 DOI: 10.3217/978-3-85125-528-7-15

On a Fast Implementation of a 2D-Variant of Weyl's Discrepancy Measure

Christian Motz[1] , Bernhard A. Moser[1]

Knowledge-Based Vision Systems
Software Competence Center Hagenberg, Austria
christian.motz@scch.at, *bernhard.moser@scch.at*

Abstract
Applying the concept of Hermann Weyl's discrepancy as image similarity measure leads to outstanding robustness properties for template matching. However, in comparison with standard measures this approach is computationally more involving. This paper analyzes this measure from the point of view of efficient implementation for embedded vision settings. A fast implementation is proposed based on vectorization of summed-area tables, resulting in a speed-up factor 16 compared to a standard integral image based computation.

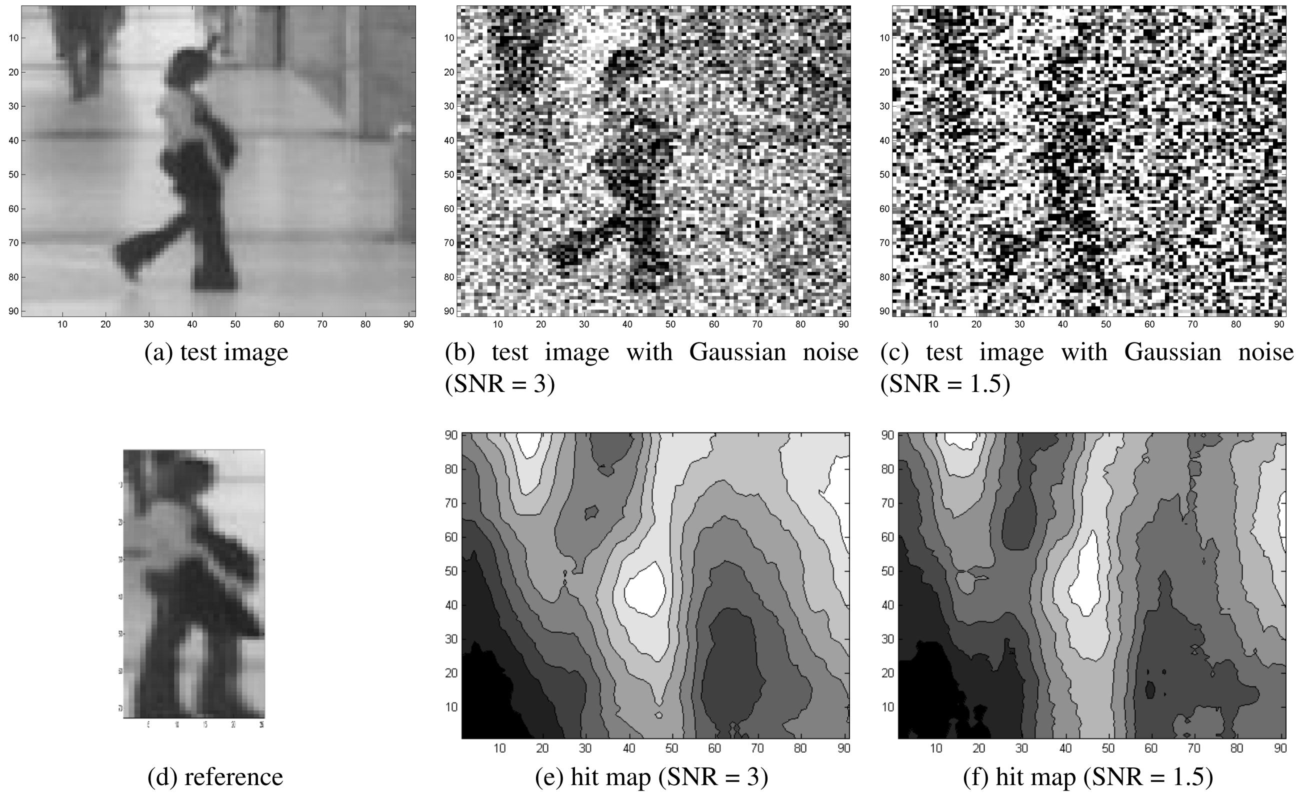
1. Introduction

In this paper we take up a novel concept of similarity measure due to [1] and investigate its applicability for the requirements of embedded vision. The core idea of this measure is its design principle based on a family of subsets rather than evaluating the aggregation of point-wise comparisons on a pixel-by-pixel level. In contrast to pixel-by-pixel based approaches with subsequent commutative aggregation such as mutual information of normalized cross correlation the subset-based approach also takes spatial arrangements into account which makes this approach interesting for pattern analysis and matching purposes [2].

This measure goes back to H. Weyl already 100 years ago and was studied in the context of evaluating the quality of pseudo-random numbers and measuring irregularities of probability distributions [3]. For one-dimensional signals (vectors) it is defined as

$$\|(x_1, \ldots, x_n)\|_D = \max_{1\leq a,b\leq n} |\sum_{i=a}^{b} x_i| = \max_{r}\{0, \sum_{i=1}^{r} x_i\} - \min_{s}\{0, \sum_{i=1}^{s} x_i\}$$

