Mixed-Resolution Patch-Matching

Harshit Sureka and P. J. Narayanan

Centre for Visual Information Technology
International Institute of Information Technology, Hyderabad
harshit.sureka@research.iiit.ac.in, pjn@iiit.ac.in

Abstract. Matching patches of a source image with patches of itself or a target image is a first step for many operations. Finding the optimum nearest-neighbors of each patch using a global search of the image is expensive. Optimality is often sacrificed for speed as a result. We present the Mixed-Resolution Patch-Matching (MRPM) algorithm that uses a pyramid representation to perform fast global search. We compare mixed-resolution patches at coarser pyramid levels to alleviate the effects of smoothing. We store more matches at coarser resolutions to ensure wider search ranges and better accuracy at finer levels. Our method achieves near optimality in terms of exhaustive search. Our approach is simple compared to complex trees or hash tables used by others. This enables fast parallel implementations on the GPU, yielding upto 70× speedup compared to other iterative approaches. Our approach is best suited when multiple, global matches are needed.

1 Introduction

Matching patches between two images or two regions is at the core of many applications including image denoising [1], super-resolution [2], texture synthesis [3], image summarization and editing [4], etc. The problem of patch-matching is to find K most similar patches in image B, for every patch in image A. The matching is *dense* if matches for every patch in image A are calculated and is *sparse* if matches for selected interest points are calculated. It can also be *local* if the search is restricted to a region in image B, or *global* if all possible patches of image B are searched.

Calculating dense and global matches is computationally intensive. The natural trade-off is between the accuracy of the matches and the computation time. This is used by most approaches in different ways. Many applications use sparse key-point matches which can be computed fast [5]. The local or sparse matches may only provide approximate solutions for the given performance requirements, with the optimal solution requiring dense or global matching. Interactive applications such as image editing/re-targeting [4, 6, 7] need fast matching to maintain user interest. Sparse matching is used by them as dense matching is slow. Local matching combined with multi-resolution refinement is used for optical flow [8], but large displacements of small objects are often missed.

The PatchMatch algorithm [9] performs a randomized, cooperative hill climbing search to calculate dense nearest neighbor matches quickly. It relies on the

coherence between patches of an image for speedup. PatchMatch is fast for many interactive applications, but is not very accurate. More iterations are required for higher accuracy. Locality sensitive hashing (LSH) [10] finds good matches and was extended to Coherency Sensitive Hashing [11] recently. CSH replaces the random step of PatchMatch with a hashing scheme and combines cues of appearance and coherence (of location) in a novel manner. This gives a rich set of candidate patches which are searched to calculate the final patch correspondences.

In this paper, we present an adaptation of multiresolution image processing as a simple alternative for dense and near-optimal patch matching. We define optimal matches as those generated by an exhaustive search across the target image. The pyramid approach reduces the resolution of the images to be matched. Dense and global patch matches are evaluated at the coarsest level using exhaustive search. We keep at-least the best K matches at each level for each patch. The matches are transferred to the next resolution level by upsampling. The search range for the pixels of the upsampled source region at these levels is an expanded window around the upsampled target patches. This naturally brings in coherence indirectly in the matches, while searching locally. This process repeats until matches at the original image resolution are found.

The multiresolution representation can smooth the image at lower resolution levels, resulting in flat peaks in the matching score. We match mixed-resolution or mixed-scale patches to reduce this problem. We match a space-scale 3D window of pixels from a multi-scale representation of the source patch with a similar window of the target patch. We can mix multiple scales for matching. In practice, adding the immediate next higher resolution level with gives sufficiently good matches. We call this the *Mixed Resolution Patch Matching (MRPM)* scheme. Our method gives near-optimal matches at faster speeds than PatchMatch and CSH methods. The MRPM method works well different kinds of images and applications but its benefits are most pronounced when they require multiple globally similar patches to be matched to each.

