Homework 3: Disparity

Daniel McArdle
CSE 573: Computer Vision

November 22, 2014

1 Report

1.1 Technique for Matching Textureless Regions

To match textureless regions in stereo images, I would try to use information about the nearest regions that do have textures. By finding the average disparity of the closest textured regions, and by assuming uniform motion, we could compute a best guess for the disparity of the textureless region.

The "Reindeer" image is likely the one with the largest textureless region. Unsurprisingly, this area of the disparity map is quite inaccurate. When the block-matching algorithm encounters a textureless region, any block matches any other block from that region, causing inaccurate calculated disparities.

1.2 MSE vs Window Size

1.3 MATLAB Optimizations

The problems of block matching and the dynamic programming algorithm are quite difficult to disentangle from looping control structures. For some operations performed in the loops, however, I was able to use built-in MATLAB functions for speed.

For example, in `BlockMatch.m`, rather than computing the SSD value with loops, I simply created two submatrices `block1` and `block2`. The SSD value of these blocks is `sum(sum( (block1 - block2) .2̂ ))`.

In `DynamicProg.m`, rather than computing the mean value of each arc with a loop, I opted to use the built-in `mean` function.

1.4 Runtime Analysis

1.4.1 Block Matching

With m rows and n columns, the block matching algorithm iterates over each pixel, $O(mn)$ operations. For each pixel, find the best column disparity, meaning we are performing $O(mn^2)$ operations in total (although we can restrict the range of columns searched, there is no asymptotic difference). Then, for each disparity value tested, we compute the SSD, an operation that takes $O(w^2)$ operations, where w is the window size.

In total, the block matching algorithm is $O(mn^2w^2)$. When the number of rows, m, is increased linearly, the runtime will increase linearly. When either the number of columns, n, or the window size, w, is increased linearly, the runtime will increase quadratically.

1.4.2 Dynamic Programming

The dynamic programming algorithm incorporates the textbook's scanline matching algorithm. This means that we need to run the textbook's algorithm for each of m rows. This gets us to $O(m)$.

For each row, we iterate over the edges from the first image and the edges from the second image. In the pathological worst case, there would be n edges in each image, meaning the algorithm is up to $O(mn^2)$ running time.

For each pair of edges, we consider each of three inferior neighbors to this pair. Three is a constant factor, which can be ignored asymptotically. For each of these neighbors, we compute the "Arc-Cost" of the corresponding two arcs from the first and second images, which is an $O(1)$ operation, since the length of an arc would be 1 if there are n arcs total.

Ultimately, this makes the dynamic programming algorithm have a complexity of $O(mn^2)$, but it is important to remember that the number of edges in a scanline will likely be very small compared the number of columns. So in reality, the number of operations performed will be $O(mk^2)$ where k is much smaller than n.

1.5 Literature Review

For the literature review, I am choosing two papers that focus on accuracy and efficiency, respectively. These are two features that my algorithm implementations are sorely lacking. Interestingly, in the area of disparity matching, there is usually a tradeoff between the two [2].

1.5.1 Paper 1: Efficient Large Scale Stereo Matching (2010)

The main idea of **Efficient Large Scale Stereo Matching** [1] is that many computed stereo correspondences will be ambiguous, but some of them can be "robustly matched." These "support points" are then used to improve the quality of the ambiguous correspondences.

This approach already has a major difference from our block matching algorithm. Our block matching algorithm naievely chooses the disparity that causes a small window of pixels to match best. This is naive because occlusion may cause a window to be impossible to match. This can produce wildly-inaccurate disparity values, which would not have occurred had we used support points.

The algorithm outlined in the paper also takes multiple measures to preserve consistency. Similar to the block matching consistency check, they require that correspondences match left-to-right and right-to-left. They go one step further, though, and "eliminate all points whose ratio beween the best and second best match exceeds a fixed threshold”. In other words, they will not accept possibly-ambiguous disparities.

The algorithm also makes use of a Probabilistic Generative Model that uses support points to actually perform the stereo matching.

1.5.2 Paper 2: On building an accurate stereo matching system on graphics hardware (2011)

This paper [2] was intriguing to me because it presents an algorithm that runs on the GPU. The abstract boasts that it achieves near-real-time performance and is the top performer in the Middlebury benchmark at the time of publication, achieving results in 0.1 seconds.

