Coherency Sensitive Hashing

Simon Korman and Shai Avidan
Dept. of Electrical Engineering
Tel Aviv University
simonkor@mail.tau.ac.il avidan@eng.tau.ac.il

Abstract

Coherency Sensitive Hashing (CSH) extends Locality Sensitivity Hashing (LSH) and PatchMatch to quickly find matching patches between two images. LSH relies on hashing, which maps similar patches to the same bin, in order to find matching patches. PatchMatch, on the other hand, relies on the observation that images are coherent, to propagate good matches to their neighbors, in the image plane. It uses random patch assignment to seed the initial matching. CSH relies on hashing to seed the initial patch matching and on image coherence to propagate good matches. In addition, hashing lets it propagate information between patches with similar appearance (i.e., map to the same bin). This way, information is propagated much faster because it can use similarity in appearance space or neighborhood in the image plane. As a result, CSH is at least three to four times faster than PatchMatch and more accurate, especially in textured regions, where reconstruction artifacts are most noticeable to the human eye. We verified CSH on a new, large scale, data set of 133 *image pairs.*

1. Introduction

Computing Approximate Nearest Neighbor Fields (ANNF) is an important building block in many computer vision and graphics applications such as texture synthesis [10], image editing [18] and image denoising [7]. This is a challenging task because the number of patches in an image is in the millions and one needs to find Approximate Nearest Neighbors (ANN) for each patch in real or near real time.

In the past, it was customary to compute ANNF with traditional approximate nearest neighbor tools such as Locality Sensitive Hashing (LSH) [13] or KD-trees [1, 16]. These tools perform well in terms of accuracy but are not as fast as one would hope. Recently, a novel method, termed PatchMatch [4], proved to outperform those methods by up to two orders of magnitude, making applications that rely on ANNF run at interactive rate. The key to this speedup is that PatchMatch relies on the fact that images are generally coherent. That is, if we find a pair of similar patches, in two images, then their neighbors in the image plane are also likely to be similar. PatchMatch uses a random search to seed the patch matches and iterates for a small number of times to propagate good matches. Unfortunately, PatchMatch is not as accurate as LSH or KD-trees and increasing its accuracy requires more iterations that cost much more time. In addition, the main assumption it relies on (i.e. coherency of the image) becomes invalid in some cases (e.g. in strongly textured regions), with noticeable influence on mapping quality. It is therefore beneficial to develop an algorithm that is as fast, or faster, than PatchMatch, and more accurate.

Coherency Sensitive Hashing (CSH) replaces the random search step of PatchMatch with a hashing scheme, similar to the one used in LSH. As a result, the process of seeding good matches is much more targeted and information is propagated much more efficiently. Specifically, information is propagated to nearby patches in the image plane, as is done in PatchMatch, and to similar patches that were hashed to the same value. In other words, we propagate information to patches that are close in the image plane or are similar in appearance. The end result is that our algorithm runs faster and gives more accurate results, in terms of RMS of the retrieved patches, compared to PatchMatch. This increased speed and accuracy comes at a modest increase in memory footprint since we need to store the hashing tables.

An interesting property of our algorithm is that its reconstruction errors are significantly lower than those obtained by PatchMatch. To measure this, we define *incoherency* to measure the number of neighboring patches in one image that are mapped to neighboring patches in the other image. We find that mapping produced by CSH is much less coherent than the one produced by PatchMatch. This is because CSH does not rely on the image coherency assumption as much as PatchMatch does. Experiments suggest a strong correlation between the coherency of the mapping and RMS error. The less coherent the mapping, the lower the error. We also characterized the errors by image content and found that CSH works better than PatchMatch in

textured regions. We demonstrate the advantages of CSH over PatchMatch on a new data set of 133 image pairs with 2 mega pixel resolution[1].

2. Related Work

Patch-based methods have been very successful in a wide variety of computer vision and graphics applications. Efros and Leung [10] introduced a simple non-parametric texture synthesis algorithm. It was quickly followed on and improved by [9, 15, 21]. Non-parametric texture synthesis was then used for various image editing applications by Simakov *et al.* [18] and it also inspired the method of non-local means for image denoising [7].

Common to all these techniques is the need to find, for each patch in image A, a similar (i.e., ANN) patch in image B, where in some cases images A and B can be the same image. Wei and Levoy [20] proposed a Tree Structure Vector Quantization (TSVQ) method to quickly find the necessary matches. Others relied on existing ANN search techniques such as kd-trees [1], perhaps enhancing them with PCA, to reduce dimensionality.

Ashikhmin [2] was the first to introduce the concept of coherency and used it to accelerate non-parametric texture synthesis. This was later extended to k-coherence by Tong *et al.* [19] that pre-computed a set of k nearest neighbors for each patch and used it to accelerate the search for ANN. They have also demonstrated it for texture synthesis.

