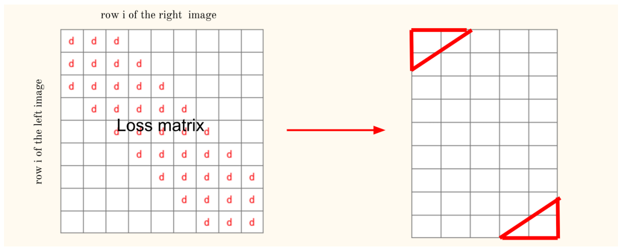

Fig. 5. Memory Optimized for GPU

IV. Results

The ground truth given by [3] is also shown.

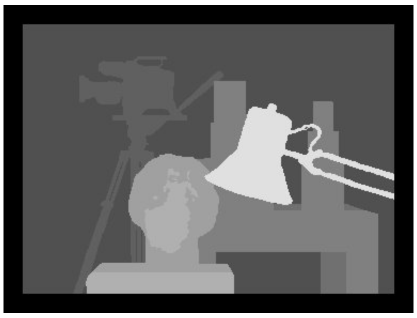

Fig. 6. Ground Truth for Tsukuba

We evaluate our results based on the benchmark Middlebury stereo data set, which provides us with rectified images. The images we use are Tsukuba [3] and bowling [13]. It can be downloaded from [14]. Dataset also provides the ground truth as shown in Fig. 6.

A. Disparity Map Comparsion

We analyze the disparity map through the smoothness of the object borders and the amount of noises in the output [1]. Corresponding disparity maps derived on GPU and CPU are the same. Thus in this part, we don't show CPU and GPU disparity maps separately for brevity.

For local algorithm SAD, we fix the window size as $7 * 7$ and alter the largest disparity value. Compared to ground truth, disparity map derived by setting the largest disparity value as 20 is as good as disparity map derived by setting the largest disparity value as 32. And they take almost same running time, we simply set the largest disparity value as 20.

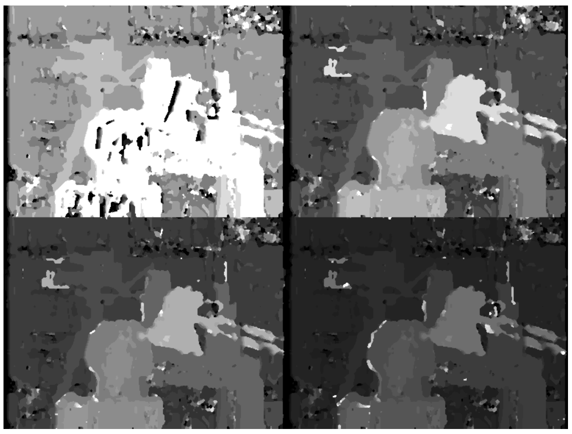

Fig. 7. Largest Disparity vale (TopLeft: 8: TopRight: 16; BottomLeft: 20; BottomRight: 32)

Another parameter we alter is the size of the support window. We choose the size of the support window as 5 and 7 and show disparity maps as following.

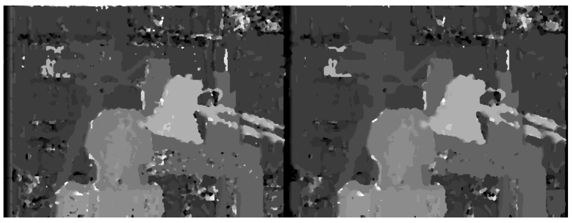

Fig. 8. Support Window Sizes: 5 and 7

From Fig. 8, the right image is smoother than the left image. We claim that if support window size is set as 7, support window manages to cluster more information and fewer noises appear in the disparity map.

Then we use same parameters for SSD. From Fig.9 we can see that SSD and SAD have almost same accuracy. Only absolute-value computation are replaced by multiplication operation, SSD and SAD have basically the same running time.

We set the max disparity value as 20 for DP. Disparity map derived by DP is shown as Fig. 10.

