Fast Approximate Nearest-Neighbor Field by Cascaded Spherical Hashing

Iban Torres-Xirau, Jordi Salvador and Eduardo Pérez-Pellitero

Technicolor R&I Hannover

Abstract. We present an efficient and fast algorithm for computing approximate nearest neighbor fields between two images. Our method builds on the concept of Coherency-Sensitive Hashing (CSH), but uses a recent hashing scheme, Spherical Hashing (SpH), which is known to be better adapted to the nearest-neighbor problem for natural images. Cascaded Spherical Hashing concatenates different configurations of SpH to build larger Hash Tables with less elements in each bin to achieve higher selectivity. Our method is able to amply outperform existing techniques like PatchMatch and CSH. The parallelizable scheme has been straightforwardly implemented in OpenCL, and the experimental results show that our algorithm is faster and more accurate than existing methods.

1 Introduction

Computing Approximate Nearest Neighbor Field (ANNF) between image patches has become a main building block for many computer vision and image processing algorithms, such as image re-targeting tools, image denoising or texture synthesis among others. Given two images A and B, the goal is to find for every patch in A a similar patch in B. The large size of images translates into millions of patches for each image A and B, and computing ANNF quickly becomes a challenge.

We propose the introduction of a recent hashing scheme, namely Spherical Hashing [1], in order to reduce the spatial and temporal complexity: we only require one set of hashing functions against the several sets of functions typically required when using planar function-based hashing (CSH [2]). In comparison to CSH and randomized search schemes like PatchMatch [3], we reduce the number of required iterations to just one to reach superior results.

To achieve this performance, we first introduce the general Spherical Hashing framework, and detail its applicability in ANNF estimation (Section 3). After observing its limitations, in Section 4 we propose a novel cascaded configuration for enhanced search without increasing testing time. Furthermore, we shed light on the implementation details of propagation strategies exploiting both local spatial coherence and similar appearance in both linear and hash space (using spherical Hamming distance).

We finally compare our proposed method to existing approaches in Section 5 and show the improvements achieved by our proposed contributions.

2 Related Work

Efros and Leung [4] proposed a simple non-parametric texture synthesis method based on sampling patches from a texture example and pasting them in the synthesized image. Various improvements for better structure preservation have been carried out by [5, 6] and outperformed by [7] obtaining globally consistent completions of large missing regions by formulating the problem as a global optimization.

Ashikhmin [8] introduced for the first time the concept of coherency and spatial propagation, which lead to speed up many previous non-parametric texture synthesis works by limiting the search space for a patch to the neighboring area of the match in the source texture. An extension of this work was carried out by Tong et al. [9], where the propagation algorithm is combined with a precomputed set of k nearest neighbors and used to quickly search for ANN.

The principle proposed in Locality-Sensitive Hashing [10] is that given a set of points in a metric space, with high probability the hashing function families will distribute points that are close to each other to the same bin. The scheme proposed by Datar et al. [11] uses a particular family of LSH functions to determine regions in the space: $h_{a,b}(v) = \frac{a \cdot v + b}{r}$ where r is a predefined integer, b is a value drawn uniformly at random from $[0, r]$, and a is a d-dimensional vector with entries chosen independently at random from a Gaussian distribution.

The idiosyncrasy of PatchMatch (PM) [3] relies on the dependency among queries. It observes the spatial coherence of images and propagates good matches to their neighbors (coherence-based propagation). That is, for a pair of similar patches in two images, their neighbors are also likely to be similar. PM has a first stage of initialization to seed the nearest neighbor field, and an iterative process to improve the search by propagating good matches. Since initialization can be done by assigning random values, and because PatchMatch does not organize the data beforehand, after seeding most candidates are unlikely to be good matches.

Coherence Sensitive Hashing (CSH) [2] at its turn relies on LSH and on PM, and it is a state-of-the-art method for ANN search that exploits the idea of randomly partitioning the data space proposed by LSH, generating a family of hash functions to index the patches and store them in bins in a hash table. But alternatively to LSH, CSH uses a different set of functions (Walsh Hadamard kernels) to achieve the dispersion to be as large as possible when projecting the patches into the kernels, and also because of its low computational cost. CSH outperforms LSH by generating a larger number of (nearest neighbor) candidates since, according to PM, it also exploits coherence in images and furthermore combines it with appearance-based candidates.

Another method for ANNF is Propagation-Assisted KD-Trees (PAKT) [12], which improves PM and CSH in terms of accuracy and performance, and therefore becomes a current state-of-the-art algorithm. It merges contributions from kd-trees [13] and PM, so it is also based on (deterministically) partitioning the space, but its key insight is to exploit both the distribution of the candidates of patches in the source image as the dependency of the query patches. Tree-based methods such as kd-trees organize the candidates adaptively to their distribution

in the search space, and a query can find its ANN by checking a small number of candidates. Indeed, PAKT introduces a propagation step after organizing the candidates in a traditional kd-tree, in which the tree nodes checked by each query are propagated from the nearby queries to its own leaf and to a nearby extra leaf. In contrast with general search schemes based on kd-trees, PAKT has no backtracking when descending the tree to a leaf, but it only checks a small number of candidates hence it becomes a fast method for computing ANNF. The principle of PAKT is similar to the one assumed at hashing: one bin has such a high likelihood to contain the NN patch that there is no need to check in other bins, but we must note that spherical hashing provides better partitioning than the tree scheme based on hyperplanes.