The contributions of this paper are the following: (a) a simple multiresolution extension to patch matching that performs fast matching, (b) a mixed-resolution patch representation to enhance the specificity of matches achieving near-optimal matches, and (c) a fast implementation of the algorithm on multi-core CPUs and a GPU for fast performance. The simplicity of the method enables a fast parallel implementation, compared to the hashing based methods. We show how mixing resolutions is better than increasing the search window around upsampled patches. We also show how the parameters of our approach give time-accuracy trade-off. We use the dataset of 133 image pairs generated by [11] for comparison with CSH. We consistently achieve lower error values than CSH in terms of RMS distance of matched patches. Our method finds up to 8% more of the globally optimal top 10 nearest patches compared to CSH over ten image pairs with ground-truth.

2 Related Work

An exhaustive search for nearest patches is computationally expensive and several optimizations have been proposed. Xiao et al. [12] eliminate redundant similarity computation of sequential overlap between patches to find exact nearest patches fast. It is practical only for local matching and has an increased memory overhead. Many other methods for finding approximate or exact nearest patches have been suggested, a review of which can be found in [13].

Patch matching has been posed as a search in a high dimensional space of patches. Efficient data structures were used for this like the KD-tree [14], Tree Structure Vector Quantization (TSVQ) [15], and Vantage Point trees [16]. These methods can perform exact matching but are popular for approximate patch-matching. A randomized PatchMatch algorithm [9] quickly finds approximate nearest matches. It was generalized to include K nearest matches, rotations and scale [17]. PatchMatch uses a cooperative hill climbing strategy and exploits coherency to achieve speed. However, the matches found by it are not as accurate. Korman and Avidan proposed Coherency Sensitive Hashing (CSH) that includes coherency cues [11]. CSH replaces the random step of PatchMatch with a hashing scheme similar to the one used in Locality Sensitive Hashing [10]. They propagate information to patches that are close in the image plane or are similar in appearance. CSH is more accurate and is about 2 times faster, making it the best today. Several image processing methods use multiresolution representation of images for acceleration and accuracy [18]. Glasner et al. find patches at different scales for super-resolution from a single image represented at multiple scales [2].

Our method combines exhaustive search at the lowest resolution level followed by local search in upsampled search windows at other levels. The search is initialized using matches at lower resolution levels as opposed to random sampling or hashing based techniques. We extend traditional pyramid approaches by using mixed-resolution vectors for matching. The simplicity of our method also allows fast parallel implementations on multiple cores and on GPUs.

3 Mixed Resolution Pyramid Matching

We adapt multi-resolution ideas to perform a global search at reduced effort. The first step of our algorithm is to generate the spatial image resolution pyramid, starting with the original Image I_0. An image at a higher level p of the pyramid is generated using

$$I_p = \downarrow_s (I_{p-1}), \tag{1}$$

where $\downarrow_s$ is the down-sampling operator by a factor s. For down-sampling, we experimented with several standard methods like Gaussian averaging, area-averaging, cubic interpolation and nearest neighbor. All except the nearest-neighbor method did similarly on accuracy and speed. In our experiments, we down-sample using Gaussian averaging. Number of pyramid levels P is determined such that the lowest resolution image I_P has size at least $Pthold$. The

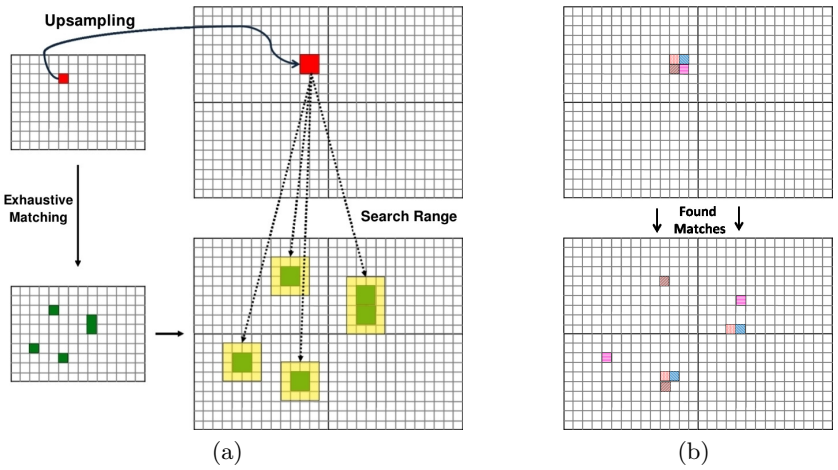

Fig. 1. 1(a): Exhaustive search is done at a reduced resolution level. The search range for the pixels of the upsampled source region at the next level is an expanded window around the upsampled target patches. 1(b): A search in this range gives the best matches

pyramid is constructed for both source and target images as Pyr_A and Pyr_B.We denote the image at level p of Pyr_A as $I_{A(p)}$.