Interestingly, this measure not only plays a central role in discrepancy theory which is related to low complexity algorithmic design by means of low discrepancy sequences [4], but as found out recently, also in other fields of applications, e.g. in event-based signal processing [5, 6], random walk analysis [7] and image and volumetric data analysis by extending it to higher dimensions by means of integral images [1]. As pointed out in [1] the extension is not unique. A possible extension is given by Equation (1).



(a) test image (b) test image with Gaussian noise (SNR = 3) (c) test image with Gaussian noise (SNR = 1.5)

(d) reference (e) hit map (SNR = 3) (f) hit map (SNR = 1.5)

Figure 1. Illustration of a pattern matching problem: find reference image in the test image. The images are taken from frame 697 and frame 705 of the EC Funded CAVIAR project/IST 2001 37540 ("Shopping Center in Portugal", "OneLeaveShop2cor"). The hit map is computed using the measure (1)

$$
\begin{aligned}
\|f\|_D := \max \Big\{ & \max_{0\le k\le N, 0\le l\le M}\{\sum_{i=0}^{k}\sum_{j=0}^{l} I_{(i,j)}\} - \min_{0\le k\le N, 0\le l\le M}\{\sum_{i=0}^{k}\sum_{j=0}^{l} I_{(i,j)}\}\ , \\
& \max_{0\le k\le N, 0\le l\le M}\{\sum_{i=0}^{k}\sum_{j=0}^{l} I_{(N-i,j)}\} - \min_{0\le k\le N, 0\le l\le M}\{\sum_{i=0}^{k}\sum_{j=0}^{l} I_{(N-i,j)}\}, \\
& \max_{0\le k\le N, 0\le l\le M}\{\sum_{i=0}^{k}\sum_{j=0}^{l} I_{(i,M-j)}\} - \min_{0\le k\le N, 0\le l\le M}\{\sum_{i=0}^{k}\sum_{j=0}^{l} I_{(i,M-j)}\}, \\
& \max_{0\le k\le N, 0\le l\le M}\{\sum_{i=0}^{k}\sum_{j=0}^{l} I_{(N-i,M-j)}\} - \min_{0\le k\le N, 0\le l\le M}\{\sum_{i=0}^{k}\sum_{j=0}^{l} I_{(N-i,M-j)}\} \Big\}
\end{aligned} \quad (1)
$$

For registration and template matching purposes the discrepancy measure is applied on the difference of the corresponding images. To be more precise, the images are considered as two-dimensional functions on the lattice of integers with with default values 0 outside the proper frame of the images.

This measure satisfies the following desirable registration properties (see also [8, 9]):

[R1] a vanishing distance entails a vanishing extent of misalignment and vice versa,

[R2] the distance measure behaves continuously at least with respect to arbitrary small misalignments,

[R3] an increasing extent of misalignment implies an increasing distance measure and vice versa (monotonicity).

