Fast Dense Stereo Correspondences by Binary Locality Sensitive Hashing

Philipp Heise¹, Brian Jensen¹, Sebastian Klose¹ and Alois Knoll¹

Abstract—The stereo correspondence problem is still a highly active topic of research with many applications in the robotic domain. Still many state of the art algorithms proposed to date are unable to reasonably handle high resolution images due to their run time complexities or memory requirements. In this work we propose a novel stereo correspondence estimation algorithm that employs binary locality sensitive hashing and is well suited to implementation on the GPU. Our proposed method is capable of processing very high-resolution stereo images at near real-time rates. An evaluation on the new Middlebury and Disney high-resolution stereo benchmarks demonstrates that our proposed method performs well compared to existing state of the art algorithms.

I. INTRODUCTION

Calculating stereo correspondences from stereo image pairs is one of the oldest and still highly active areas in computer vision. Most stereo algorithms are either incapable of real-time processing or are only applicable to small images. Additionally most algorithms require a prior knowledge about the depth of the scene, manifested as fixed range of allowed disparity values. One of the limiting factors for real-time applications is that the complexity of most state of the art algorithms is not only dependent on the width \(w\) and height \(h\) of the images but also on the number of possible disparity labels \(d\) leading to a complexity \(O(whd)\).

We propose to solve the stereo correspondence problem by applying hashing leading to a complexity \(O(wh)\) that is only dependent on the image size and not on the number of disparity labels. We use simple binary strings generated from simple intensity tests similar to the BRIEF descriptor from Calonder et al. [6] to perform the hashing. Under mild assumptions about the intensity tests we can show that we can get the same results as with performing an exhaustive search with arbitrary high probabilities. In practice the hashing approach allows us to get reasonable disparity maps even from high definition images at interactive rates. To the best of our knowledge such a hashing scheme has not yet been used to perform stereo matching.

A. Related work

Existing state of the art stereo matching algorithms can be generally classified into three distinct categories. Local methods perform data association on a per pixel basis combined with an aggregation step with winner takes it all disparity label selection [18]. Examples for local methods are algorithms using adaptive window weighting [14], [22], which have been shown to give very good results and mitigate the problem of edge bleeding to some extent, which is typical to pure block matching algorithms. Local methods are often the only available choice for applications that need real-time or interactive frame rates. Their runtime complexity is dominated by the evaluation of the likelihood function, which itself is dependent on the number of disparity labels for most algorithms. Global methods perform a global minimization of an energy formulation that incorporates prior knowledge into the disparity estimation by means of regularization of the data term with a smoothing term. The smoothing term models the fact that neighboring pixels should have similar disparity labels and are very likely to belong to the same surface. The graph-cuts algorithm of Boykov et al. [5] and belief propagation [8] are well known methods belonging to this category as well as variational formulations mainly used for the related problem of optical flow [24]. The computational and memory requirements of global methods are very high and generally prohibit their usage for real-time applications. The third category are the so called seed and grow approaches like the sampling based methods by Cech et al. [7] and the PatchMatch stereo algorithm by Bleyer et al. [4], itself based on the general PatchMatch correspondence algorithm by Barnes et al. [2]. While sampling can be very fast if only a fraction of the candidates is considered, no guarantees can be given compared to an exhaustive search and the methods also suffer from the same problems in textureless and ambiguous regions as the local methods. Our method does not fit into one of these main categories but is related to the sampling based approaches as well as the local methods. In contrast to the sampling based methods our method can get identical results to methods performing an exhaustive search of the possible disparity label space with very high probability while being independent of the size of the disparity range and only a fraction of the possible correspondence candidates need to be considered. Therefore our algorithm belongs to the broad class of approximate nearest neighbour (ANN) approaches. Most ANN algorithms in the area of computer vision are used for fast descriptor matching [19] and establishing general image correspondences [1], [16], [10]. Many of the algorithms use tree based data structures like KD-Trees [10] to speed up the matching process or are based on randomized sampling and propagation, making use of coherency [1]. Another approach is hashing with the intention that items with small distances are producing collisions and are therefore hashed into the same bucket [13]. For the purpose of establishing
dense image correspondences the objective is very often to approximate one image with another image and to achieve a high image reconstruction quality e.g. for compression. For stereo correspondences the overall aim is also to establish correspondences, but here the focus is on finding the real scene depth and a single correct correspondence needs to be found for each pixel. Our algorithm is based on Locally Sensitive Hashing [13] and we pair it with the Hamming distance as our distance measurement. In order to work in the Hamming space our work also uses the concept of simple image intensity tests to generate a compact representation, which has been successfully used in form of the CENSUS likelihood [23] for many stereo algorithms and as a very compact feature descriptor referred to as BRIEF by Calonder et al. [6]. The binary strings generated by randomized tests have already been used in [25] as a likelihood function for binary stereo matching and their local method was shown to be comparable with other state of the art methods in terms of quality and speed while being robust to radiometric differences between the image pairs.