Two leading methods for ANN search are kd-trees [1] and Locality Sensitive Hashing (LSH) [13]. Both partition the space, either deterministically (KD-tree) or randomly (LSH) in order to allow for quick query time. In this work we focus on LSH and show how to extend it to deal with coherent data, such as patches in an image.

The work most closely related to ours, and indeed the one that inspired ours, is that of PatchMatch [4]. PatchMatch takes image coherence to the extreme and uses it for various image editing applications. It was recently generalized and applied to other applications as well, such as image denoising [5] and an attempt to add appearance-guided information to its search was reported in [3].

PatchMatch works in rounds. Given a pair of images it randomly assigns each patch in image A to a patch in image B. Most assignments yield poor matches, but some are quite good. PatchMatch then propagates the good matches to nearby patches, in the image plane. To avoid being trapped in a local minima, it also performs a number of random patch assignments for each patch, keeping the best match after each stage. The algorithm usually converges after a small number of iterations.

[1]Code and data-set is available at www.eng.tau.ac.il/~simonk/CSH/index.html

3. LSH for Nearest Neighbor Search

The notion of *locality sensitive hashing* (LSH) was first introduced by Indyk and Motwani [13]. Given a set of points in a metric space, LSH function families have the property that points that are close to each other have a higher probability of colliding (under random members of the family) compared to points that are far apart. The first usage of LSH for nearest neighbor search in high dimensions worked in high dimensional binary Hamming space [11]. Our algorithm will follow the general lines of an LSH-based approximate nearest neighbor search scheme later proposed by Datar *et al.* [8]. In the rest of this section we outline their algorithm.

At the base of the algorithm is a family H of LSH functions and the ANN search algorithm consists of two stages: *indexing* and *search* (query). In the indexing stage, primitive hash functions from H are used to create an $index$ in which similar points map into the same hash bins with high probability. M such primitive hash functions are concatenated to create a *code* which amplifies the gap between the collision probability of far away points and the collision probability of nearby points. Such a code creates a single hash table, by evaluating it on all data-set points. In the search stage, a query point is hashed into a table bin, from which the nearest of residing data-set points is chosen. In order to decrease the probability of falling into an empty bin (with no data-set points), multiple (L) random codes are used to create L hash tables, which are searched sequentially at search stage. Datar et al.[8] show that the above scheme results in significantly improved efficiency compared to previous methods in the case of L_2 distances, which are the ones of interest in our case.

4. Coherency Sensitive Hashing (CSH)

In this section we layout our algorithm for approximate dense nearest patch search. The straight forward way to use the LSH search scheme for image patches is by treating each d-by-d patch as a d^2 vector in Euclidian space and the rest follows. However, it wouldn't take advantage of the wide extent of overlaps between nearby patches.

Instead, we follow the general lines of the LSH scheme, but replace several of its main ingredients with new ones, which are designed to exploit the image patches setup. At the *Indexing* stage, we replace the family of LSH functions with a new set of functions, which make use of the Walsh-Hadamard kernels (details in section 4.1). At the *search* stage, we dramatically extend the set of candidate patches that are considered, compared to the limited set of patches that point to the same index (details in section 4.2). We term the resulting scheme *Coherency Sensitive Hashing* (CSH). The CSH Algorithm is given in algorithm 1, while the details are given in the next subsections.

4.1. Indexing

The LSH scheme of Datar et al. [8], uses the particular family of LSH functions of the form $h_{a,b}(v) = \frac{a \cdot v + b}{r}$, 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 action of such a random function of this distribution (family) on a vector v (or patch) could be described by the 3 following stages: (1) Take a random line, defined by the vector a, divide it into bins of constant width r and shift this division by a random offset of $b \in [0, r)$ (2) Project the vector v on to the line (3) Assign it a hash value, being the index of the bin it falls into. The role of the random offset b is to neutralize the quantization limits of fixed binning. Specifically, it ensures that similar patches (which project to nearby locations on the line) will collide (fall into the same bin) with high probability.

In our case, the vector is a patch and we don't project it onto a random line, but rather on one of the first (most significant) 2D *Walsh Hadamard* (WH) kernels. The reasons for doing this are twofold. First, it is an extremely efficient (only 2 additions per patch per kernel) method of computing these projections [6]. More importantly, when projecting all the patches onto a line, we would like the dispersion to be as large as possible, since this would make this line very discriminative with respect to patch similarity (namely, the distance between the projection of dissimilar patches will be large, while in the case of similar patches - small). Therefore, the optimal strategy would have been to take the leading eigenvectors of the covariance matrix of the entire set of image patches. In the case of natural images (not letting the choice of lines be image dependent), these turn out to be a sinusoidal basis, ordered in increasing frequency [17]. The 2D WH kernels, when ordered by increasing frequency form such an optimal sequence of projection lines. These have been shown by Hel-Or *et al.* [12, 6] to be extremely descriptive and efficient for pattern matching in images.