The Spherical Hashing (SpH) algorithm introduced by Heo et al. [1], uses a family of spherical hashing functions because it considers that the partitioning of the data space by means of hyperplanes can be improved by using hyperspheres: a higher number of closed regions can be constructed by using multiple hyperspheres, while the distances between points located in each region are bounded. The SpH algorithm has an initialization step to conveniently choose the spherical hashing functions in order to balance the amount of data falling in each bin. When trained with a large data set, the content in bins is likely to be uniformly distributed in the hash table. Although SpH does not focus on ANNF (it does so on data mining), its idea is still extrapolable for our purpose, and the improvement at partitioning the data space is a key insight for our method.

Our method builds on previous approaches like CSH and SpH, therefore it is able to quickly find similar patches with more accuracy and reducing the computational time over CSH, since the spherical hashing algorithm guarantees an increment of closed regions with less functions. The propagation step of our method is similar to the one adopted in CSH, which, based on PM, exploits query dependence. Our method additionally makes use of the Spherical Hamming Distance introduced by [1] and proposes a new class of candidates (spherical-neighbor propagation) to enlarge the list.

3 Spherical hashing for ANNF

In this section we present the general idea behind our coherence-sensitive ANNF search method based on a spherical hashing scheme. We first discuss the training stage for an optimal choice of the hashing functions for image patches.

3.1 Training

At the training stage we use a large set of data patches from a wide range of images to create a family of data-dependent hashing functions that guarantees a good partitioning of the space, aiming to similar amount of data falling in each region (balance). Given a sufficiently large training set, the functions selection will remain valid for any new given data we may process.

We treat each patch as a D-dimensional vector in Euclidean space. These m vectors $X = \{x_1, x_2 ... x_m\}, x_i \in \mathbb{R}^D$ form an input manifold M of dimension D. Compared to previous hashing approaches, which used hyperplanes as splitting functions, spherical hashing considers the inclusion in hyperspheres. The algorithm models those hyperspheres by a pivot $p_k \in \mathbb{R}^D$ and a distance threshold (i.e. radius) $t_k \in \mathbb{R}^+$. Each data point x_i is then encoded with a binary number $b_k = \{-1, +1\}^c$ being c the number of hyperspheres and the length of the code, where the k-th bit is computed as follows:

$$b_k(x) = \begin{cases} -1, & d(p_k, x) > t_k \\ 1, & d(p_k, x) \leqslant tk \end{cases}, \tag{1}$$

where $d(\cdot, \cdot)$ denotes the Euclidean distance between two points in $\mathbb{R}^D$. The main benefit of using hyperspheres instead of hyperplanes is that defining closed regions in the spherical case is much more accurate, since these regions are bounded, and elements belonging to a region must be strictly closer; moreover, the number of spheres needed to define a closed region is minor than the number of hyperplanes, therefore the number of functions is also lower and the computational time decreases significantly.

Following the scheme proposed by SpH [1], the algorithm aims to minimize the search time in the bins of the hash table and improve the accuracy of the search, which can be achieved by reaching independence between hashing functions and a balanced partitioning of the data space. The two conditions to satisfy are the following:

$$Pr[h_x(x) = 1] = \frac{1}{2},\ x \in X,\ 1 \leq k \leq c \tag{2}$$

$$\begin{gathered} Pr[h_i(x) = 1, h_j(x) = 1] = Pr[h_i(x) = 1] \cdot Pr[h_j(x) = 1] = \tfrac{1}{2} \cdot \tfrac{1}{2} = \tfrac{1}{4}, \\ x \in X,\ 1 \leq i, j \leq c \end{gathered} \tag{3}$$

To fulfill these conditions, an iterative step for the selection of the functions has to be carried out, so that the centers of the hyperspheres p_k and their radii t_k (thresholds) are refined at every iteration. In order to accomplish Eq. 2 and Eq. 3, the algorithm uses two variables to help these computations:

$$o_i = \mid \{s_k \mid h_i(s_k) = 1,\ 1 \leq k \leq m\} \mid$$

$$o_{i,j} = \mid \{s_k \mid h_i(s_k) = 1,\ s_k \mid h_j(s_k) = 1,\ 1 \leq k \leq m\} \mid$$

o_i denotes the number of data points which have 1 bit set for the i-th hashing function (number of data points falling inside the i-th hypersphere) and is used to satisfy balanced partitioning for each bit following Equation (2), while $o_{i,j}$ counts the number of patches that are contained within both of two (i-th and j-th) spheres and is used to guarantee the independence between the i-th and j-th hashing functions following Equation (3).