B. Contributions

We propose to use dense binary strings similar to the BRIEF descriptor [6] combined with a hashing scheme based on LSH [13]. We contribute an extremely fast parallel algorithm and an opensource GPU implementation that has a complexity $O(wh)$ only depending on the width $w$ and height $h$ of the image and independent of disparity range. Further we show that under mild assumptions results equivalent to exhaustive search within a disparity range with arbitrary high probability can be achieved. The trade-off between exactness and speed can therefore be precisely controlled in our algorithm. The proposed algorithm is further shown to be extremely fast on high-resolution images several megapixels in size. We show that the algorithm is well suited to calculate disparity maps at interactive frame rates for high-resolution images while being a viable approach for systems with limited computational and memory resources.

II. Binary Stereo Hashing

Our algorithm uses binary strings for each pixel and computes the likelihood in terms of the hamming distance between two possible correspondences. Therefore the likelihood directly depends on the binary string and we follow the general approach used for the binary BRIEF descriptor by Calonder et al. [6]. We perform simple binary comparisons between pixel positions on a previously filtered image $I$. Our binary string $B$ of length $N$ is densely calculated for each pixel position $u = (x, y)^T$ using two dimensional offsets $o_{i,1}, o_{i,2}$ for each bit $i$ generated in an offline operation. The number of bits $P$ determines the number of buckets to be $2^P$. In general we are interested in having a large number of buckets in order to keep the average fillrate per bucket $\frac{w^2}{2^P}$ low. In reality the fillrate highly depends on how much the tests correlate and if the tests are able to allow the distinction between visually similar regions e.g. untextured regions. Instead of performing the hashing only once we hash $N$ times. Figure 2 gives an overview of the hashing procedure.

![Figure 2](image-url)

Figure 2. Overview of the hashing procedure: for each scanline the $x$ position of each pixels is hashed into $N$ hashtables with $2^P$ buckets using $P$ randomly selected bits from the bit string $B$. The figure illustrates this for two pixels marked in green and blue of one scanline marked by a red line.

The exact algorithm for the parallel hashing procedure is given in algorithm 1. Although the overall memory require-
ments for the offsets are known \( w \times h \times N \), the exact size for each of the buckets of one scanline is unknown. Therefore the algorithm 1 is split into 3 phases. In the first phase the size of each bucket is determined in lines 4–8 using the atomic increment operation ATOMIC_INC. The previously calculated bucket size is then used to calculate the exact offsets for each bucket for all the scanlines in parallel (lines 9–14). In the remaining lines 15 to 19 the real \( x \) positions of the pixels hashing into certain buckets are written into the previously determined offset positions. A size counter keeps track of the current sub offset within one bucket, which is also atomically incremented after each addition of an element. The first and last phase are completely performed in parallel for each pixel.