It is interesting that it can be shown that these natural properties are not satisfied simultaneously by commonly used matching and registration techniques [1]. Figure 1 illustrates its robustness by applying this measure as fitness function for finding the best match between a reference and a test image. In this demonstration the discrepancy measure is directly applied without any image preprocessing or denoising. As this measure relies only on the evaluation of integral images and max/min operations, it is well-suited for parallelization. An efficient implementation can be tackled by means of of the concept of a summed-area table [10] which is a matrix generated from an input image in which each entry in the matrix stores the sum of all pixel values between the entry location and the lower-left corner of the input image. For applications of summed-area table see also [11] and related concepts based on integral images e.g. [12]. The power of the summed-area table comes from the fact that it can be used to perform filters of different widths at every pixel in the image in constant time per pixel. This makes SAT very useful for embedded vision purposes.

The paper is organized as follows: Section 2. introduces a two-dimensional definition of the discrepancy norm and makes algorithmic optimizations to reduce computation effort. Section 3. presents a vectorization concept for the previously optimized algorithm. Section 4. presents the speedup achieved by the optimizations.

2. Algorithmic Analysis for Implementation

While in the 1-dimensional case partial sums over intervals are evaluated in the 2-dimensional case rectangles are taken instead of intervals. As shown in [1] is suffices to restrict on the rectangles with one corner being coincident with a corner of the image. This suggests to use integral images spreading of each of the four corners of the image. However, a single integral image already contains the information of the remaining integral images from the other corners. This leads to the first optimization step by deducing the values for the four integral images from the original integral image with the top left corner as the starting point. Assume point P_1 is our current index. The value at this position naturally corresponds with the first integral image in the definition of discrepancy norm. The second integral image in the definition has the top right corner as a reference. This corresponds to area II_2 in the figure, which can be computed by subtracting sums $P_2 - P_1$. The third and fourth integral images are very similar. Equations (2) to (3) provide a mathematical formulation of the integral image

transformations as explained above:

$$\begin{aligned}
\Pi_1(x,y) &= \Pi(x,y) \qquad (2)\\
\Pi_2(x,y) &= \begin{cases} \Pi(W,y)-\Pi(x,y) & \text{if } x \neq W,\\ \Pi(W,y) & \text{if } x = W. \end{cases}\\
\Pi_3(x,y) &= \begin{cases} \Pi(x,H)-\Pi(x,y) & \text{if } y \neq H,\\ \Pi(x,H) & \text{if } y = H. \end{cases}\\
\Pi_4(x,y) &= \begin{cases} A(x,y) & \text{if } x \neq W, y \neq H,\\ \Pi(x,H)-\Pi(x,y) & \text{if } x = W, y \neq H,\\ \Pi(W,y)-\Pi(x,y) & \text{if } x \neq W, y = H,\\ \Pi(x,y) & \text{if } x = W, y = H. \end{cases}\\
A(x,y) &= \Pi(W,H)+\Pi(x,y)-\Pi(W,y)2-\Pi(x,H) \qquad (3)
\end{aligned}$$

W indicates the last valid x index of a row and H the last valid y index of a column. Care has to be taken, if any index lies on the edge: Here, some components simply refer to the same area and are equal.

Now, we make use of transformations which the summation terms are invariant to: we are interested in the difference between maximum and minimum of the summation. Thus, adding a constant factor to all elements within the integral image will not affect the difference between maximum and minimum. When it comes to (3), the term $\Pi(W,H)$ clearly is constant, neither depending on index variable x nor on y. As a result, it can be omitted for the discrepancy norm calculation. Note that this needs to be compensated in the others cases of Π_4, too. Taking the constraint for the indexes into account, Equations (4) and (5) are obtained:

$$\tilde{\Pi}_4(x,y) = \begin{cases} \tilde{A}(x,y) & \text{if } x \neq W, y \neq H,\\ -\Pi(W,y) & \text{if } x = W, y \neq H,\\ -\Pi(x,H) & \text{if } x \neq W, y = H,\\ 0 & \text{if } x = W, y = H. \end{cases}, \qquad (4)$$