The iterative process for pivots refinement first adjusts the pivot centers of two hyperspheres in a way that $o_{i,j}$ becomes close to $\frac{m}{4}$, and then a threshold t_i is chosen such that o_i becomes $\frac{m}{2}$ to meet balanced partitioning. At each iteration the centers of the pivots are moved to new locations according to the forces computed regarding to each $o_{i,j}$. For each pair of spheres i and j, a repulsive or attractive force from p_j to p_i , $f_{i \leftarrow j}$ is computed as the following:

$$f_{i \leftarrow j} = \frac{1}{2} \frac{o_{i,j} - m/4}{m/4} (p_i - p_j) \ ,$$

and the accumulated force f_i is the average of all the computed forces from all the other pivots:

$$f_i = \frac{1}{c-1} \sum_{j \neq i} f_{i \leftarrow j} \ .$$

Convergence of the system is achieved when the ideal values for mean and the standard deviation of $o_{i,j}$ are $\frac{m}{4}$ and zero respectively (within $100\epsilon_m\%$ and $100\epsilon_s\%$ error tolerances).

3.2 Indexing and building the hash table

A hash code of length c is computed for each patch of a new source image using Eq. (1) and is stored in a bin with all its similar patches, as shown in Figure 1. At the indexing stage, a total of c patch-to-pivot squared L_2 distances are computed, which translates into a computational cost of $O(Dc)$ operations per patch. We build a hash table (HT) of 2^c entries of different sizes to store the entire image.

The building process is divided into two runs: we first compute all patch indices and determine the size of each bin in the hash table, and then we create the table according to the dimensions of each bin and we orderly store each patch in its position in the HT. However, we reduce the equivalent building time and search time compared to similar structures as PAKT [12], since the kd-tree scheme requires the decomposition in a suitable basis, while the hashing algorithm allows more flexible partitions in a native way. Furthermore, our algorithm is highly and easily parallelizable so the final build time is reduced to practically $O(Dc)$ in e.g. an OpenCL implementation.

3.3 Direct search

For every patch in a given query image we compute its index, so that a list of similar candidates with the property of space similarity due to belonging to the same region can be found in the hashed bin. Since bins often contain several patches, various techniques can be carried out to select the ANNF, e.g. a re-ranking algorithm based on similarity between images (L_2), which increases the computational time in exchange for higher accuracy, or a random sampling within the HT entry that provides a fast approximate match.

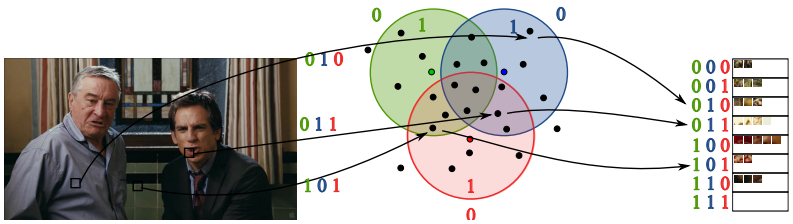

Fig. 1. Spherical hashing for ANNF places data points in a $\mathbb{R}^D$ space ($D = 2$ in the figure) and a set of c spheres (to be computed at the training phase) determine which region each point belongs to. The hash table on the right side stores in a bin every point falling in its correspondent region. The search stage hashes query points through the same system of spheres as the indexing phase. Patches of the query image point to bins of the hash table where the source data is stored creating a list of match candidates.

Even though the system is designed to achieve balance in the HT, when increasing the number of spheres up to a certain limit, this property becomes unstable, so a closed region in the $\mathbb{R}^D$ space is likely to contain no data points. This is the main reason why we cannot build a highly discriminative set of spheres and the motivation of our proposal in Section 4.

3.4 Drawbacks

For large values of c, a small number of patches in each bin is expected, and hence the re-ranking algorithm in the search is less expensive. However, this assumption is not always valid. Given the dimensionality of the $\mathbb{R}^D$ space determined for the size of the patches, a certain threshold value for c exists, at which convergence to guarantee the properties of Eq. (2) and Eq. (3) is not achievable when we create the hash functions. Beyond this level, several bins result to be empty after building the HT and some others are overpopulated. We propose an extension of the given scheme to scalably increase the number of filled bins making them less populated and increase accuracy without introducing any cost during testing.

4 Cascaded Spherical Hashing for ANNF

We propose a novel approach to exploit the space partitioning with special influence in the densest areas. Despite the fact that spherical hashing guarantees a good equability in the HT, given the dimensionality D of the $\mathbb{R}^D$ space it turns into a nearly impossible task to unlimitedly increase the number of hyperspheres and still ensure balancing. Concatenating multiple dependent systems of hyperspheres and building a novel multi-dimensional hash table based on this cascading concept results to be a more accurate method, since, although it does not guarantee perfect balance in the final HT, we reduce the overfilling and simultaneously enlarge the number of filled bins, which translates into better

performance in terms of accuracy at no additional processing cost during the testing time.

Points falling in a certain region in one system are then distributed according to the region they belong to in the next dimension and so on, hence our method provides more similarity between patches belonging to the intersection of closed regions between cascaded systems of spheres.