**Algorithm 1 Parallel Stereo Hashing**

Require: Binary string \( b = B(x,y) \) for each pixel position \( x, y \), \( P \) random bit positions \( p_i \) for each of the \( N \) hash functions \( h_i \)

```
1: function BINARY_HASH\((b, p, P)\)
2: return \( \sum_{i=0}^{P-1} ((b \gg p[i]) \& 1) \cdot 2^i \)
3: function STEREO_HASH\((B,N)\)
4: Determine the size of each bucket
5: for all pixel positions \( x, y \) in parallel do
6: for \( i = 1 \) to \( N \) do
7: \( h_i \leftarrow \text{BINARYHASH}(B(x,y), p_i, P) \)
8: ATOMIC_INC\((\text{buck}_i[y][h_i].\text{len})\)

▷ Use the previously estimated size to calculate offsets for each bucket
9: for all scanlines \( y \) in parallel do
10: for \( i = 1 \) to \( N \) do
11: for \( b = 2 \) to \( 2^P \) do
12: \( \text{buck}_i[y][b].\text{off} \leftarrow \text{buck}_i[y][b - 1].\text{off} + \text{buck}_i[y][b - 1].\text{len} \)
13: \( \text{buck}_i[y][b - 1].\text{len} \leftarrow 0 \)
14: \( \text{buck}_i[y][2^P - 1].\text{len} \leftarrow 0 \)

▷ Use the offsets to store the \( x \) position of each candidate
15: for all pixel positions \( x, y \) in parallel do
16: for \( i = 1 \) to \( N \) do
17: \( h_i \leftarrow \text{BINARY_HASH}(B(x,y), p_i, P) \)
18: \( l \leftarrow \text{ATOMIC_INC}((\text{buck}_i[y][h_i].\text{len}) \)
19: \( M_i(y, \text{buck}_i[y][h_i].\text{off} + l) = x \)
20: return Buckets \( \text{buck}_i \) and maps \( M_i \)
```

**B. Stereo Hash-Matching**

For the matching process given in algorithm 2 the binary strings must also be generated from the intensity comparisons for the second image and also the hashing has to be performed in an identical manner. But instead of storing possible candidate positions in hash tables we use the already available positions and evaluate the hamming distance between the complete binary strings referred to by the \( x \) positions stored inside the buckets for each of the \( N \) hash values. Offsets values that would lead to negative disparity values are not considered. The same skipping could also be done for a maximum allowed disparity value. The function HAMMING is used in the pseudo-code and computes the hamming distance between two binary strings. Besides the hamming distance any other likelihood function could be used e.g. SAD, SSD or NCC. We decided to use the hamming distance to avoid further image lookups and to take advantage of the compact size of the bit strings.

**Algorithm 2 Parallel Stereo Matching with Hashing**

Require: Binary strings for right and left \( b_2 = B_2(x,y), b_1 = B_1(x,y) \) for each pixel position \( x, y \), \( P \)
random bit positions \( p_i \) for each of the \( N \) hash functions \( h_i \), the hashmap buckets \( \text{buck}_i \) and the maps \( M_i \)

```
1: function STEREO_HASH_MATCH\((B_1, B_2, \text{buck}, M, N)\)
2: for all pixel positions \( x, y \) in parallel do
3: \( \text{match} \leftarrow x \)
4: \( \text{dist} \leftarrow P \)
5: for \( i = 1 \) to \( N \) do
6: \( h_i \leftarrow \text{BINARY_HASH}(B_2(x,y), p_i, P) \)
7: for \( i = 1 \) to \( \text{buck}_i[y][h_i].\text{len} \) do
8: \( \text{cmatch} \leftarrow M_i(y, \text{buck}_i[y][h_i].\text{off} + i) \)
9: if \( \text{cmatch} > x \) then
10: \( \text{cdist} \leftarrow \text{HAMMING}(B_1(\text{cmatch}, y), B_2(x,y)) \)
11: if \( \text{cdist} < \text{dist} \) then
12: \( \text{match} = \text{cmatch} \)
13: \( \text{dist} = \text{cdist} \)
14: \( D(x,y) = \text{match} - x \)
15: return \( D \)
```

**C. Matching Probabilities and Parameter Settings**

In the following we assume that the probability \( p_{\text{hit}} \) of a bit resulting from an intensity test for truly corresponding pixels is independent for each of the tests and identically distributed. The probability that at least one of the \( P \) bits chosen for hashing is flipped, is therefore given by

\[
p_{\text{fail}} = 1 - p_{\text{hit}}^P. \tag{3}
\]

The probability \( p_{\text{hit}} \) tells us how likely it is that two truly corresponding pixels are hashed into two different buckets. On the other side we hash not only once but \( N \) times. Also here we assume that all \( N \) tries are independent and identically distributed. The probability that at least one of the \( N \) hashing tries succeeds is given by

\[
p_{\text{onehit}} = 1 - \left( 1 - p_{\text{hit}}^P \right)^N. \tag{4}
\]

The resulting probabilities are highly dependent on the stability of the intensity test and therefore the probability \( p_{\text{hit}} \). The tests also need to be uncorrelated, otherwise many
pixels will be hashed to the same bucket and in the worst case all pixels are hashed into the same bucket. In Figure 3 three plots for $p_{bit} \in \{0.6, 0.75, 0.9\}$ and varying values of $P, N$ are shown. As shown in the figure for reasonable values for a bit flip the probability soon becomes very small and given the bit flip probability we can easily estimate the number tries necessary to reach a certain probability level that values are hashed into the same bucket. Additionally it becomes obvious that the number of bits $P$ used for hashing can not be arbitrarily high. The usage of too many bits $P$ decreases the hit probability and much more hashing tries need to be performed to get the same overall hit probability.