Images $I_{A(P)}$ and $I_{B(P)}$ at the highest level are small. It is feasible to perform an exhaustive global search on them. However, it is also possible to use other techniques, such as CSH [11] or PatchMatch [9] to find the matching at this level based on the application. Every patch of $I_{B(P)}$ is considered as a candidate match for every patch in $I_{A(P)}$. K best matches are stored for each patch after the search. The search in level p is limited to a few windows, given the matches at level $p+1$. This is illustrated in Figure 1. The procedure is as follows:

1. For each pixel in level p, search only in a window around the matches transferred from level $p+1$.
2. Transfer is done by mapping the current patch a at level p to patch a^* in level $p+1$ using the downsampling rule.
3. The matched-list of a^* is upsampled to find their corresponding locations at the current level p.
4. The union of $u \times u$ windows around each of these transferred matches form the search range for matching patch a.

The candidate list for patch a at level p is given by

$$candidateList_{I_{A(p)}}(a) = \bigcup N(\uparrow_s (matchList_{I_{A(p+1)}}(a^*))), \tag{2}$$

where a^* is the downsampled patch in $I_{A(p+1)}$ corresponding to a, $\uparrow_s$ represents upsampling, and $N()$ represents a neighborhood operator. This is outlined in

Algorithm 1 Calculate $matchList_{(0)}$ (Dense matching from image A to B)

$candidateList_{(P)} \leftarrow I_{B(P)}$
$matchList_{(P)} \leftarrow$ Search $candidateList_{(P)}$ //(Algorithm 2)
$candidateList_{(P-1)} \leftarrow s * matchList_{(P)}$ //map $matchList_{(P)}$ to the next level
for $p = P - 1$ **to** 0 **do**
  $matchList_{(p)} \leftarrow$ Search $candidateList_{(p)}$ //(Algorithm 2)
  if $p = 0$ **then**
    return $matchList_{(0)}$ //original resolution level
  else
    $candidateList(p-1) \leftarrow N_{u\times u}(\uparrow_s matchList_{(p)})$
  end if
end for

Algorithm 1. Matches from level p for patch a are given by

$$matchList_{I_{A(p)}}(a) = \underset{b_i}{\text{argmin}}_K \; D(a, b_i), \quad b_i \in candidateList_{I_{A(p)}}(a), \qquad (3)$$

which are locations of the nearest K patches in image B from the candidate list. $D(a, b)$ is an arbitrary distance measure between two patches (or patch descriptors) a and b. In our experiments, the Euclidean distance is used. This is outlined in Algorithm 2.

This process is repeated until level 0 is reached. At each level and for each patch, the candidate list is generated by transferring matched-list of the downsampled pixels to the current level and considering a window around them. At the original image resolution $I_{A(0)}$, we get the final list of K best matches for each patch. For a pair of real images (image 4(d) of Figure 4 and its pair), an example run of the algorithm is shown describing a simple two level pyramid matching process in Figure 2. This method combines the advantages of fast exhaustive search at the highest level with local searches at other levels. This achieves near-optimal matching at fast speeds. However, small regions can disappear with repeated downsampling and if a match is missed at a lower resolution level, its vicinity may not be searched in higher resolution levels by this approach.

Algorithm 2 function Search. Given $candidateList_{(p)}$, Return $matchList_{(p)}$

for all patches a at level p **do**
  Form mixed resolution vectors $V(a, \uparrow_s (a), \ldots, \uparrow_s^l (a))$ //mixing l resolution levels
  Calculate match distance $D(V_a, V_{b_i})$ for $b_i \in candidateList_{(p)}(a)$
  $matchList_{(p)}(i) \leftarrow$ Nearest K matches
end for
return $matchList_{(P)}$