A total of 2^c bins are created for a chosen number of spheres c in the original SpH. For a large set of m random data distributed in the space, each bin is likely to store $\frac{m}{2^c}$ patches, but when trained with real data, we observe that the distribution of the source patches in the space is not uniform and therefore some bins happen to end up empty while others are overpopulated, especially for large c. Our approach concatenates various hashing systems of different numbers of spheres $c_1, c_2, ..., c_N$, and builds a final hash table of $2^{c_1} \cdot 2^{c_2} \cdot ... \cdot 2^{c_N}$ bins, achieving high discrimination at indexing.

4.1 Cascade training

The offline training is an iterative algorithm where each iteration consists of two phases carried out to concatenate N dependent hash systems, in order to make every system more discriminative to the regions presenting more data (Figure 2).

4.1.1 Spheres training In this phase a data set is trained following the scheme detailed in Section 3.1 to obtain a system of c_i hyperspheres.

4.1.2 Data set refinement The HT deduced from the previous system(s) is filled by the whole data set, and observing the density of data falling in each entry we determine whether a region is overpopulated. We generate a subset of all vectors/patches lying in overpopulated bins and train a specialized sphere system as in Section 4.1.1.

4.2 Indexing and building the Hash Table

Actually, the way we create the resulting hash table can be seen as a multi-dimensional Spherical Hashing, where every dimension (or spheres set) contributes to higher accuracy. For a given source patch, its index is created by a combination of the N hash codes generated using the function of Eq. 1 for each system. The source data is stored in the resulting HT which owns a large number of bins and, since the different spherical hashing systems are built in cascade, no bin contains a large amount of data and it yields a higher number of filled ones.

Building the cascaded-HT requires the same computational cost than the building time of the method proposed in Section 3 when we compare both approaches with the same total amount of hyperspheres. Besides, this interpretation comes handy to introduce the spherical propagation introduced in Section 4.4.2.

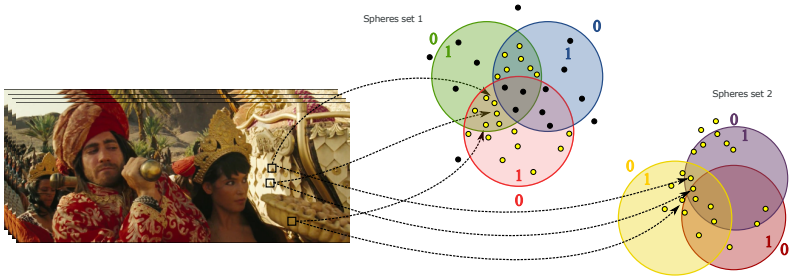

Fig. 2. Cascaded training: the data set is trained to obtain the first set of spheres (top) using the hashing scheme of Section 3.1 until convergence is achieved. When the Hash Table is filled by the data set, our training stage observes the density of data in space regions, and uses the subset of data contained in the most populated regions (highlighted data points) to train the next system of spheres (bottom). This is similar to the procedure in other cascaded methods, such as AdaBoost [14].

4.3 Direct search

For a given query image every patch is hashed to an entry of the cascaded-HT in order to find a list of patches of the source image simultaneously belonging to the same regions in the N sets of spheres. It is interesting to note that, if $c_1 \leq c_2 \leq ... \leq c_N$, the sets of hashing systems ensure more stability for the firsts systems, so that if a query patch is hashed to an intersection of regions where no source data is stored, the algorithm can fall back to the other (lower) dimensions to find plausible matches (Figure 3). All patches belonging to the hashed bin are likely to be a good match and, again for the ANNF search, a re-ranking or an in-bin random sampling are the main alternatives to adopt. However, since we manage to have more non-empty bins and containing less data at the same time, the re-ranking technique results to be faster and the random search is more accurate compared to the original method. In our experiments we avoid re-ranking thanks to the accurate partition obtained with the cascaded setup (we just perform random sampling within the selected bin).

4.4 Propagation

We improve the search by exploiting both local spatial coherence and hashing appearance in both linear and hash space.

4.4.1 Spatial Propagation To enlarge the candidates list provided by the Hashing scheme and following the idea proposed by PM, we also adopt the concept of image coherence to propagate good matches to their spatial neighbors. We do so in a similar way to the extended mechanism introduced by PAKT [12], a bin-propagation to find better matches. Suppose a query patch $p_A(x-1, y)$ has found a similar patch $p_B(x'-1, y')$, we improve the result of $p_A(x, y)$ trying patches belonging to the same bin as $p_B(x', y')$, and so on with the other