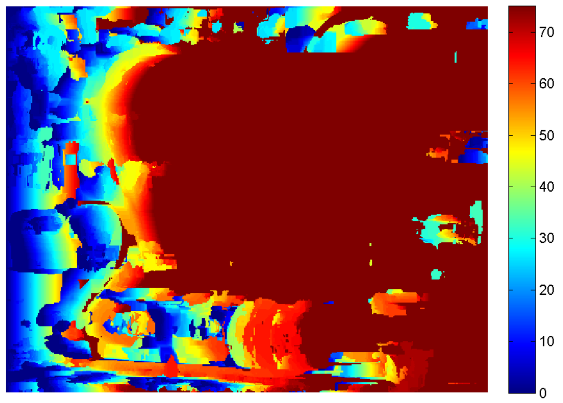
The algorithm uses techniques developed in multiple papers. The key features are Cost Initialization, Cost Aggregation, Scanline Optimization, Disparity Refinement. The final feature is an efficient GPU implementation using CUDA.

Our algorithm would not lend itself well to GPU implementation, as GPUs are highly parallel and our algorithm relies on mutable state.

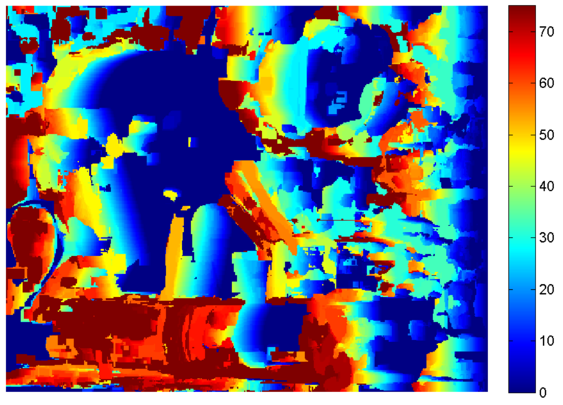
2 Results

2.1 Data

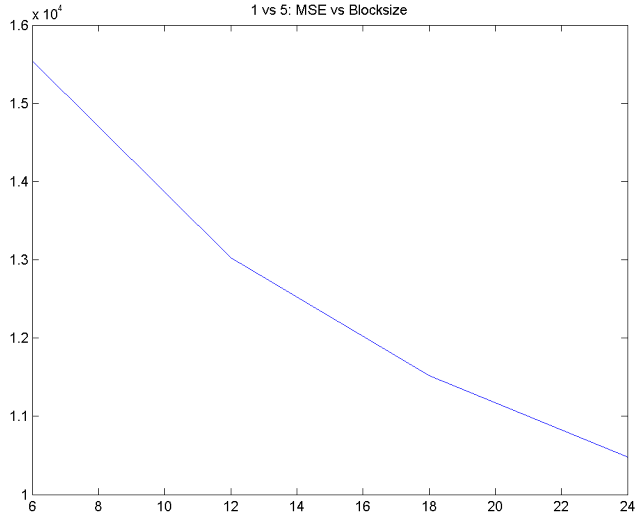
2.1.1 output/Data/disp1.png



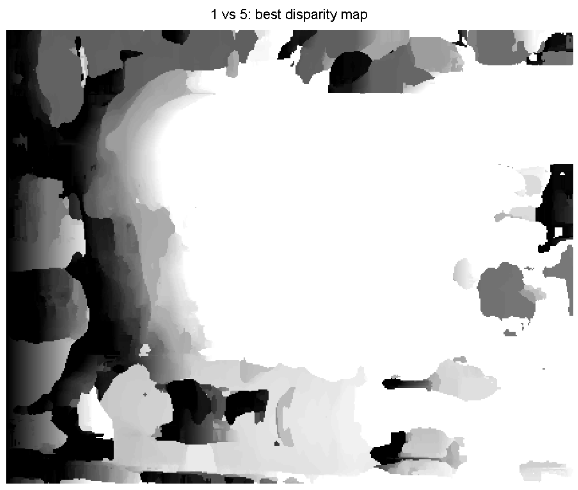
2.1.2 output/Data/disp2.png



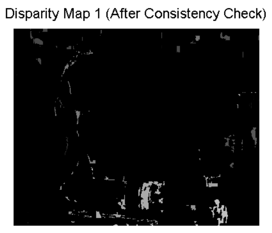
2.1.3 output/Data/mse_vs_blocksize.png

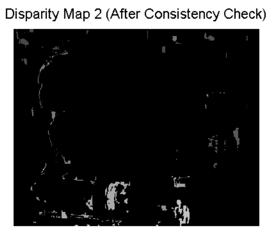


2.1.4 output/Data/best_dispmap.png

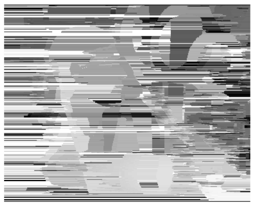


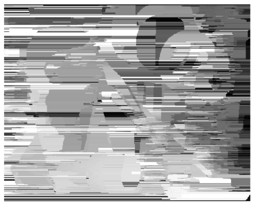
2.1.5 output/Data/after_consistency_check.png