4.2. Search

In the Indexing stage we built a set of L hash tables, with the desired property of *local sensitivity* in the appearance plane. Namely, that similar patches (disregarding their image location) are likely to be hashed to the same entry.

The straight forward LSH search scheme would have simply implied, for each patch in image A, considering the set of patches of image B, which are hashed to the same entry as itself in any of the L tables. This set of potential candidates is rather small, doesn't exploit the known spatial arrangement of the patches and doesn't allow propagation of information between patches. Rather, CSH creates a rich set of candidates by combining cues of both *appearance* and *coherence* (of location) in a novel manner.

Algorithm 1 ***Coherency Sensitive Hashing (CSH)***

Input: color images A and B
Output: A dense nearest patch map ANNF

Indexing (of all patches of images A and B)

1. Compute the projection of each of the patches in A and B on M Walsh-Hadamard kernels : $\{WH_j\}_{j=1}^{M}$, using the Gray Code Kernels technique of [6].
2. Create L hash tables $\{T_i\}_{i=1}^{L}$. Table T_i is constructed as follows:
   (a) Define a code $g_i(p) = h_1(p) \circ ... \circ h_M(p)$ which is a concatenation of M functions $\{h_j\}_{j=1}^{M}$ of the form:
   $$h_j(p) = \frac{WH_j \cdot p + b_j}{r}$$
   where r is a predefined value and b_j is drawn uniformly at random from the interval $[0, r)$
   (b) Then, each patch p (of both A and B) is stored in the entry $T_i[g_i(p)]$

Search

1. Arbitrarily initialize the best candidate map ANNF
2. Repeat for $i = 1, ..., L$ (for each hash table):
   (a) For each patch a in A
      i. Create a set of candidate nearest patches P_B using the table T_i and the current mapping ANNF (as described in section 4.2.1)
      ii. Let b be the patch from P_B which is most similar to a
      iii. If $dist(a, b) < dist(a, \text{ANNF}(a))$ then update: $\text{ANNF}(a) = b$ (distances are only approximated, see section 4.2.2)
3. return ANNF

4.2.1 Candidate Creation

Let g_i denote the hash code (function) used to create the hash table T_i. To simplify the discussion, we'll drop the subscript and refer to a hash function g and the resulting hash table T. Furthermore, the hash function g will be denoted g_A when applied on patches of image A and g_B when applied on patches of image B. Let g_A^{-1} (g_B^{-1}) be the inverse of g_A (g_B) and $Left(p)/Right(p)/Top(p)/Bottom(p)$ be the patch obtained as a result of shifting a patch p one single pixel to the left/right/top/bottom. In addition, let $Cand(a)$ for any patch a in A be its nearest currently known patch in B.

Here are four observations that we use to create a large pool of candidates per patch of image A. Considering

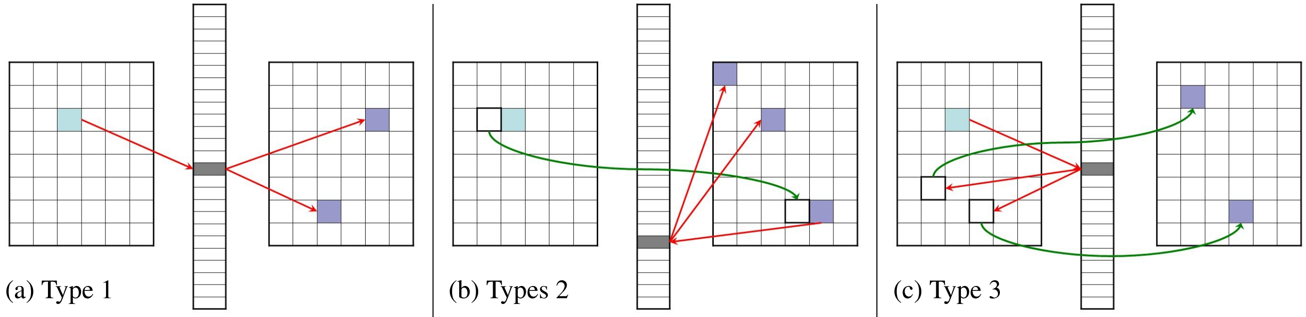

Figure 1. **Candidate types for a patch.** In each of the sub-figures, Image A is on the left, image B is on the right and the hash table in use is in the center. Arrows relating to a pixel actually relate to the patch who's top left corner is at the pixel. Red arrows represent the hashing (notice their direction), while green arrows point to the patch's current best known representative. The highlighted pixels (patches) in image B on the right are the candidates of the highlighted pixel (patch) in image A on the left. If the $width$ of the hash table is defined to be k (i.e. it stores k representative patches from each of the two images) then the total number of candidates is between $4k$ and $4k+2$ (types 1 and 3 each contribute k candidates, while type 2 appears both in left/right and top/bottom configurations and contributes k or $k+1$ in each configuration). In our implementation (and this illustration) we use $k = 2$.