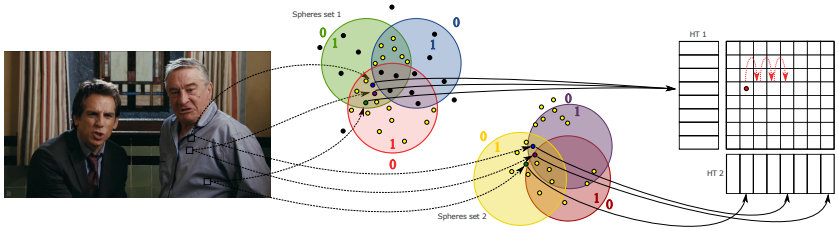

Fig. 3. Cascaded Hash Table search: Patches are hashed to bins in the corresponding HT for each set of spheres ($N = 2$), and by combining them we obtain a cascaded-HT capable of distributing data with high discrimination based on the closed region defined by the intersection of the systems where data falls. When the cascaded-HT of the figure is filled with real data from an image, not all bins happen to store data, but significantly more than in a single HT (and also less populated). Indeed, we still observe empty bins due to physical impossibility of belonging to regions which never intersect, although with the procedure we follow to create the HT it does not represent a cost in memory nor time. When a query patch is hashed to a bin that contains no source data (red point), the naive process of our method relies in the lower dimensions (or hyperspheres systems) by a linear search until a filled bin is found to find matches.

directions (Figure 4). By proceeding this way, we obtain $x4$ times candidates for every query patch.

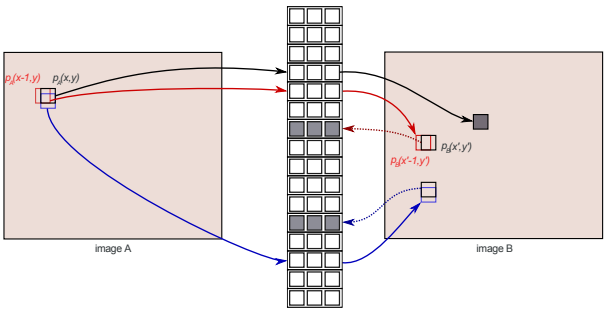

Fig. 4. Bin-Propagation in Spherical Hashing Scheme: If the spatial neighbor of $p_A(x, y)$ (black patch), $p_A(x-1, y)$ (red), has a good match $p_B(x'-1, y')$ (red), the spatial neighbor of the match, $p_B(x', y')$ (black) belongs to a bin in the Hash Table where all the patches are plausible candidates (shaded).

We analyze the benefits of the propagation step, adopted from the spatial coherence concept proposed by PM, to find out the impact of the accuracy gain over the cost in time. In the same scenario we reconstruct 2500 independent images using the single and the cascaded algorithm both with and without propagation, and the results are shown in Figure 5. Note that propagation has

notable influence but also implies a computational cost penalty avoided by the proposed cascaded approach.

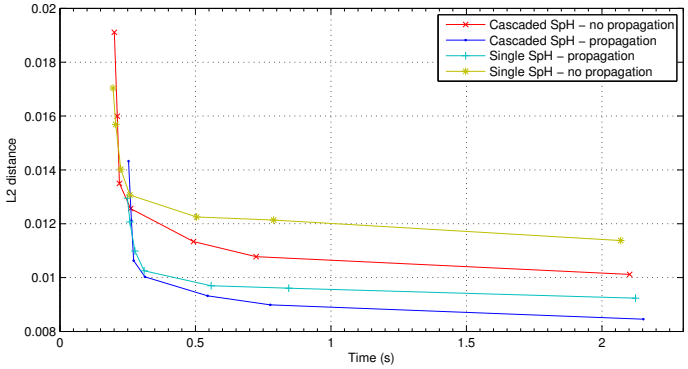

Fig. 5. Error/Time tradeoffs of Cascaded and Single Spherical Hashing, both with and without propagation.

4.4.2 Spherical Propagation We make use of the concept of Spherical Hamming Distance (SHD) introduced by [1], which computes the distance between codes of two bounded regions (in one dimension of our multi-SpH) as follows:

$$d_{shd}(b_i, b_j) = \frac{\mid b_i \oplus b_j \mid}{\mid b_i \wedge b_j \mid}, \tag{4}$$

where b_i, b_j are the hash codes of patches i, j, and $\mid b_i \oplus b_j \mid$ measures the number of different bits (belonging to different spheres) and the term $\mid b_i \wedge b_j \mid$ denotes the number of common $+1$ bit (belonging to same spheres). This distance is useful when a given query patch hashes to a bin where no source data is stored. When searching in a bin that happens to be empty, a straightforward implementation could initially rely on looking into the following bins until one contains data (Figure 3). Since this situation does not happen often ($\sim 0,005\%$ of patches in our experiments), empirical results show that this is often a good assumption. Nevertheless, an improvement in accuracy/speed-up versus the naive approach is achieved by using the Spherical Hamming distance to look in spherical-neighboring cells at minimum distance, as Figure 6 shows.

