Real-Time Stereo Matching Using Semi-Global Methods with CUDA Implementation

Michael Lowney
Robert Mahieu

Introduction

Acquiring depth information from sets of images is incredibly important in many emerging fields such as augmented reality, robotics, autonomous vehicles, etc. However, these applications rely on the produced depth information to be both accurate and generated in a short amount of time—ideally close to real-time—to ensure safety of the system and users, as well as for reliable pose tracking.

Many algorithms for computing this information have been previously explored, either looking at localized information or global information throughout the entire image. Local techniques such as Winner-Takes-All (WTA) or scanline optimization (SO) [1] compute results for pixels independently and require minimal computation, but due to lack of consideration for global trends typically result in inaccurate conclusions. The dynamic programming (DP) [2] approach is also computationally efficient, but because the algorithm only looks at a single row per iteration, it also lacks consideration global trends and commonly causes streaking patterns to show up in the output. On the other hand, while global techniques such as a Graph Cuts [6] and Belief Propagation [4] produce more accurate results and better avoid the errors encountered in the local methods, these techniques are significantly more memory intensive and end up being much slower.

To obtain both reasonable accuracy as well as real-time performance, we instead move to what can be referred to as a semi-global matching technique [3], which makes some use of both local and global methods. Additionally, offloading data calculation onto the GPU, which is ideal for handling SIMD (single instruction multiple data) computation, allows us to exploit the parallelizable nature of our image-based calculations and significantly reduce computation time for estimating the optimal disparity map.

Technical Approach

The implementation of the semi-global matching method comes down to minimizing an energy function describing the quality of a potential disparity image. This is represented by the expression below:

$$E(D) = \sum_{p} \left(C(p, D_p) + \sum_{q \in N_p} P_1 \mathbf{1}\{|D_p - D_q| = 1\} + \sum_{q \in N_p} P_2 \mathbf{1}\{|D_p - D_q| > 1\} \right)$$

Where D is the disparity map, p is a pixel location on the map, D_p is the disparity value at pixel p, N_p is the neighborhood of pixels around p, $\mathbf{1}\{?\}$ is an indicator function that is equal to one if the argument within the braces is true and zero if false, and P_1 and P_2 represent penalty value given to various changes in disparity within the local neighborhood. $C(p,d)$ represents an initial cost function which is based on the absolute difference between gray levels at a pixel p in the base reference image and pixel p shifted by d along the epipolar line in the matching image (assumed to be the right image in the stereo pair):

$$C(p,d) = |I_b(p_x, p_y) - I_m(p_x - d, p_y)|$$

Note also that the images are assumed to be rectified. The energy function thus penalizes heavily (with P_2) large jumps in the disparity map and less heavily (with P_1) small changes that may represent sloped surfaces. Note that $P_1 < P_2$. In order to fit these penalty values to the scene at hand, we choose them inversely proportionally to the image gradient.

However, optimizing this energy function using global minimization in 2-dimensions requires too much computation to solve, while minimizing in 1-dimension over image rows, such as in the DP approach, is light on computation, but suffers from accuracy issues as discussed above. To handle this problem, semi-global matching leverages several 1-dimensional minimization functions to more efficiently construct an adequate estimate of the solution.

The first step in the actual implementation of the algorithm is to calculate initial costs for all pixels in the image pair at disparities ranging from 0 to some selected d_{max}. This can be efficiently carried out on the GPU with a kernel that gives each thread a computation for one pixel and one disparity value.

To carry out the next step denoted “cost aggregation”, we iterate and compute the energy function locally over 8 directions (two horizontal, two vertical, two for each diagonal). The recursive expression, for example, for the horizontal direction, going from left to right across the image, is represented as:

$$\begin{aligned} E(p_x, p_y, d) = {} & C(p,d) \\ & + \min\Big(E(p_x - 1, p_y, d)E(p_x - 1, p_y, d-1) + P_1, E(p_x - 1, p_y, d+1) \\ & + P_1, \min_i\big(E(p_x - 1, p_y, i) + P_2\big)\Big) \end{aligned}$$

This can be parallelized by having each thread on the GPU handle one iteration of a given direction and one disparity. Note that after each step along the direction, the threads must be synchronized.

Once all paths have been traversed, results are compiled into a single value:

$$S(p,d) = \sum_r w_r * E_r(p,d)$$

Where r represents a given direction and w_r represents a weight value for that particular direction. By introducing this weighting term, we account for the fact that results obtained from a certain path orientation may still lead to better estimates that the results from the other orientations. Therefore, we are able to weight each direction accordingly based on the scene.

Similarly, another extension is to replace the constant penalty terms P_1 and P_2 with values that instead change depending on the orientation of the path, $P_1(r)$ and $P_2(r)$. Both these penalties and the weighting values are computed with an evolutionary algorithm trained on an initial set of images and provided ground truth. More specifically, the covariance matrix adaptation evolution strategy (CMA-ES) is used, which is implemented using the open-source Shark library.

The final step is to, for each pixel, iterate over the calculated energy values and determine which disparity value corresponds to the minimum energy. This can be parallelized on the GPU by having each thread handle all disparities for a given pixel. The disparities returned from this step represent the final disparity map.

Milestones Achieved

1. Finished literature review of underlying concepts
2. Reviewed CUDA
3. Collected stereo imagery from datasets (Middlebury, Kitti)
4. Integrated required libraries (CUDA toolkit v7.5, OpenCV 3.1)
5. Built GPU kernel for converting RGB image to grayscale on device
6. Built GPU kernel for calculating initial cost values

Remaining Milestones

1.	Build GPU kernel to handle cost aggregation step	5/21/16
2.	Build GPU kernel for determining optimal disparity values per pixel	5/23/16
3.	Implement CMA-ES for training weight and penalty constants	5/26/16
4.	Test performance on various datasets	5/30/16
5.	Test performance on custom stereo images	6/1/16
6.	Write final report	6/6/16

References

[1] D. Scharstein and R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” International Journal of Computer Vision, vol. 47, pp. 7–42, 2002.

[2] G. Van Meerbergen, M. Vergauwen, M. Pollefeys, and L. Van Gool. A hierarchical symmetric stereo algorithm using dynamic programming. International Journal of Computer Vision, 47(1/2/3):275–285, April-June 2002.

[3] H. Hirschmuller, “Accurate and efficient stereo processing by semiglobal matching and mutual information,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2005, pp. 807–814.

[4] J. Sun, H. Y. Shum, and N. N. Zheng. Stereo matching using belief propagation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(7):787–800, July 2003

[5] M. Michael, J. Salmen, J. Stallkamp and M. Schlipsing, "Real-time stereo vision: Optimizing Semi-Global Matching," Intelligent Vehicles Symposium (IV), 2013 IEEE, Gold Coast, QLD, 2013, pp. 1197-1202.

[6] Y. Boykov, O. Veksler, and R. Zabih. Efficient approximate energy minimization via graph cuts. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(11):1222–1239, 2001.