patches a, a_1 and a_2 of image A and patches b, b_1 and b_2 of image B:

observ. 1 (***appearance-based***)
If $g_A(a) = g_B(b)$, then b is a (good) candidate for a

observ. 2 (***appearance-based***)
If b is a candidate for a_1 and $g_A(a_1) = g_A(a_2)$, then b is a candidate for a_2

observ. 3 (***appearance-based***)
If b_1 is a candidate for a and $g_B(b_1) = g_B(b_2)$, then b_2 is a candidate for a

observ. 4 (***coherence-based***)
If b is a candidate for $Left(a)$, then $Right(b)$ is a candidate for a [2]

Observations 1 - 3 follow from the local sensitivity property of the function g (which follows from the local sensitivity of its parts h). This happens in appearance space. On the other hand, Observation 4 follows from the coherency of patches in the image.

Here are 3 types of *candidate* patches we generate for a patch a of image A, via compositions of observations 1-4:

type	definition	using observ.
1	$g_B^{-1}(g_A(a))$	1 and 3
2	$g_B^{-1}(g_B(Right(Cand(Left(a))))$	3 and 4
3	$Cand(g_A^{-1}(g_A(a)))$	2

These candidate types are further illustrated in figure 1. In our implementation, we set the *width* of the table k (the number of patches of each of A and B that can be stored in a hash table entry) to be 2. We end up with $4k + 2$ candidates (10 in our case) and a rough estimate on the individual type contributions to the final match is 20%,50%,30%, respectively.

We can now compare the candidate patches used by CSH to those used by the different algorithms and notice how CSH generalizes them. LSH uses exactly the candidates of type 1. These candidates on their own are especially limited, mainly since they don't exploit image coherency (which is generally very high), but also since they don't take advantage of appearance similarity (hash collisions) between patches in image A. On the other hand, PatchMatch exploits only image coherency. It uses exactly 2 out of the 4-6 candidates of type 2 (Namely, $Right(Cand(Left(a))$ and $Bottom(Cand(Top(a)))$, in addition to random location candidates, using no cues of appearance whatsoever.

One clear limitation of PatchMatch, which our algorithm overcomes, is its assumption that mappings that are mostly (spatially) smooth may achieve pleasing approximations. The PatchMatch algorithm looks around the patch's neighbor's nearest patch (propagation) as well as at random patches around the current known nearest patch, with probability dropping exponentially with increase in distance. This approach works well on large contiguous areas that appear in both images, since a proper random guess will propagate to the whole area. However, it has difficulties in textured areas, which aren't replicated in both images. In our approach, we intensively relate patches which collide under some hash function. Such collisions occur based entirely on the appearance of the pair of patches without any relation to their spatial arrangement. The spatial layout of our mapping is much less continuous compared to that of PatchMatch. This is evident in the second row of figure 7, where the x-coordinates of both algorithm's mappings are presented.

4.2.2 Candidate Ranking

Given the candidate set (of size $4k + 2$), all that remains is to find the nearest one. This step of the algorithm is actually the main overall time consumer. We therefore resort to an approximation of the process, which has a negligible impact on the overall precision but greatly reduces run time.

[2] This holds also for Right/Left Top/Bottom and Bottom/Top pairs

This is where we make a second use of the Walsh Hadamard (WH) projections, which we already computed in the indexing stage. We use the WH kernels here in the way Hel-Or *et al.* use them in their rejection scheme for pattern matching [12]. The idea is that accumulating the projections of the differences of patches on the WH kernels, one at a time, produces an increasingly tighter lower bound on the Euclidean distance between the patches. We use only the leading kernels out of the full basis (in decreasing frequency ordering), which capture a large enough portion of the patch's energy. This method incorporates an early termination mechanism, rejecting a candidate once the sum of projected differences exceeds the current nearest approximation of patch distance.

4.3. Implementation Details

All our experiments were done in a fixed setting of the following options. Our hash functions g_i concatenate projections of $M = 8$ leading WH kernels (6 on Y and and 1 on each of the chroma channels). In terms of bin width r (which is equivalent to the number of bins, in our finite projection scheme), we found that the higher the frequency of the WH kernel - the lower the dispersion of the projected patches and therefore we reduce the number of bins from 32 (on first DC kernel) down to 2, at exponential rate. Also, the number of patches that fall into equally spaced bins is extremely image dependent and unbalanced in general. We handle this to improve hashing by using variable bin widths, achieving approximately a balanced distribution, based on an on-the-fly estimation of the distribution using a sparse sample of the image patches. We note that our extensive use of the WH kernels, limits our patch dimensions to powers of 2. In all our experiments, 8×8 patches were used. Aside from the clear need to store the source, target, mapping and error images in memory, CSH requires some extra memory in order to store the hash tables as well as the pre-computed projections of the image patches on the WH kernels. However, instead of constructing the complete index of L hash tables and then searching through them sequentially (as described in algorithm 1), our implementation performs L iterations (cycles) of the index and search steps, using only one table at a time. For further improvement in memory consumption, one could compute the WH projections on the fly, while making a slight change in ordering in the ranking stage. This is possible, since we use them in a sequential order that complies with the Gray Code ordering [6] of these kernels.