$$\tilde{A}(x,y) = \Pi(x,y)-\Pi(W,y)-\Pi(x,H). \qquad (5)$$

2.1. Reducing the compare operations

So far, the discrepancy norm in 2D has been reduced to computing a single integral image and expressing the other forms based on this single one. It still requires 8 compare operations per pixel: one for the minimum, one for the maximum and this has to be done four times for the different components. Some of these operations are redundant if we concentrate on a specific row, meaning y is constant. Applying this method to Π_2 of Equation (2), we obtain equations (6) and (7) which differ only by a constant and a sign. The negative sign will swap minimum and maximum. As a result, the second component can be deduced with simple operations that are only necessary at the end of each

row:

$$\Pi_{p1}(x) = \Pi_p(x) \tag{6}$$

$$\Pi_{p2}(x) = \begin{cases} \Pi_p(W) - \Pi_p(x) & \text{if } x \neq W, \\ \Pi_p(W) & \text{if } x = W. \end{cases} \tag{7}$$

Equations (8) and (9) provide a mathematical formulation of this, in both cases c corresponds to the constant $\Pi_p(W)$ per row. In other words, we only need to compute a single integral image, and compute the minimum and maximum per row, by which we have half of the computation done to get the discrepancy norm according to (1):

$$\max\{\Pi_{p2}(x)\} = \max\{c, c - \min\{\Pi_{p1}(x)\}\}, \tag{8}$$

$$\min\{\Pi_{p2}(x)\} = \min\{c, c - \max\{\Pi_{p1}(x)\}\}. \tag{9}$$

The third component behaves similar to the second one — the only difference is that the constant now is per column and we need the maximum and minimum for each column. Unfortunately, the fourth and last component is more complex. Here, each value is based both on the last value per column and the last value per row. The problem is that for normal maximum or minimum we only take care of numbers that are larger or smaller but not the equal ones. Yet, for the problem mentioned above, we would need all maximized subexpressions with the same value and their corresponding position consisting of the x and y index pair. Further research is necessary to check whether the minimum and maximum of the fourth component could be determined in a more convenient and less complex way. However, a subexpression refers to the third component and only one addition is necessary to get the fourth component.

2.2. Proposed algorithm

Based on the previous findings, we will now consider the complete algorithm and compare it to the base implementation in terms of runtime complexity. The base algorithm consists of four passes over the data; each will compute one integral image component and, simultaneously, yield minimum and maximum by compare operations. The optimized version consists of only two passes. The first pass will calculate one integral image and get the first and the second component at the same time. Here, the second component needs a small overhead at the end of each row. The second pass will deduce the third and fourth component based on the previously computed integral image.

Basically, both versions have $\mathcal{O}(n \cdot m)$ complexity, where n is the image width and m the image height. If we take a closer look at it, the proposed version has $\mathcal{O}(2n \cdot m)$, compared to the initial $\mathcal{O}(4n \cdot m)$. The optimized version will show further improvements if we consider the number of operations more precisely. Additions and subtractions will be considered as the same operation from the view of complexity. The base version has four very similar passes, consisting of integral image computation and comparisons. Computing an integral image point takes three additions, though, there is an optimized version needing only two which requires extra storage for cumulative row sums [13]. The reference version from [14] is implemented with three operations and will be used for real performance comparison.

Table 1 compares both version in terms of the overall operation count. Additions are reduced heavily down to less than a half. Comparisons are brought down by a fourth approximately. As each pass consists of a double-nested loop that produces overhead, the column *Passes* is very important. Another

	Passes	*Additions*	*Comparisons*
Ref	4	$12n \cdot m$	$8n \cdot m$
Opt	2	$5n \cdot m + 2m$	$6n \cdot m + 4m$

Table 1. Comparing the number of operations for the reference and the optimized version.

factor is storage, given the fact that both versions require additional storage of $n \cdot m$. But the base version writes four times to this area, whereas the optimized version only once. If this storage area is accessible to the user, calculating the discrepancy norm in 2D always yields the according integral image for free.