5 Results

All algorithms were run on a PC with an Intel Xeon CPU 2.67GHz CPU and 12GB RAM. Note that all the methods use the same CPU platform, even for the parallelized OpenCL code. We test the search algorithms on a subset of the

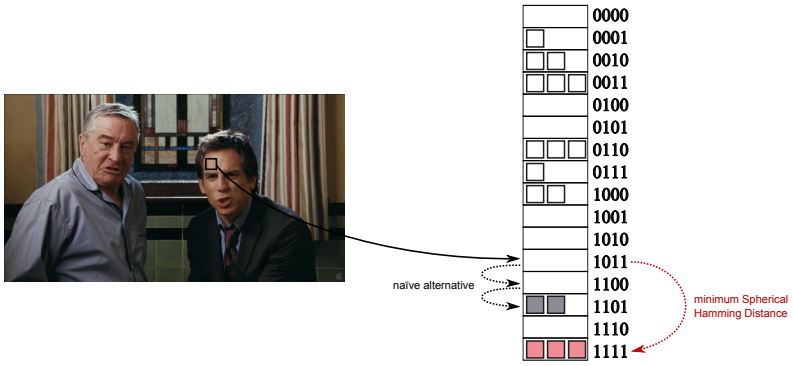

Fig. 6. Spherical Hamming Distance in the Hash Table: a patch hashed to an empty bin is likely to have better matches using the SHD to compute the spherical-neighboring cell instead of looking into the consecutive bins until a filled one is found.

public data set VidPairs [15] and the training of our method is carried out using the publicly available Kodak data set.

5.1 Validation

In order to validate the proposed approach, we test different hypersphere configurations. Figure 7 shows that the best performance is achieved when using our proposed cascaded approach. We also train and test independent systems of hyperspheres and combin them, obtaining worse performance results than with single SpH. This proves that there is no benefit in training independent system without focusing on the overpopulated bins.

Finally, we validate that our Cascaded SpH method is more accurate than the single SpH at no additional testing computational cost, obtaining results up to 0.7 dB higher in PSNR.

Note how the progressive improvements by adding more cascading levels outperform the other alternavies, e.g. our method with 21 bits and 3 cascades outperforms both alternatives with 25 bits. The small margins reflect the already excellent performance of spherical hashing in the basic configuration with a relatively high number of bits, but the interesting aspect is that the gain of the cascaded configuration comes at literally no computational cost.

5.2 Comparison to state-of-the-art

We compare our algorithm with PM [3] and CSH [2]. Our algorithm is implemented in OpenCL and exploits its parallel-friendly nature, yielding important speed-ups. We obtained the CSH code from the authors website [15], which is mainly implemented in C++. The PM algorithm is an OpenCL self-implemented code, which at its turn, outperforms the original PM in processing time.

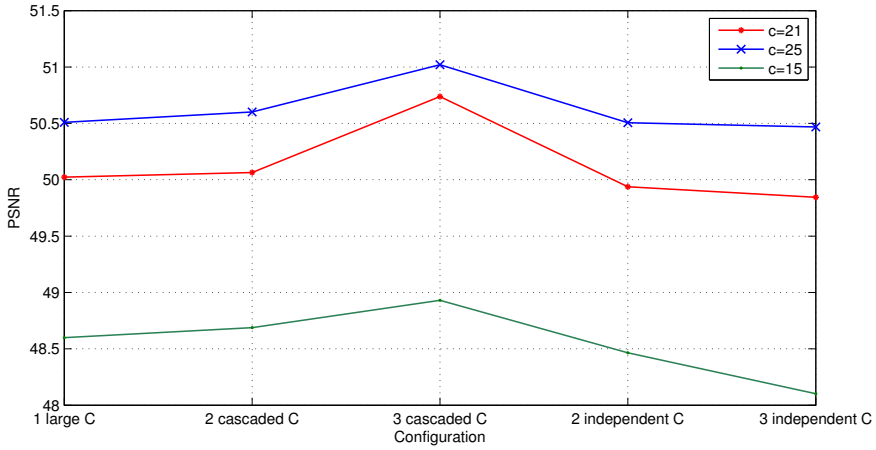

Fig. 7. Time-PSNR tradeoffs averaged on 225 reconstructions of independent images. The image size is 2Mp and the patch size is 3-by-3. *1 large C* refers to a single spherical hashing system with large C, *2/3 independent C* refers to configurations of independent hyperspheres systems with multiple tables, and *2/3 cascaded C* refers to the proposed method. In the experiments, configurations for 2 and 3 stages with equal number of spheres as for *1 large C* are: $c_{15} = \{6, 9\}, \{4, 5, 6\}$, $c_{21} = \{8, 13\}, \{6, 7, 8\}$ and $c_{25} = \{11, 14\}$, $\{7, 8, 10\}$.

Even though an open implementation of PAKT is not available, by the comparison against CSH published in the original paper [12], we can extrapolate that our algorithm is qualitatively competitive in terms of time and accuracy.

Figure 8 shows time-accuracy results for 4-by-4 patches and 2Mp images. Accuracy is measured as the squared L_2 distance between the query patch and its nearest neighbor match found by PM, CSH, our single SpH and our cascaded SpH (in the experiments $N = 3$). We also present results obtained by an exhaustive search to compare the measures to the maximum possible accuracy.