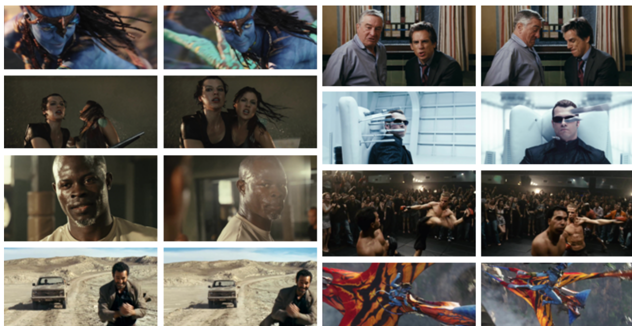

Figure 2. **Video Pairs data set** (8 out of the 133 pairs)

5. Experiments

We collected 133 pairs of images, taken from 1080p HD (∼2 megapixel) official movie trailers. Each pair consists of images of the same scene with usually some motion of both camera and subjects in the scene (The images are between 1 and ∼30 frames apart in the video). We note that pairs of images with only slight camera and subject motion aren't very challenging in the dense patch matching framework and could be handled specifically via registration or optic flow techniques. See figure 2 for some example image pairs of this database. Our implementation of CSH is in Matlab, using Mex functions in critical sections. PatchMatch implementation was taken from the PatchMatch website[3]. Both algorithms were run in a single core configuration on a 2.66 GHz machine, with 8 GB of RAM.

5.1. Efficiency

The goal of this experiment is to compare the error-to-time tradeoff of CSH to that of PatchMatch, whose tradeoff was shown [5] to be superior relative to previous methods, in the sense that it reaches reasonable error rates faster.

Our algorithm goes one step forward by being able, on the one hand, to reach reasonable error rates much faster than PatchMatch and on the other - reaching error rates that are out of PatchMatch's reach, as do the (much slower) LSH and KD-Tree algorithms.

We ran both algorithms on the Video Pairs data-set at original resolution using 8×8 patches[4]. The error to time performance of the algorithms was measured by averaging (errors and run-times) over all image pairs. The results are shown in figure 3. The mapping error (as in [4, 5]) is the average L_2 distance between the matching patches. For comparison, we also computed the exact nearest neighbor match to serve as a ground truth.

In terms of speed, it is clear that our algorithm is much faster than PatchMatch. In order to compare speed, take a certain error rate and compare how long it would take to reach it by each of the algorithms. For instance, the error rate that PatchMatch reaches after 5 iterations (as suggested in [4]) is reached by our algorithm 3 or 4 times faster.

[3] www.cs.princeton.edu/gfx/pubs/Barnes_2010_TGP/index.php

[4] Similar results were observed in different settings, when using lower image resolutions as well as different patch sizes [14]

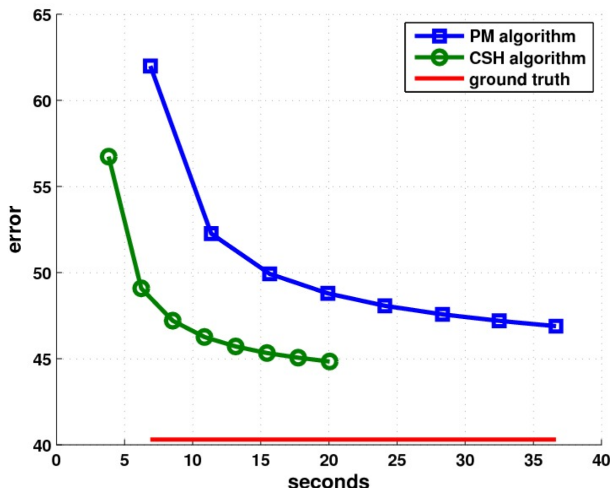

Figure 3. **Error/Time tradeoffs of PatchMatch and CSH.** Averages are over the 133 image pairs of the data set. Markers on the lines indicate the time it took each algorithm to complete an iteration, and errors are average L_2 distances between patches. Lower error rates (such as those reached by CSH on its third iteration) are reached more than 4 times faster by CSH compared to PatchMatch. Notice that CSH errors are significantly lower and approach the ground truth average error (denoted by solid red line).

