# MIXED RESOLUTION PATCH-MATCHING

Thesis submitted in partial fulfillment
of the requirements for the degree of

*MS by Research*
*in*
*Computer Science*

by

HARSHIT SUREKA
200702015
harshit.sureka@research.iiit.ac.in

CENTER FOR VISUAL INFORMATION TECHNOLOGY
International Institute of Information Technology
Hyderabad - 500 032, INDIA
November 2012

International Institute of Information Technology
Hyderabad, India

# CERTIFICATE

It is certified that the work contained in this thesis, titled "Mixed Resolution Patch-Matching " by Harshit Sureka, has been carried out under my supervision and is not submitted elsewhere for a degree.

| | |
|---|---|
| Date | Adviser: Prof. P. J. Narayanan |

To *Mom and Dad*

# Acknowledgments

First of all, I would like to extend my sincere gratitude towards my advisor, Dr. P. J. Narayanan who has been a constant support and motivation throughout the duration of my work. I thank him for insightful discussions on topics related to work and beyond. He is an inspiration and has influenced my personal and professional development, shaping my attitude towards research and life, in general. I would also like to thank the other professors at CVIT; Prof. C.V. Jawahar, Prof. Jayanthi Sivaswamy and Prof. Anoop Namboodiri for teaching me the basics, being motivational and creating a wonderful atmosphere in the lab which makes working fun!

I wish to thank Eugene Vendrovsky at Rhythm and Hues studios, who was the key person to start this work. Initial work and discussions with him sparked the whole process and brought in ideas which resulted into this thesis. I also thank other member of his team for initial collaboration with us and hope that more such joint projects are carried out in the future.

Being in IIIT-H and working at CVIT, I had the chance to know several interesting people with various outlook towards life. I thank my senior Siddharth Ch. for support in the lab and Kshitij Chandrasen for being helpful and believing in me. This journey was fun with fellow dual-degree students Abhinav, Aditya Pratap, Akhil, Ankur, Chandra, Gautam, Kogta, Kulkarni, Manan, Mujumdar, Mayank, Nigam, Parikshit, Pulkit, Rakshit, Romit, Sahni, Sankalp, Yasir and so many others, all of whom I couldn't mention. I thank my friends from back home and Addhyan, Rahul and Prasoon, who graduated early but kept in touch and showed faith in me. I specially mention Ruhi who helped me sail through so many different phases and motivated me constantly!

I owe this thesis to my family who has supported me throughout and encouraged me in every aspect. I thank Dad taking out time to hear about my work and Mom getting excited on achieving every small milestone, which boosted my spirits. I thank my Dadaji for being an inspiration and teaching me to be organized, Dadiji for her immense love and guidance and Bhaiya-Bhabhi for fun times and support. This thesis is a result of my family being there for me, always!

# Abstract

The problem of matching small image regions or patches between images is central to many image editing and computer vision tasks. In this thesis, we propose two related solutions to the problem of fast and global patch-matching. First, we present a multi-resolution spatial pyramid based solution, which we call *Pyramid Patch-Matching (PPM)*. It uses a simple coarse-to-fine framework powered by exhaustive search at the coarsest resolution and local directed search at finer resolutions. PPM quickly finds accurate patch correspondences between images or image regions. The search step in our algorithm is flexible to accommodate the use of other nearest-neighbor algorithms within its framework. A proposed *increase K* technique enables PPM to store more number of matches than required at coarse resolutions which maintains a wide search range. This improves accuracy by up-to 2%. Its effect is most significant when less number of matches are required. The *early-termination* technique preempts the distance calculation between patches, if the distance exceeds that of the farthest match currently stored. This speeds up the algorithm by up-to $1.7\times$. Following the downsampling rule, the *reduce patch-size* technique matches smaller patches than required at coarser resolutions of the pyramid. Empowered by these techniques, PPM achieves state-of-the-art accuracy outperforming previous standards of Coherency Sensitive Hashing (CSH) [36] and PatchMatch [12] without compromising on execution-time. PPM finds up-to 4% more of the ground-truth matches and achieves lower error values compared to CSH, the current standard.