3. Parallelization

The proposed algorithm seems to be well-suited for parallelization methods. Computing and comparing the other components of the discrepancy norm is highly independent. When it comes to parallelization, modern computers offer various options. A common classification in this area comes from [15]. The classification is based on the number of parallel instruction and data streams. A traditional processor belongs to SISD, whereas multi core or multi processor systems are MIMD. Instruction set extensions like SSE and AVX, also referred to as vector units, belong to SIMD.

A similarity measure like the discrepancy norm will normally be applied many times. Pattern matching requires evaluating the discrepancy norm at many different positions of a patch. Therefore, SIMD is a promising approach. It is especially suitable for applying the same kind of operation to several data values at once. Furthermore, SIMD means choosing certain special instructions. At runtime, they do not have any overhead, compared to normal SISD instructions. On the other hand, making use of multiprocessing would lead to an overhead due to the fact that it involves spanning threads, distributing data and synchronizing at the end. As shown by [16], using multi core processors is complex. On the one hand, the work succeeded in using multiple cores to improve performance. On the other, hand the processor topology has an impact. The authors had to bind the threads to cores sharing the same L2 cache in order to improve performance. Not fulfilling this requirement results in a significant performance penalty.

SIMD instructions operate on a dataset or a so called vector. For example, a traditional add would perform $a := a + b$. The SIMD version of this instruction would perform the same operation, but a would be a vector. Typical vector sizes of SIMD units range from 2 to 8 elements. Normally, vector units have registers of a fixed size. Depending on the size of the data type, they can process a certain amount of elements in one step. Vector units are not designed to operate horizontally, which would mean combining elements within a vector register. We will concentrate on the common SIMD extensions for the x86/x64 architecture. There are two extensions in this area: SSE and AVX - both exist in different versions, with each new version extending the previous one by adding new computing capabilities [17].

AVX doubled the vector size comparesd to SSE. Yet, in terms of data shuffling, the situation became much more complex: with vector registers and operations split into lanes, one AVX register consists of two 128-bit lanes which simplified implementing the architecture for the designers. It does not make any difference for vector operations like additions. Nevertheless, for instance, the SSE shuffle operation takes an immediate value that allows indexing of up to four elements. The according AVX

input	v_3	v_2	v_1	v_0
shuffle	v_2	v_3	v_0	v_1
max	$\max_{32}$	$\max_{32}$	$\max_{10}$	$\max_{10}$
shuffle	$\max_{10}$	$\max_{10}$	$\max_{32}$	$\max_{32}$
max	$\max_{3210}$	$\max_{3210}$	$\max_{3210}$	$\max_{3210}$

Table 2. Getting minimum / maximum of a vector register holding 4 elements.

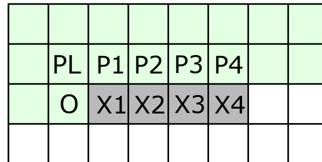


Figure 2. Dependencies for computing new values for an integral images with vector units.

instruction has the same indexing capabilities but operates on a doubled data amount. The AVX instruction simply takes the immediate value and applies the shuffle for each lane. Only a small number of special instructions allow crosslane data exchange in some restricted ways [17, Volume 1 Chapter 14].

3.1. Vectorization

Taking into account the points mentioned above, we will now develop a vectorization scheme. Using vector units for the comparisons is straight forward. Vector units will help us to compare n elements at once. Finally, we have to get the maximum and minimum of the vector itself, which results in overhead because of operating horizontally. To be precise, additional comparisons of $\log n$ are necessary and the same amount of data permutations. The basic idea is to compare pairs of values and then use the result again for pairwise comparisons but with half the number of pairs. Vector units permit comparing several pairs with a single instruction at the same time. Data shuffling assures we are comparing different pairs in the next step. Table 2 illustrates the procedure for a vector unit holding 4 elements.