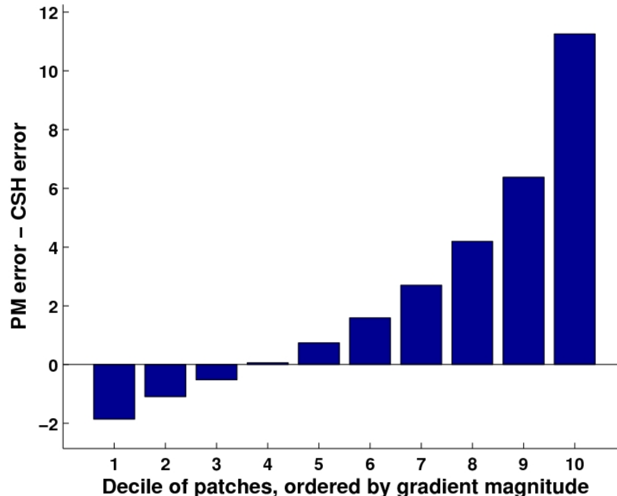

Figure 4. **Mapping errors ordered by patch energy**. x-axis: Patches of the source image are divided into 10 deciles, according to their energy level (mean gradient magnitude). y-axis: the difference between PatchMatch and CSH mapping errors, averaged over each of the deciles. On the lower end, the first decile represents patches with low energy in the range $[0, 14]$ on which PatchMatch error (mean L2 patch distances) is slightly lower (2 graylevels), while at the tenth decile (high energy in the range $[155, 255]$) - CSH error is significantly lower (over 11 graylevels).

5.2. Other Properties

Aside from its good error to time tradeoff, CSH possesses other pleasing properties, which are of high importance (not less than the error rate itself), in the common usages of such dense patch mappings. In this section we will review these properties, in comparison to the PatchMatch mapping and ground-truth (exact) mappings.

5.2.1 Image Energy and Mapping Quality

PatchMatch and CSH differ in the way the quality of a match depends on the energy level of the patch (i.e. how textured is the patch). Generally speaking, PatchMatch copes slightly better with flat areas, while CSH does better in the mid range and going towards textured, edgy patches. This is, again, due to the locality of the PatchMatch search and propagation, which will work well in large homogeneous areas, but will fail in high energy areas where usually nearby patches might only be well matched to patches that are very distant in the target image.

For our experiment we used the same 133 image pairs. For each such pair, we ordered the source image's patches according to their spatial energy (mean of gradient magnitudes) in increasing order and divided them into ten equal sized deciles. For each such decile of patches we calculated the mean error of the patch matches, produced by each of the algorithms. In figure 4, we plot the difference between the PatchMatch error and the CSH error for each of the deciles. The general trend of the plot is clear and consistent across the range of patch energies. We argue that the distribution of errors produced by CSH is preferable to that of PatchMatch, since it is known that errors along edges and textured areas have a much stronger visual impact compared to inaccuracies in textureless areas. This is the reason that CSH is able to avoid many artifacts along edges (compared to PatchMatch) when reconstructing a source image from a target image patches using the dense mapping between them (this is shown in section 5.3).

5.2.2 Incoherence of the Mapping

Given a dense patch mapping from image A to image B, we define the *incoherence* of the mapping at each pixel a of A to be the number of *different* pixels in B that a is mapped to under all of the patches that contain it. For instance, incoherence of 1 (the minimum possible) at a pixel, means that all the patches containing it map coherently (by a constant translation). The maximal coherence is the patch size. This definition is illustrated in figure 5.

The higher incoherence of the CSH mapping (compared to the PatchMatch mapping) is due to the different way in which the patches are found. In PatchMatch, the vast majority of final matches are ones that were directly propagated from neighboring patches or randomly found extremely close to them. In CSH, different good quality matches that are spatially spread in the target image have a fair chance

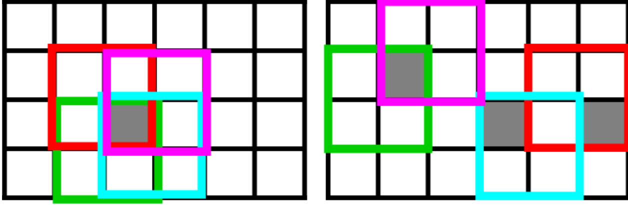

Figure 5. **Incoherence of a pixel**. In this example patches are 2-by-2. There are 4 patches containing the pixel on the left. Each of these patches is mapped to the patch of the corresponding color on the right. The incoherence of the mapping at the pixel is 3.

to be found by the algorithm. This is especially true for regions that do not appear as a whole in the target image.