As Figure 9 shows, our results at reconstruction are visually better than PM/CSH. When compared to the fastest method we achive speed-ups of $\times 1.3$ with a quality difference of 9 dB in PSNR. When compared to the most accurate method, we still improve by 7 dB considering also that our method is $\times 3.5$ faster.

It is interesting to note that reconstructed images obtained using PM are not able to accurately reconstruct some flat areas (especially noticeable in the first column of images in Figure 9). This effect appears due to the search algorithm in PM, which considers the entire patch, while CSH and our method carry out the search by subtracting the mean value.

6 Conclusions

We propose a new algorithm for computing ANNF. We build on the Spherical Hashing algorithm of Heo et al. [1], originally presented in the context of data

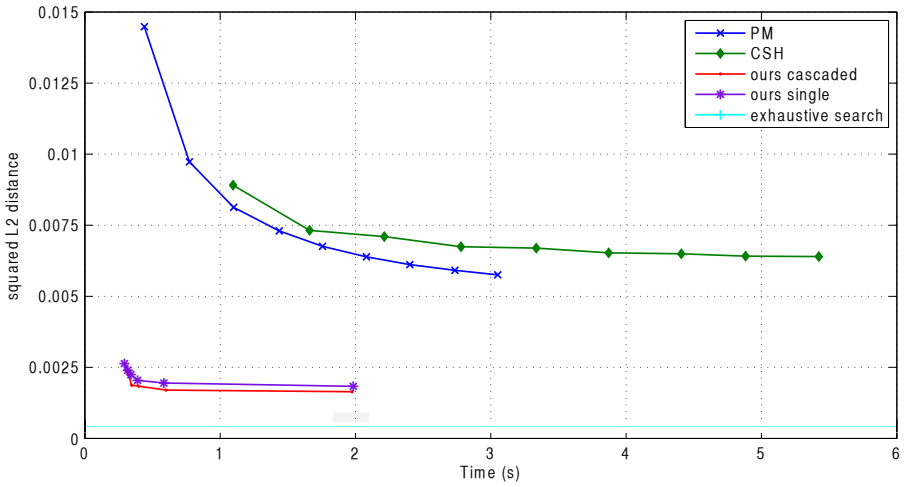

Fig. 8. Time vs accuracy averaged over 225 reconstructions of independent images. The image size is 2Mp and the patch size is 4-by-4. Each marker on PM's and CSH's curves represents the performance for each iteration. Each marker on our method represents the performance for increasing total number of hyperspheres c_i {12, 15, 18, 21, 24, 27}. Exhaustive search is included to determine the lower bound error, but we avoid to show the long computation time to keep a proper scaling for the more efficient approaches.

mining, which we adapt to the specific problem of ANNF estimation. We improve the baseline method by adding a propagation mechanism based on both local and visual coherence, inspired by the scheme proposed by PAKT [12], which in turn already extends the original PM local coherence concept [3].

We observed practical limitations in spherical hashing for ANNF when trained with large numbers of hyperspheres, and we overcome them by introducing a cascaded training approach. This cascaded scheme aims to improve the partitioning of the overpopulated regions without introducing any additional cost in testing time. In order to do so, we add complexity to the offline training stage to guarantee a better balance within the hash table.

We also introduce the usage of the spherical Hamming distance as an alternative hash selection for the rare situations in which a patch is hashed to an empty bin. Our algorithm, which allows straight-forward parallelization, has been compared to well-known state-of-the-art methods, obtaining the best-scoring results both in speed and quality.

We encourage to investigate further the benefits of our ANNF algorithm in other computer vision applications due to the applicability and feasibility of the method shown in the experimental results.