More complicated is the vectorization of computing the integral image, which can be interpreted as a 2D version of the prefix sum. [18] provides a good summary of prefix sums in general, their applications and a parallel version. The proposed parallelization model is well suited for using GPU acceleration. This was proven by [19]. On the other hand, the GPU version turned out to be only useful for large dataset. Moreover, this was tested for traditional prefix sums and not for 2D versions suitable for integral images which would consist of two passes: one prefix sum over the rows, a second one over the columns, taking the result of the first pass as the input. A similar two-pass algorithm for integral images using GPU was developed and tested by [20]. A notable speedup was not gained for images with a pixel count less than 0.5 million. Furthermore, the data transfer to and from the GPU was not included in the time measurement. Using SIMD extension for integral image computation is a quite new approach, leading to the fact that there are few literature reference that use SIMD extensions. [16] applies SSE for a part of the computation algorithm. Nonetheless, finally, this version is slower than a sequential algorithm presented in the same work. We will show an implementation consisting of a single pass instead of two traditional prefix sum passes. Figure 2 shows the dependencies for computing the new pixels X1 .. X4 in the integral image. All new pixels

have the same row offset (O) and the same compensation factor PL, which is optimal for a vector unit. The same is true for adding the different previous values P1 .. P4. Using PL and O for all vector elements requires one additional instruction to broadcast the single value to all vector elements. We will refer to the summing of X1 to X4 as partial prefix sum. The approach is similar to horizontal minimum / maximum within a vector register. The difference is that a shuffle would not help, but shifting solves the problem. Table 3 lists the single steps. Two additions and two shifts are necessary. This definitely is an improvement over [16] as their scheme required three additions and the same amount of shifts. It might be the reason that their SSE version was slower than an enhanced serial algorithm.

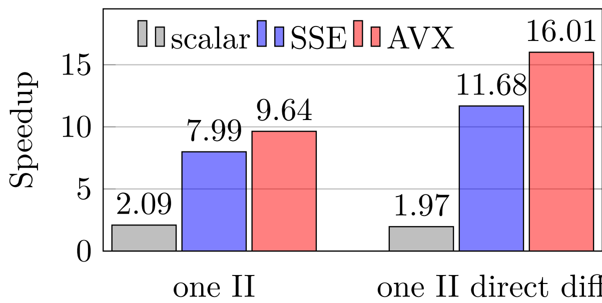
Unfortunately, as AVX does not have shift operations, they are considered to be only useful for integer data. So the shift has to be emulated by combining a shuffle and a blend. The shuffle rearranges data and the blend masks the first element with zero, which is not supported by the shuffle or other operations. AVX2 added integer support and shift operations at the same time. Due to the lane concept, a special cross-lane operation is necessary. The idea is to do the partial sum for each lane. In the last step, the overall sum of the lower lane is broadcasted to all elements in the higher lane of a register and added. What is helpful is that the partial prefix sum in the first step is independent from the other values. Without any doubt, $O - PL + P_x$ does not depend on the sum at first. In the final step, both temporal results have to be merged with a vector addition. In the first place, we have two independent data streams, which helps exploring instruction level parallelism. This is especially important due to the fact that — as stated before — summing within the vector is not ideal for vector units. The data preparation is another step optimal for vector units. If pattern matching is done using a norm without an inner product, the similarity measure is applied to the difference between pattern and test candidate.

We can estimate the expected speedup. For the regular version, we require 3 additions (or subtractions). The vectorized version has an overhead of $2 \cdot \log n$, where n is the number of vector elements. Then, there are three additions and one broadcast, but this already computes n pixels at once. Note, that this is a very rough estimation. We have not taken into account instruction level parallelism. This means instructions differ in latency and throughput. Moreover, the processor might have more operational units for some instructions than for others [21]. Another fact we did not consider is moving data around. SSE and AVX are — like the whole x86 instruction set — based on load and store. The normal version requires a load for each single element, however, the corresponding instructions for vector units load data chunks as large as the vector unit in a single step at the same time. Making the process faster, the bandwidth is also exceeded faster.