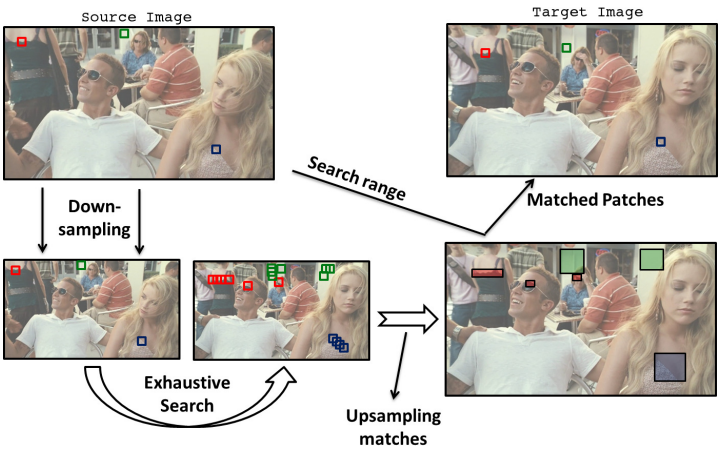

Fig. 2. Original images are downsampled and exhaustive search for matches is performed. More number of matches than required are located at coarser resolutions. The matches found here are transfered to the original resolution level by upsampling. Union of windows around them defines a search range. This is shown for three patches (in *red*, *blue* and *green*). The search for best match is done within this search range. Please note that the patches are drawn for representation purpose and are not to scale.

3.1 Mixed-Resolution Matching

The matching can be performed using patches at each level of the pyramid. The higher levels of the pyramid contain approximate or smoothed versions of the original image. This can lead to smooth distance functions and poor localization of matches. We use mixed-resolution or mixed-scale windows for matching to solve this problem. This is different from traditional multi-scale matching that looks for a patch at different resolutions. We use a vector for each patch containing pixels from representations at multiple scales. These mixed resolution vectors are matched with other such vectors to reduce the effects of smoothing. In theory, any number of resolutions can be mixed. Mixing pixels represented at resolution levels finer than the current level alone helps because

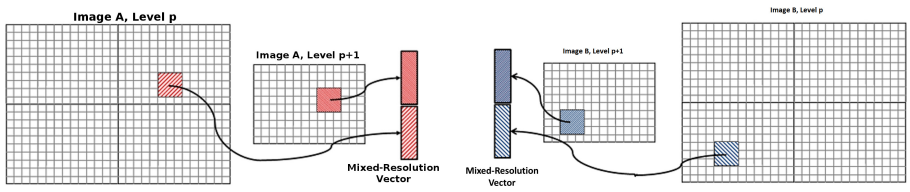

Fig. 3. Mixed Resolution Matching Process. Vectors to be matched are formed using information from the current as well as higher resolution levels

Table 1. Increasing search range and mixing resolutions both have a positive effect on accuracy but different effects on time. Error is the average Euclidean distance in RGB space between source patch and matched patches over all patches and all images. Lower error values are achieved with less cost of time by mixing resolutions. Depending upon the application these parameters can be set accordingly. Values are for $K = 5$, patch $= 7 \times 7$, $Pthold = 32$

Search Range	**No. of Resolutions Mixed (l)** Values: [Avg. Error (Time)]		
	$l = 1$ (no mixing)	$l = 2$	$l = 3$
3x3	109.47 (13 sec)	96.71 (19 sec)	93.93 (24 sec)
5x5	99.92 (22 sec)	90.38 (30 sec)	87.74 (35 sec)
7x7	96.81 (30 sec)	88.62 (45 sec)	86.18 (53 sec)
9x9	94.8 (41 sec)	87.36 (63 sec)	85.03 (73 sec)

they contain more information. Figure 3 illustrates how the mixed-resolution vectors are formed by mixing two resolution levels, which works best in practice.

In a mixed-resolution approach, windows to be matched would contain pixels from l levels of the pyramid. Such a patch is a $d \times d \times l$ cuboid (or a $d_i \times d_i \times l$ pyramid, where d_i is the size of the window from level i) in the $x-y-scale$ space. A 1D vector representation of this mixed-scale-patch is matched with other such vectors in its candidate list to find the top K matches.