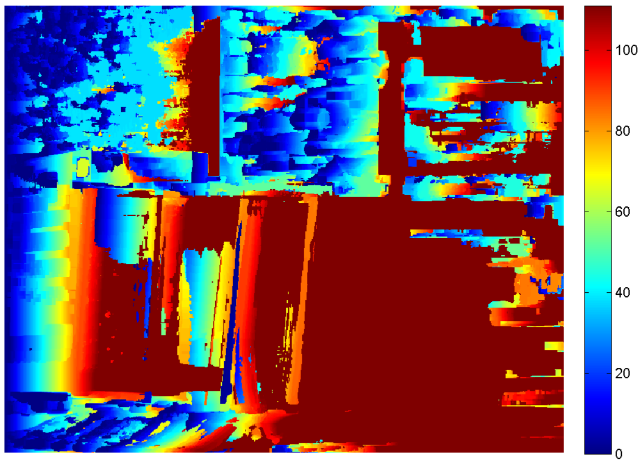
2.1.6 output/Data/dynamic_prog.png



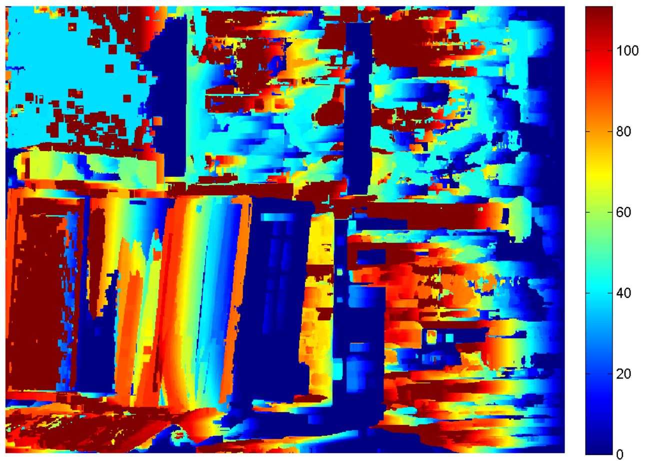


2.2 Books

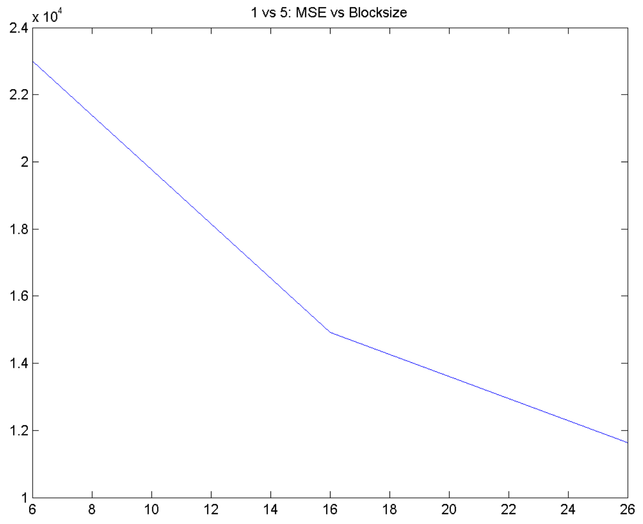
2.2.1 output/Evaluation/Books/disp1.png



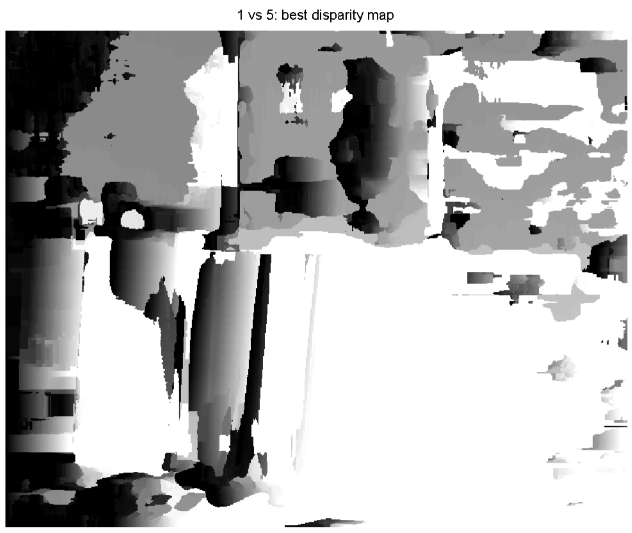
2.2.2 output/Evaluation/Books/disp2.png



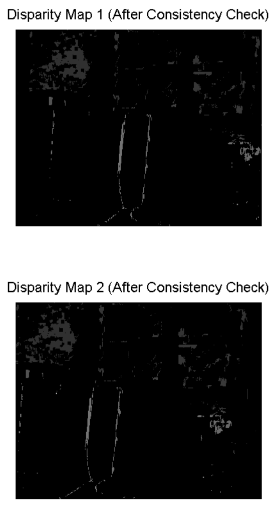
2.2.3 output/Evaluation/Books/mse_vs_blocksize.png



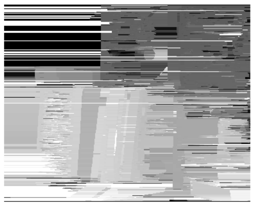
2.2.4 output/Evaluation/Books/best_dispmap.png



2.2.5 output/Evaluation/Books/after_consistency_check.png

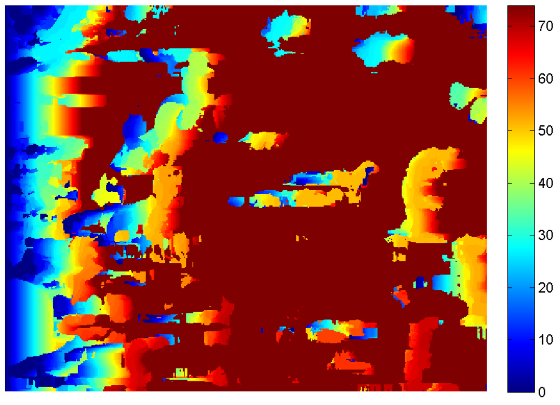

Disparity Map 1 (After Consistency Check)
Disparity Map 2 (After Consistency Check)