Large incoherence of a dense mapping is a crucial property, when it comes to some of the applications that make use of dense patch mappings. This is true for applications, where an image area is reconstructed, pixel by pixel, according to 'votes' that come from patches in the target image of an ANN mapping. The reason being simply that the incoherence measures the number of votes a pixel gets. Therefore, for different mappings of the same error level, regardless of how the votes are integrated into a single decision (e.g. by taking the median or some weighted average) - the precision of the estimate increases with the incoherence. This (negative) correlation between incoherence and reconstruction will be shown experimentally in section 5.3. The average incoherence over the entire data-set was found to be 15% higher in CSH compared to PatchMatch.

5.3. Image Reconstruction

The combination of these CSH properties is useful in various image editing and denoising applications. We demonstrate this in the most direct manner, using the reconstruction of a source image A, given a target image B and a dense patch map from A to B. This kind of reconstruction is the main ingredient of the patch based versions of the above mentioned applications. We use the code supplied with PatchMatch to calculate the image reconstruction and its quality. It simply replaces each pixel with the average of the corresponding pixels that it is mapped to by all patches that contain it. This kind of averaging was shown [18] to maximize the (Bi-)Directional Similarity from A to B. For this experiment we used all images from the Video Pairs data-set, resized to 0.4 MP.

We use as a baseline the ground-truth (exact) mapping, which results in the best possible reconstruction under the Bidirectional Similarity framework. The results are summarized in table 1. The RMSE error is the square root of the mean (over pixels in all images) of the squared L2 (in RGB) norm between original and reconstructed pixels. It can be seen from the table that the CSH average error is more than 20 percent lower than that of PatchMatch. Figure 6 clearly shows the correlation which we discussed in section 5.2.2 between mapping incoherence and reconstruction error.

	PatchMatch	CSH	Ground Truth
reconst. RMSE	7.62	6.29	5.81

Table 1. **Average reconstruction errors** - PatchMatch vs. CSH, relative to using ground truth mapping. Averages are over the 133 image pairs data-set, at 0.4 MP. CSH achieves reconstruction error rates that are only 8% higher than those produced using the ground truth mapping, while PatchMatch's errors are more than 30% percent higher than those produced using the ground truth mapping.

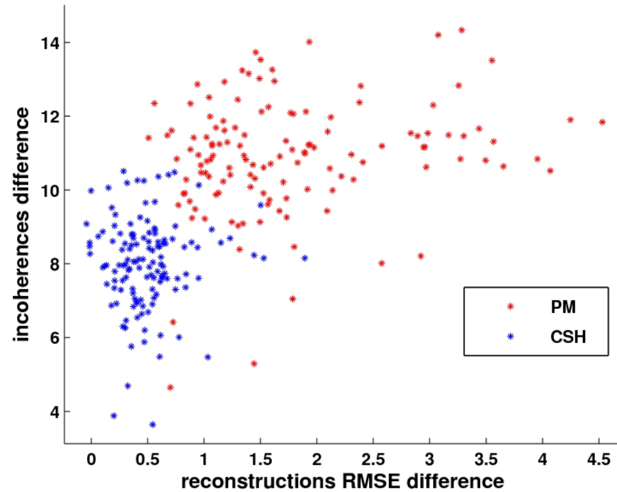

Figure 6. **Incoherence and Reconstruction Error.** Each point denotes the reconstruction error and incoherency of one of the 133 image pairs. The x-axis is the difference between reconstruction error when using the algorithm (CSH or PatchMatch) and reconstruction error when using the ground truth mapping. Similarly, the y-axis is the difference between ground truth mapping incoherence and algorithm (CSH or PatchMatch) mapping incoherence. Being close to the origin, means being close to the ground truth. The two separate clusters emphasize the negative correlation, between incoherence and reconstruction error, which we discussed in section 5.2.2.

A typical reconstruction example[5] is shown in figure 7, in which the reconstructions produced using PatchMatch and CSH mappings are compared with the reconstruction produced using the ground truth mapping.

6. Conclusions

We proposed an algorithm for computing ANN fields termed Coherency Sensitivity Hashing, which follows the concepts of LSH search scheme, but combines image coherency cues, as well as appearance cues in a novel manner. It was shown to be faster than PatchMatch and more accurate, especially in textured areas. In addition, its high incoherence improved reconstruction results, which are at the basis of many patch based methods.

[5] Please refer to CSH web page [14] for additional examples.