Increasing the search range around upsampled matches reduces the error with extra cost in time. This behavior can be seen down the rows in Table 1. Mixing resolutions achieves even lower error values with less cost in time, as shown across the columns in Table 1. Finer resolutions surely contain more significant information for matching than areas around upsampled matches. Hence, mixing resolutions is more effective than increasing the search range. However, mixing more than 3 resolution levels incurs more cost in time without significantly improving accuracy. The choice of parameters depends entirely on the application. In general, mixing 2 or 3 resolution levels and a search range of 5×5 gives the best of time and accuracy.

The following modifications improve the accuracy and speed of the mixed-resolution patch matching process:

Increase K: The best match at the original resolution level may not be the best match at a lower resolution level. Decisions made early cannot be reversed later. Hence, at lower resolution levels, we store more matched patches by increasing K than required finally. This number is slowly reduced towards the required number towards the higher resolution levels. More matches provide a wider search range at the next level which helps in searching for accurate matches. Accuracy significantly improves when only few matches are required.

Reduce Patch-size: The same size patch at a lower-resolution level captures a bigger area. If the required patch size at the original-resolution level is $d \times d$, the size can be reduced at lower resolution levels. At the lowest resolution level,

tiny patches of source image can be matched with tiny patches from the target image. We reduce the patch size as we move up in the pyramid until the size reaches 3×3. The reduced patch-size makes matching more efficient.

Early Termination: Calculating distance between patches is expensive. We can terminate the distance calculation of a patch if it exceeds the distance of the farthest patch currently in the matched list of patches. This speeds up the matching process by eliminating dissimilar patches quickly.

3.2 GPU Implementation

Our approach works in the pixel space, which is inherently parallel. We take advantage of this by using the multiple cores of the CPU using OpenMP. This achieves near-linear time speed up with number of cores. Mixed-Resolution Patch-Matching (MRPM) processes each pixel independently which enables fast parallel implementations. We implemented our matching algorithm in GPU using CUDA with pyramid construction and mapping done on the GPU. Each thread handles a pixel and performs the matching independently. Upscaling of the matched locations and forming the search ranges for the next resolution level are also done by the GPU. We use the shared memory to store the per-pixel mixed-resolution-patch for fast access. Table 2 shows the performance of our GPU code compared against the multi-core implementation.

The GPU code is further optimized by representing each patch as a vector and making each thread handle a vector distance calculation. Parallel sorting of the distances and picking top K values gives the best matches. The CPU algorithm achieves a $4\times$ speed-up with a 4 core CPU and a further $70\times$ speed-up is seen on a commodity GPU. The GPU used is an Nvidia GTX580 with 512 cores. Benefits of GPU are best visible in global exhaustive search, which can enable strict thresholds in the pyramid and parameter values for more accuracy.

4 Experiments

We use the 133 image-pairs dataset provided by Korman and Avidan [11, 19] for experiments. We selected 10 image pairs among these (one of each pair shown in Figure 4) and calculated the ground truth matching of patches using exhaustive global search for comparison. The ground truth was calculated for 10 nearest

Table 2. GPU and CPU timings of MRPM implementation. Values for $K = 5$, Patch $= 7 \times 7$ and $Pthold = 32$ px, except in global exhaustive search

Image(s)	Search	CPU	GPU	Speed-up
Images of Fig. 4	3×3	13.46 sec	278 msec	$48\times$
	5×5	20.45 sec	684 msec	$30\times$
	7×7	29.68 sec	1113 msec	$26\times$
Lena (256×256)	Global	356.63 sec	4.89 sec	$72\times$

Fig. 4. One each from image pairs of the CSH dataset used in our experiments

neighbors using 8×8 patches. Our implementation of MRPM is in C++ using OpenMP. Code for CSH [11] was taken from their webpage [19]. Both algorithms were run on the same computational platform: an Intel i7 920 processor running at 2.67GHz with 3 GB of RAM. In these experiments, we mix pixels from two resolution levels in the matching windows. *Avg. Error* used to quantify matching quality is the average L2 distance between a patch and its matches in RGB space, averaged over all patches in the image and across all images whenever multiple images are used.

4.1 Varying Parameters

The performance of global nearest patch matching algorithms usually depends on several factors, including image size, patch size, and the number of nearest patches. Other parameters allow trade off between time and accuracy also. Effects of varying the number of required matches K and the patch-size can be seen in Table 3.