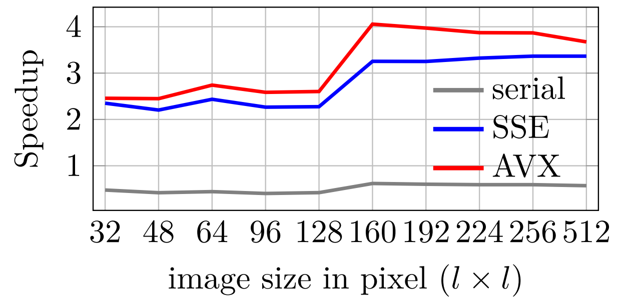
The complexity of the analysis above should make it clear that it is nearly impossible to give an estimated speedup for the whole discrepancy norm calculation. Thus, we will use practical tests to evaluate the performance impact.

input	v_3	v_2	v_1	v_0
shift	0	v_3	v_2	v_1
add	v_3	v_{3+2}	v_{2+1}	v_{1+0}
shift	0	0	v_3	v_{3+2}
add	v_3	v_{3+2}	v_{3+2+1}	$v_{3+2+1+0}$

Table 3. Computing partial prefix sum for a vector register holding 4 elements.



(a) Details about performance if algorithmic optimization and vectorizationis applied to discrepancy norm.



(b) Details about performance if vectorization is applied to integral image computation with the OpenCV algorithm serving as references.

Figure 3. Results of performance evaluation tests

4. Performance Analysis and Evaluation

Coding is done with C++, whereas *Visual Studio 2013* from Microsoft severs as the compiler. The only adjustment is the setting *Enable Enhanced Instruction Set* in the group of *Code Generation*. The selected target architecture is 64-bit. The test system is based on an Intel *i5-4460*. The computer runs *Windows 7 Professional Service Pack 1 64-bit*. The test algorithm applies the discrepancy norm in a sliding window approach, that the implementation is executed many times. Furthermore, the whole test setup is run several times to eliminate random influences. As we measure similarity compared to a pattern, reference subtraction has to be applied for each window. We include this step in time measurement as it is vital for this task and can not be omitted.

Figure 3a summarizes the speedup with the test setup. The direct difference approach outperforms the other implementations by far. With the AVX vectorization leading to a speedup of 16 and SSE vectorization to a speedup of 12. The algorithmic optimized serial version already doubled performance. The average execution time of the AVX version is 0.612 seconds matching a 64×64 data patch within a 512×512 image. AVX can process eight 32-bit integer values at the same time, which is exactly the speedup gained by the vectorization compared with the serial version. On the other hand, SSE produces super linear speedup exceeding theoretical maximum. The data indicates that reference subtraction has a big impact on the runtime. Embedding the difference building in the algorithm itself improved the performance about 50% for SSE and 65% for AVX.

Another comparison concentrates on the vectorization of the integral image algorithm alone. Many common algorithms like *SURF* are based on this intermediate representation [22]. The OpenCV implementation for integral image is compared to the vectorized implementation of the authors and a straight forward serial version. Figure 3b shows the results. The OpenCV algorithm serves as the reference and is twice as fast as a simple serial implementation. This suggests that OpenCV uses the approach from [13] that requires extra storage but reduces the necessary additions. Nonetheless, the vectorized version outperforms OpenCV at any image size, gaining a speedup of 2.5 to 4, depending on the image size.

For embedded applications it is interesting whether the vectorization scheme is applicable to other architectures, too. In embedded computing the ARM architecture plays a crucial role. Here, the ARM Cortex-A series offers SIMD capabilities with *NEON* technology offering a data width equal to SSE. [23]. All in all, the whole vectorization can be coded with the NEON instructions. Unfortunately,

both SIMD architectures have totally different instructions when it comes to data reordering. The most powerful data rearrange instructions of *NEON* are *VTBL* and *VTBX*. One the one hand, they are expensive in terms of execution cycles. On the other hand, several cases can be replaced by faster instructions. For example, the vector shift can be achieved using the NEON instruction *VEXT* and a register containing zero; broadcasting a single element can be done with *VDUP*. The newest ARM instruction set named *ARMv8* should allow further performance optimizations by adding cross-lane instructions providing functionality exactly needed by discrepancy norm calculation like horizontal summation and taking minimum or maximum [24]. Though, it is impossible to estimate the achievable speedup without practical tests.