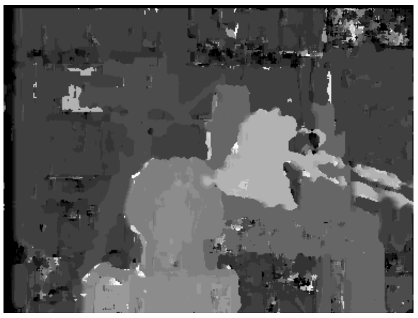

Fig. 9. Derived by SSD

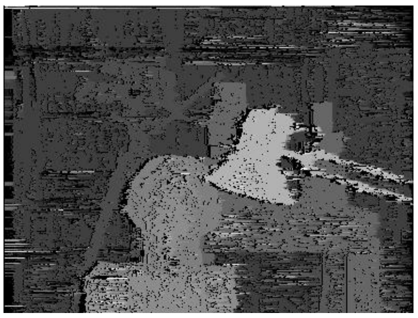

Fig. 10. Derived by DP

The disparity map derived by SAD/SSD is better than DP because there's less noise on the object.

B. *Execution Time Comparison*

The execution time by both the SAD algorithm and DP is shown in the following table. We apply SAD with the largest disparity value as 20. The support window size of SAD1 is $5 * 5$. support window size of SAD2 is $7 * 7$. For SSD, the support window size is $7 * 7$.

TABLE I
EXECUTION TIME COMPARISON

	SAD1	SAD2	SSD	DP
CPU	56.552s	100.401s	92.151s	38.1595s
GPU	0.242s	0.233s	0.437s	0.383s
Speed UP	233.70	431.81	210.87	99.63

V. FUTURE WORK

In local matching algorithms, adaptive cost aggregation could be used to improve the disparity accuracy. Besides WTA and DP, many optimization algorithms including Graph Cut could be applied to get a smoother disparity map. In this project, we focus on one color channel. However, different color channels provide more information.

VI. Conclusions

We implement three different algorithms including SAD, SSD and DP on GPU to accelerate disparity map calculation. Regardless of the algorithm, GPU can increase the speed of calculation by hundreds of times with almost the same output. Even though optimized memory structure is used for DP, it still takes much more time than SAD on GPU because of too many conditional statements. Disparity map derived by DP is no better than SAD. GPU is good at dealing with parallel tasks with simple control and roughly can achieve a real-time computation in some occasions.

VII. Aknowledge

Codes for SSD and SAD on CPU are from [15]. Other works are all original.

References

[1] "https://en.wikipedia.org/wiki/Computer_stereo_vision".

[2] R. Jain, R. Kasturi, and B. Schunck, "Machine vision, book," 1995.

[3] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," International journal of computer vision, vol. 47, no. 1-3, pp. 7–42, 2002.

[4] M. J. Hannah, "Computer matching of areas in stereo images," tech. rep., DTIC Document, 1974.

[5] T. Kanade, H. Kano, S. Kimura, A. Yoshida, and K. Oda, "Development of a video-rate stereo machine," in Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on, vol. 3, pp. 95–100, IEEE, 1995.

[6] M. J. Black and P. Anandan, "A framework for the robust estimation of optical flow," in Computer Vision, 1993. Proceedings., Fourth International Conference on, pp. 231–236, IEEE, 1993.

[7] R. C. Bolles, H. H. Baker, and M. J. Hannah, "The jisct stereo evaluation," in DARPA Image Understanding Workshop, pp. 263–274, 1993.

[8] R. D. Arnold, "Automated stereo perception.," tech. rep., DTIC Document, 1983.

[9] M. Okutomi and T. Kanade, "A multiple-baseline stereo," IEEE Transactions on pattern analysis and machine intelligence, vol. 15, no. 4, pp. 353–363, 1993.

[10] T. Kanade and M. Okutomi, "A stereo matching algorithm with an adaptive window: Theory and experiment," IEEE transactions on Pattern analysis and Machine Intelligence, vol. 16, no. 9, pp. 920–932, 1994.

[11] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister, "High-quality real-time stereo using adaptive cost aggregation and dynamic programming," in 3D Data Processing, Visualization, and Transmission, Third International Symposium on, pp. 798–805, IEEE, 2006.

[12] R. Kalarot and J. Morris, "Implementation of symmetric dynamic programming stereo matching algorithm using cuda," in Sixteenth Korea-Japan Joint Workshop on Frontiers of Computer Vision, vol. 4, p. 5, 2010.

[13] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in 2007 IEEE Conference on Computer Vision and Pattern nition, pp. 1–8, IEEE, 2007.

[14] "http://vision.middlebury.edu/stereo/data/".

[15] "https://github.com/davechristian/Simple-SSD-Stereo".