The number of pyramid levels depend upon the parameter *Pthold*. It sets the minimum image size allowed in the pyramid and can be varied to favor time or accuracy. Table 4 shows the running time and average error using different

Table 3. Table showing the effects of varying KNN and Patch-size. Error is RMS patch distance averaged over all matched patches and over all images in Fig. 4.

KNN	Patch-size	Error	CPU Time	GPU Time
1	5x5	68.1	11 sec	120 msec
1	7x7	108.4	12 sec	135 msec
1	9x9	146.8	14 sec	150 msec
5	5x5	58.1	14 sec	230 msec
5	7x7	96.1	18 sec	315 msec
5	9x9	138.3	25 sec	420 msec
10	5x5	56.7	19 sec	370 msec
10	7x7	94.5	28 sec	530 msec
10	9x9	136.5	37 sec	750 msec

Table 4. Effect of changing the pyramid threshold parameter *Pthold*. Error increases and time decreases as smaller images are allowed in the pyramid. It can be seen that choosing a 32×32 threshold works best in practice.

Pthold	**Avg. Error**	**Avg. Time**
64 pixels	45.41	30.84 sec
32 pixels	54.47	13.03 sec
16 pixels	62.13	10.19 sec

values of *Pthold* for 10-NN using 5×5 patches. Error values increase as we allow smaller images in the pyramid. Search range used is 3×3 around upsampled matches. It can be seen that stopping at a resolution near 32×32 gives best performance and accuracy.

4.2 Comparison with CSH

We compare our algorithm with the state-of-the-art Coherency Sensitive Hashing [11]. We perform the following experiments:

Proximity to ground-truth: For each image pair, we calculate 10 nearest neighbors in the target image using 8×8 patches. We then compute how many of the ground truth matches are found by this process. Table 5 shows the average across all patches and images as the percentage of top-10, top-5, and the best matches contained in the ground truth matches found by our MRPM approach and CSH approach. The best match is mostly captured by both the algorithms and MRPM holds only a slight edge over CSH. However, MRPM finds around 8% more ground-truth matches among the top-5 and top-10 matches than CSH.

Error: For each image in Fig. 4, we calculated the 10 nearest neighbors with 8×8 patches by matching image pairs. Matches of CSH are found using the code provided [19]. Figure 5 compares the average error for each image of our approach with CSH and the ground truth. Our match error is very close to ground truth error and we outperform CSH significantly. CSH focuses on finding coherent matches and misses far-off optimal matches increasing their error. We use their default setting of 5 iterations for calculating their matches. Our parameters at $Pthold = 32$ pixels and $u \times u = 3 \times 3$ achieve similar computation times as CSH for this problem.

Table 5. Percentage of ground truth matches captured by MRPM and CSH. MRPM captures up to 8% more matches when the top 10 matches are required.

Best K Matches	**CSH [11]**	**MRPM**
Best Match	92.43%	93.57%
Best 5 Matches	88.06%	91.51%
Best 10 Matches	81.72%	89.87%

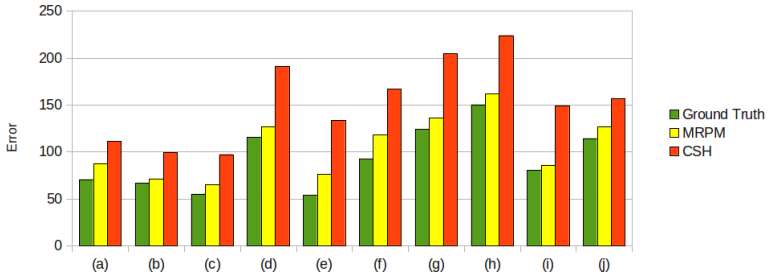

Fig. 5. Average error values of CSH and MRPM compared with ground truth.

We also perform an experiment by fixing an error value and monitoring the time taken by both algorithms to achieve it. CSH is an iterative algorithm which improves its matches with each iteration based on the framework of coherency. This framework is different from the optimality framework of MRPM. In most cases, the error values fixed are achieved at similar times by both the algorithms. However, with more iterations of CSH, average RMS error between patches does not decrease significantly, even increasing in some cases. This is due to the focus of CSH on finding coherent matches which might vary from optimal matches. Hence, the choice of algorithm depends highly upon the application.