5. Conclusion

We analyzed a variant of an image similarity measure based on Hermann Weyl's discrepancy from the point of view of efficient implementation by exploiting redundancies in computing multiple integral images. Finally, we proposed an implementation based on vectorization of prefix sums and summed-area tables which results in a speed-up factor 16 compared to a standard integral image based computation. Future research is left for checking parallelization and implementation optimizations also of other variants of Weyl's discrepancy.

References

[1] B. A. Moser, "A similarity measure for image and volumetric data based on Hermann Weyl's discrepancy.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, pp. 2321–9, nov 2011.

[2] B. Moser, G. Stübl, and J.-L. Bouchot, "On a non-monotonicity effect of similarity measures.," in *SIMBAD* (M. Pelillo and E. R. Hancock, eds.), vol. 7005 of *Lecture Notes in Computer Science*, pp. 46–60, Springer, 2011.

[3] H. Weyl, "Über die Gleichverteilung von Zahlen mod. Eins," *Mathematische Annalen*, vol. 77, pp. 313–352, Sept 1916.

[4] B. Chazelle, *The discrepancy method: randomness and complexity*. Cambridge University Press, 2000.

[5] B. A. Moser and T. Natschlager, "On stability of distance measures for event sequences induced by level-crossing sampling," *Signal Processing, IEEE Transactions on*, vol. 62, no. 8, pp. 1987–1999, 2014.

[6] B. A. Moser, "Stability of threshold-based sampling as metric problem," in *Event-based Control, Communication, and Signal Processing (EBCCSP), 2015 International Conference on*, pp. 1–8, IEEE, 2015.

[7] B. A. Moser, "The range of a simple random walk on $\mathbb{Z}$: An elementary combinatorial approach," *The Electronic Journal of Combinatorics*, vol. 21, no. 4, pp. P4–10, 2014.

[8] J.-L. Bouchot, G. Stübl, and B. Moser, "A template matching approach based on the discrepancy norm for defect detection on regularly textured surfaces," in *10th International Conference on Quality Control by Artificial Vision*, pp. 80000K–80000K, International Society for Optics and Photonics, 2011.

[9] G. Stübl, B. Moser, and J. Scharinger, "On approximate nearest neighbour field algorithms in template matching for surface quality inspection," in *Computer Aided Systems Theory-EUROCAST 2013*, pp. 79–86, Springer, 2013.

[10] F. C. Crow, "Summed-area tables for texture mapping," *ACM SIGGRAPH computer graphics*, vol. 18, no. 3, pp. 207–212, 1984.

[11] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra, "Fast summed-area table generation and its applications," in *Computer Graphics Forum*, vol. 24, pp. 547–555, Wiley Online Library, 2005.

[12] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.

[13] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. 511–518 vol.1, 12 2001.

[14] G. Stübl, "Robust defect detection for near-regular textures based on Hermann Weyl ' s discrepancy measure," 12 2013.

[15] M. J. Flynn and K. W. Rudd, "Parallel architectures," *ACM Comput. Surv.*, vol. 28, no. 1, pp. 67–70, 1996.

[16] N. Zhang, "Working towards efficient parallel computing of integral images on multi-core processors," in *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, vol. 2, pp. 30–34, April 2010.

[17] Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes: 1, 2A, 2B, 2C, 3A, 3B and 3C*, June 2015.

[18] G. E. Blelloch, "Prefix sums and their applications," tech. rep., Synthesis of Parallel Algorithms, 1990.

[19] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with cuda," *GPU gems*, vol. 3, no. 39, pp. 851–876, 2007.

[20] B. Bilgic, B. Horn, and I. Masaki, "Efficient integral image computation on the gpu," in *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pp. 528–533, June 2010.

[21] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. San Francisco, California: Morgan Kaufmann, 4 ed., 2007.

[22] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer vision–ECCV 2006*, pp. 404–417, Springer, 2006.

[23] ARM, *Cortex-A9 NEON Media Processing Engine - Technical Reference Manual*, June 2012.

[24] ARM, *ARM Cortex-A Series - Programmer's Guide for ARMv8-A*, March 2015.