Of course no guarantee can be given that the real candidate has also the lowest distance in terms of the hamming distance for the complete string.

### III. Evaluation

The goal of our algorithm is to show that correspondences can be found extremely fast by employing the hashing scheme. We do not perform any complex post-processing of our raw disparity maps with the exception of a simple $3 \times 3$ median filter. For all results we used $P = 8$, $N = 8$ and 256 intensity tests. The input images were blurred with a simple Gaussian with a sigma value of 0.5 in the horizontal direction and a sigma value of 2.5 in the vertical direction. We choose the values for $P = 8$ and $N = 8$ after having evaluated a vast number of combinations on the Middlebury 2014 [17] dataset. The combination $P = 8$, $N = 8$ gave good results and was still very fast. Figure 4 shows other combinations of $P/N$ and their respective error rates and processing times. Some combinations have even lower runtimes, nearly twice as fast, but with worse error rates. Increasing the number of hashing tries $N$ above a certain point only leads to a very minor improvement of the error rate.

For the complete evaluation we used the images and evaluations of other algorithms provided by Sudipta et al. [20]. We used the images of the group MiddNew7 from the new 2014 Middlebury public stereo dataset [17] and the images from the group Disney4 from the high-resolution multi-baseline datasets used in [15]. The images in the MiddNew7 group have an average size of 5.5 megapixel and the images in the Disney4 group an average size of 10 megapixels. We also used the error rates and runtimes reported in [20] to compare our results with SGM [12], SGM-HH an optimized version by the author, PatchMatch [4], ELAS [9] and the LPS algorithm [20].

As expected from a purely local method without any post-processing our algorithm does not perform as well as most of the others in terms of the error rate as shown by the graphs in Figure 5. On average our method still outperforms the PatchMatch algorithm and also SGM for the Disney dataset as measured by the percentage of disparity estimates with an error greater than one. In terms of error vs. runtime the proposed approach is very attractive and we are able to process the megapixel stereo pairs at interactive frame rates. As shown by the scatter plot in Figure 6 the proposed method is by far the fastest method.

In Figure 7 the results of the proposed method for the MiddNew7 group of images are shown.

### IV. Conclusion

We have presented a simple and extremely fast stereo processing algorithm that uses locality sensitive hashing of binary strings to become independent of the number disparity...
Figure 5. Percentage of disparity errors > 1.0 for all algorithms for the new Middlebury and the Disney dataset.

Figure 6. Scatter plot of error in percent versus time in seconds in logarithmic scale for all algorithms for the new Middlebury and the Disney dataset.

labels. The method is capable of handling high definition images in near real time while offering a competitive tradeoff between speed and accuracy. In domains where real time processing is essential the proposed approach alone is a viable approach to handle high definition stereo images. In applications where accuracy is crucial the proposed method can be easily used in tandem with a global method [4], [3], [11], [20] as part of the initialization phase, in place of the two typical choices of initialization: random sampling and sparse correspondences.

In our current implementation a randomly selected sampling pattern is employed. Learning an optimized pattern for stereo matching could lead to a further improvement in accuracy and has been demonstrated for general purpose feature descriptors in [21]. Also a more detailed evaluation of the blurring and intensity test dependent blurring, together with the correlation between the tests, is worth pursuing. Further the proposed approach is not limited to the hamming distance between the binary strings as a likelihood function. Since only positions of possible candidates are stored in the hash tables, arbitrary likelihood functions could be used for matching and evaluation of possible candidates. Left/right consistency checking and more sophisticated post processing strategies [26] could also be used to improve the matching confidence and quality.

ACKNOWLEDGMENT

The authors would like to thank Sudipta Sinha for making the detailed evaluation results of the various stereo algorithms available. This research was supported in part by a grant from the German Science Foundation (DFG) for the research project FOR 1321 Single-Port Technology.

REFERENCES

Figure 7. Results of the proposed method for the MiddNew7 set of images.