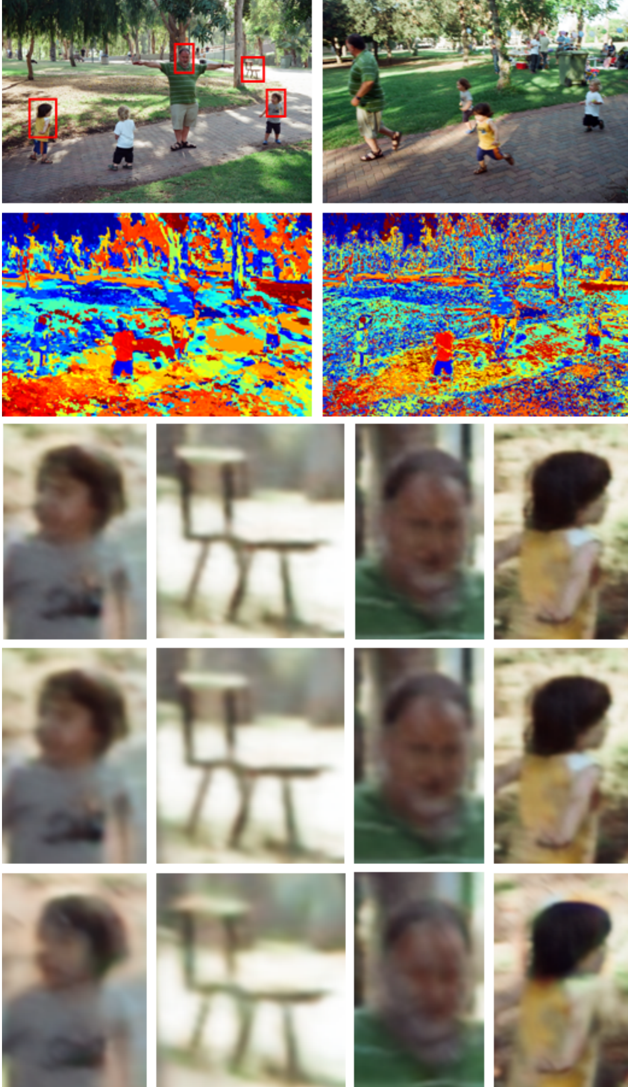

Figure 7. **Reconstruction Example.** We visually compare reconstruction results using PatchMatch, CSH and Ground truth mappings on a typical pair of 0.5 MP images. **Row 1**: The dense mappings are computed from A (left) to B (right). **Row 2**: x-coordinates of PatchMatch mapping (left) and CSH mapping (right). Blue/red areas in A are mapped to the left/right side of B. These images illustrate the lower coherency of the CSH mapping compared to that of PatchMatch. As discussed in the text - this enables better reconstruction. **Rows 3-5**: Enlarged areas from reconstructed image A, using ground-truth, CSH and PatchMatch mappings (in this order). In this example, reconstruction RMS errors are: 19.4 (ground-truth), 20.1 (CSH) and 22.0 (PatchMatch). Visually, the PatchMatch reconstruction is less accurate (especially around edges), introducing blur and color distortion.

Acknowledgments: This work was partially supported by Israel Science Foundation grant 1556/10 and the Israeli Ministry of Science and Technology. We thank Yonatan Hyatt and Guy Shwartz for their assistance.

References

[1] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45(6):891–923, 1998.

[2] M. Ashikhmin. Synthesizing natural textures. In *Proc. symposium on Interactive 3D graphics*, pages 217–226, 2001.

[3] C. Barnes. *PatchMatch: A Fast Randomized Matching Algorithm with Application to Image and Video*. PhD thesis, Princeton University, 2011.

[4] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. PatchMatch: A randomized correspondence algorithm for structural image editing. *In SIGGRAPH*, 28(3), 2009.

[5] C. Barnes, E. Shechtman, D. B. Goldman, and A. Finkelstein. The generalized PatchMatch correspondence algorithm. In *European Conference on Computer Vision*, 2010.

[6] G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or. The gray-code filter kernels. *In PAMI*, pages 382–393, 2007.

[7] A. Buades, B. Coll, and J. Morel. A non-local algorithm for image denoising. In *CVPR*, volume 2, pages 60–65, 2005.

[8] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. of annual symposium on Computational geometry*, pages 253–262, 2004.

[9] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. *SIGGRAPH*, pages 341–346, 2001.

[10] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. In *ICCV*, pages 1033–1038, 1999.

[11] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *International Conference on Very Large Data Bases*, pages 518–529, 1999.

[12] Y. Hel-Or and H. Hel-Or. Real-time pattern matching using projection kernels. *In PAMI*, pages 1430–1445, 2005.

[13] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Symposium on Theory of Computing*, pages 604–613, 1998.

[14] S. Korman. CSH webpage. www.eng.tau.ac.il/~simonk/CSH/index.html.

[15] V. Kwatra, A. Schdl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *SIGGRAPH*, 22(3):277–286, 2003.

[16] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISSAPP*, pages 331–340. INSTICC Press, 2009.

[17] D. Ruderman. Statistics of natural images. *Network: Computation in Neural Systems*, 5(4):517–548, 1994.

[18] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani. Summarizing visual data using bidirectional similarity. In *CVPR*, pages 1–8. IEEE, 2008.

[19] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H. Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Trans. on Graphics*, 21(3):665–672, 2002.

[20] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH*, 2000.

[21] Y. Wexler, E. Shechtman, and M. Irani. Space-time completion of video. *PAMI*, 29:463–476, 2007.