Image Reconstruction: We reconstruct image A using image B, given a dense patch-matching from A to B. This was done over all 133-image pairs in the CSH dataset. Each pixel is simply replaced by the average of corresponding pixels that it is mapped to by all patches that contain it. Error is the RMS distance (in RGB space) between original and reconstructed pixels. See [11] for experiment details. We obtain an average error value of 6.47 while CSH achieves a marginally better value of 6.29 and PatchMatch gets 7.62 compared to the ground-truth value of 5.81. We focus on finding the optimal matches without enforcing coherency. Our performance on image reconstruction is comparable even without this facility.

5 Conclusion

We proposed a simple multiresolution approach for nearest-patch finding using mixed-resolution vectors for matching. Our simplicity enables fast parallel implementations. We find near-optimal match locations and reach lower error values than Coherency Sensitive Hashing. Advantages of the MRPM method are more pronounced when several nearest matches are needed. The advantages in time and quality over CSH diminish while finding one closest match. Several applications require matched patches to be coherent. Upsampling maintains coherency but including other cues in our framework is part of the future work. We further wish to explore matching patches across arbitrary rotations and scale and in videos.

Acknowledgements: We acknowledge the partial financial support of the DST Indo-Israeli project.

References

1. Buades, A., Coll, B.: A non-local algorithm for image denoising. In: CVPR. (2005)
2. Glasner, D., Bagon, S., Irani, M.: Super-resolution from a single image. In: Computer Vision, 2009 IEEE 12th International Conference on. (2009) 349 –356
3. Efros, A., Leung, T.: Texture synthesis by non-parametric sampling. In: In International Conference on Computer Vision. (1999) 1033–1038
4. Simakov, D., Caspi, Y., Shechtman, E., Irani, M.: Summarizing visual data using bidirectional similarity. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on. (2008) 1 –8
5. Lowe, D.G.: Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision **60** (2004) 91–110
6. Kopf, J., Fu, C.W., Cohen-Or, D., Deussen, O., Lischinski, D., Wong, T.T.: Solid texture synthesis from 2d exemplars. In: ACM SIGGRAPH 2007 papers, ACM (2007)
7. Wei, L.Y., Han, J., Zhou, K., Bao, H., Guo, B., Shum, H.Y.: Inverse texture synthesis. ACM Transactions on Graphics **27** (2008)
8. Baker, S., Scharstein, D., Lewis, J.P., Roth, S., Black, M.J., Szeliski, R.: A database and evaluation methodology for optical flow. In: In Proceedings of the IEEE International Conference on Computer Vision. (2007)
9. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B.: Patchmatch: a randomized correspondence algorithm for structural image editing. ACM Trans. Graph. **28** (2009)
10. 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 (2004) 253–262
11. Korman, S., Avidan, S.: Coherency sensitive hashing. In: ICCV. (2011)
12. Xiao, C., Liu, M., Yongwei, N., Dong, Z.: Fast exact nearest patch matching for patch-based image editing and processing. IEEE Transactions on Visualization and Computer Graphics **17** (2011) 1122–1134
13. Kumar, N., Zhang, L., Nayar, S.: What is a good nearest neighbors algorithm for finding similar patches in images? In: Proceedings of the 10th European Conference on Computer Vision: Part II. ECCV '08 (2008) 364–378
14. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. J. ACM **45** (1998) 891–923
15. Wei, L.Y., Levoy, M.: Fast texture synthesis using tree-structured vector quantization. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques. SIGGRAPH '00 (2000) 479–488
16. Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms. SODA '93 (1993) 311–321
17. Barnes, C., Shechtman, E., Goldman, D.B., Finkelstein, A.: The generalized PatchMatch correspondence algorithm. In: European Conference on Computer Vision. (2010)
18. Anderson, C.H., Bergen, J.R., Burt, P.J., Ogden, J.M.: Pyramid methods in image processing (1984)
19. CSH webpage: `http://www.eng.tau.ac.il/~simonk/CSH/index.html`.