2.2.6 output/Evaluation/Books/dynamic_prog.png

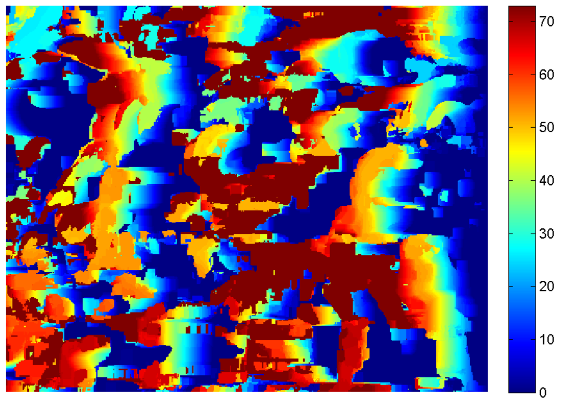


2.3 Dolls

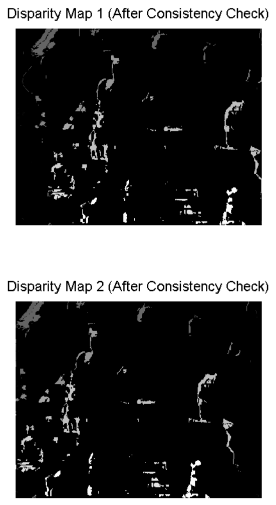
2.3.1 output/Evaluation/Dolls/disp1.png



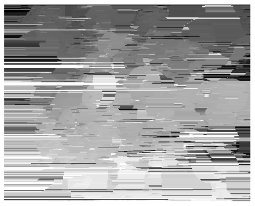
2.3.2 output/Evaluation/Dolls/disp2.png

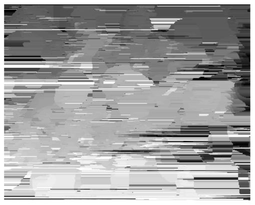


2.3.3 output/Evaluation/Dolls/after_consistency_check.png



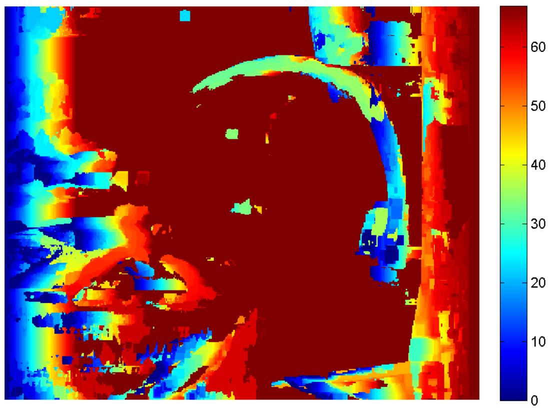
2.3.4 output/Evaluation/Dolls/dynamic_prog.png



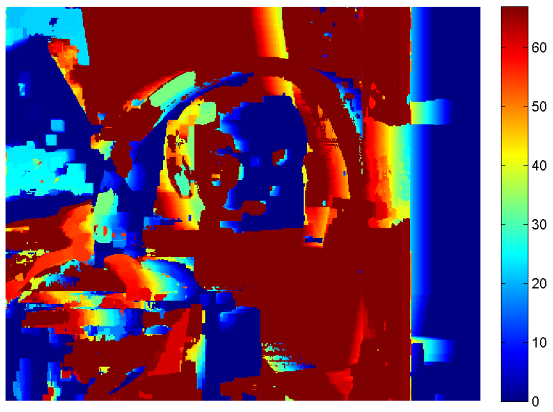


2.4 Reindeer

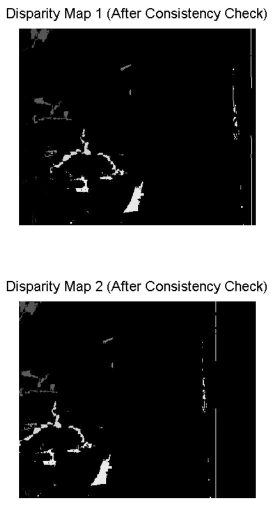
2.4.1 output/Evaluation/Reindeer/disp1.png



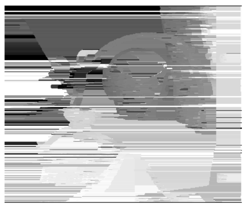
2.4.2 output/Evaluation/Reindeer/disp2.png

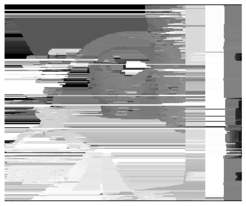


2.4.3 output/Evaluation/Reindeer/after_consistency_check.png



2.4.4 output/Evaluation/Reindeer/dynamic_prog.png





References

[1] Geiger, A., Roser, M., and Urtasun, R. Efficient large-scale stereo matching. In *Computer Vision ACCV 2010*, R. Kimmel, R. Klette, and A. Sugimoto, Eds., vol. 6492 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 25–38.

[2] Mei, X., Sun, X., Zhou, M., shaohui Jiao, Wang, H., and Zhang, X. On building an accurate stereo matching system on graphics hardware. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on* (Nov 2011), pp. 467–474.