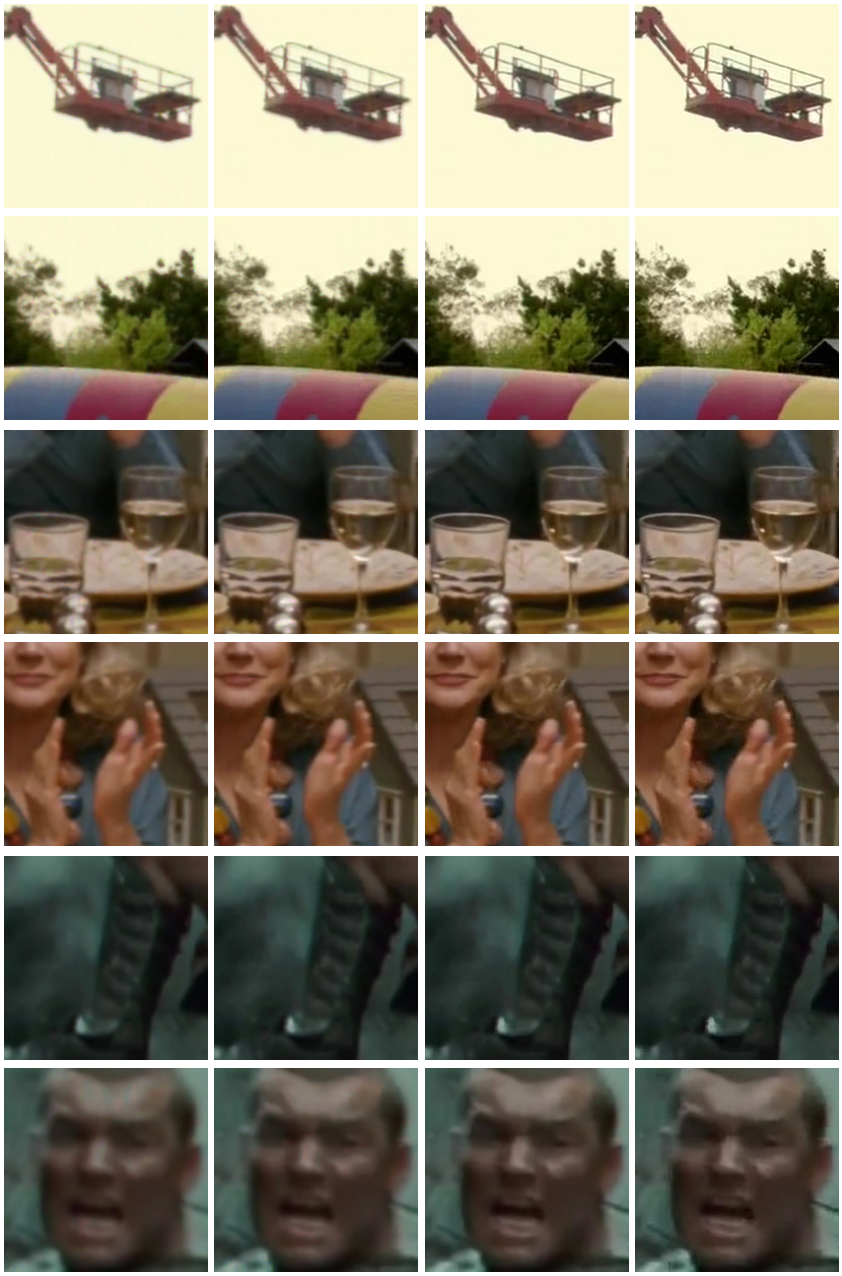

Fig. 9. Visual comparisons of the reconstructed images. Images are 2Mp and the patches are 4-by-4. All methods are run in 1 iteration, and $N = 3$ in our method for a total number of hyperspheres equal to 18 ($c_1 = 5$, $c_2 = 6$, $c_3 = 7$). The running times are: 0.47s, 1.22s, and 0.35s, with PSNR values are: 39.08dB, 41.44dB and 48.23dB for PM, CSH and ours, respectively.

Acknowledgment

We thank the anonymous reviewers for their constructive feedback, which resulted in an improved manuscript.

References

1. Heo, J.P., Lee, Y., He, J., Chang, S.F., Yoon, S.e.: Spherical hashing. In: IEEE International Conference on Computer Vision and Pattern Recognition (CVPR). (2012)
2. Korman, S., Avidan, S.: Coherency sensitive hashing. In: Proceedings of the 2011 International Conference on Computer Vision. ICCV '11, Washington, DC, USA, IEEE Computer Society (2011) 1607–1614
3. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B.: PatchMatch: A randomized correspondence algorithm for structural image editing. ACM Transactions on Graphics (Proc. SIGGRAPH) **28** (2009)
4. Efros, A.A., Leung, T.K.: Texture synthesis by non-parametric sampling. In: Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2. ICCV '99, Washington, DC, USA, IEEE Computer Society (1999) 1033–
5. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '01, New York, NY, USA, ACM (2001) 341–346
6. Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: Image and video synthesis using graph cuts. In: ACM SIGGRAPH 2003 Papers. SIGGRAPH '03, New York, NY, USA, ACM (2003) 277–286
7. Wexler, Y., Shechtman, E., Irani, M.: Space-time completion of video. IEEE Trans. Pattern Anal. Mach. Intell. **29** (2007) 463–476
8. Ashikhmin, M.: Synthesizing natural textures. In: Proceedings of the 2001 Symposium on Interactive 3D Graphics. I3D '01, New York, NY, USA, ACM (2001) 217–226
9. Tong, X., Zhang, J., Liu, L., Wang, X., Guo, B., Shum, H.Y.: Synthesis of bidirectional texture functions on arbitrary surfaces. ACM Trans. Graph. **21** (2002) 665–672
10. Slaney, M., Casey, M.: Locality-sensitive hashing for finding nearest neighbors [lecture notes]. Signal Processing Magazine, IEEE **25** (2008) 128–131
11. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the Twentieth Annual Symposium on Computational Geometry. SCG '04, New York, NY, USA, ACM (2004) 253–262
12. Sun, J.: Computing nearest-neighbor fields via propagation-assisted kd-trees. In: Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). CVPR '12, Washington, DC, USA, IEEE Computer Society (2012) 111–118
13. Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Trans. Math. Softw. **3** (1977) 209–226
14. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. **55** (1997) 119–139
15. Korman, S., Avidan, S.: Csh website.@ONLINE (2011)