Several parameters are provided to tune the search for good matches based upon the application. We perform experiments to study effects of these parameters on matching accuracy and computation time. While consistently achieving lower error values for RGB distance between matched patches, the simplicity of our algorithm also enables fast parallel implementations, providing further advantage over previous randomized, hashing or tree-based iterative approaches. On CPU cores, it achieves a near-linear time speed-up w.r.t. the number of cores and a GPU implementation provides a further speed-up of up to $70\times$ compared to a CPU implementation run on a quad-core machine ($250\times$ w.r.t. single core).

We observe that the matching accuracy of PPM suffers from low confidence matching of patch-vectors at coarse resolutions. To solve this, we propose a novel way to mix information by concatenating corresponding patch-vectors from finer resolution levels with patch-vectors of coarse levels. We call these *Mixed-Resolution Vectors* (MR-vectors). Unlike traditional patch-vectors that contain information from a single resolution level, MR-vectors contain more relevant information for matching. We build over our PPM algorithm by substituting MR-vectors in place of traditional patch-vectors. We call this

the *Mixed-Resolution Patch-Matching* (MRPM) algorithm. MRPM further improves the accuracy of matches found by PPM with a marginal increase in time. It captures up-to 4% more of the ground-truth matches (8% more than CSH) and achieves near-optimal match accuracy. We also show that the MR-approach has computational benefits over an approach which uses an increased search range. We describe the effects of changing the number of resolution levels mixed and show that a balance between accuracy and time is achieved at mixing 2 levels. We perform various experiments to compare MRPM with PPM and other previous approaches, in all of which MRPM consistently performs better. Fast parallel multi-core and GPU implementations of MRPM follow from parallel implementations of PPM. We develop applications of image denoising, image summarization and auto-crop based on our MRPM algorithm. Results with better visual quality are obtained when MR-vectors are used. The computational bottleneck of these algorithms is the search for nearest neighbors of patches. We demonstrate the utility of MRPM as a fast patch-matching engine which empowers these applications. We open up new possibilities of research to explore the scope of MR-vectors which we believe has applications in several other image editing and computer vision tasks.

# Contents

# List of Figures

# List of Tables

*Chapter 1*

# Introduction

Digital photography has advanced at a tremendous pace in the past decade. Advanced techniques have been developed to understand and edit images and videos which, in the past, seemed like a distant dream. To name a few, *Image Denoising* has reached a stage where information lost due to noise can be recovered with good accuracy [18, 23, 43]. *Image Retargeting* has made possible resizing images preserving important information and to various aspect ratios [55, 52, 63]. With *Image Completion* techniques, a user can completely remove unwanted objects in an image and even reconstruct missing frames of videos [66]. Algorithms synthesize data from other parts of the image filling in missing information in a way that blends naturally with the rest of the image [33, 21].

The basic element of the image is a pixel. A group of pixels, typically rectangular (say, $7 \times 7$), is called a *patch*. An image can be broken down into small patches and many algorithms manipulate the image by *patch-based synthesis*: rearranging and averaging image patches. A problem central to these approaches is patch-matching, which finds similar patches between images and image regions. An illustration of this is shown in Figure 1.1 . For a brute-force matching, this problem is $O(nd^2)$ where patches to be matched are of size $d \times d$ and the image has $n$ pixels. With the increasing resolution of images, this is very computation intensive. Patch-based methods have recently become popular, mainly due to the design of fast patch-matching algorithms in recent times. A variety of applications have been developed around it. Figure 1.2 shows a collection of problems solved with algorithms having a patch-matching engine.

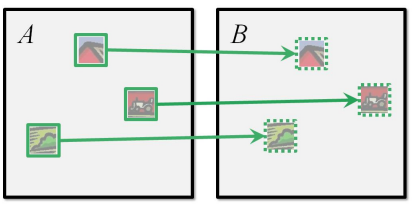



**Figure 1.1** Sample figure to illustrate the patch-matching problem that we solve. Given a source image A and a target image B; for every patch in image A, we find the "closest" patch(es) in image B. (Image credit: [11])

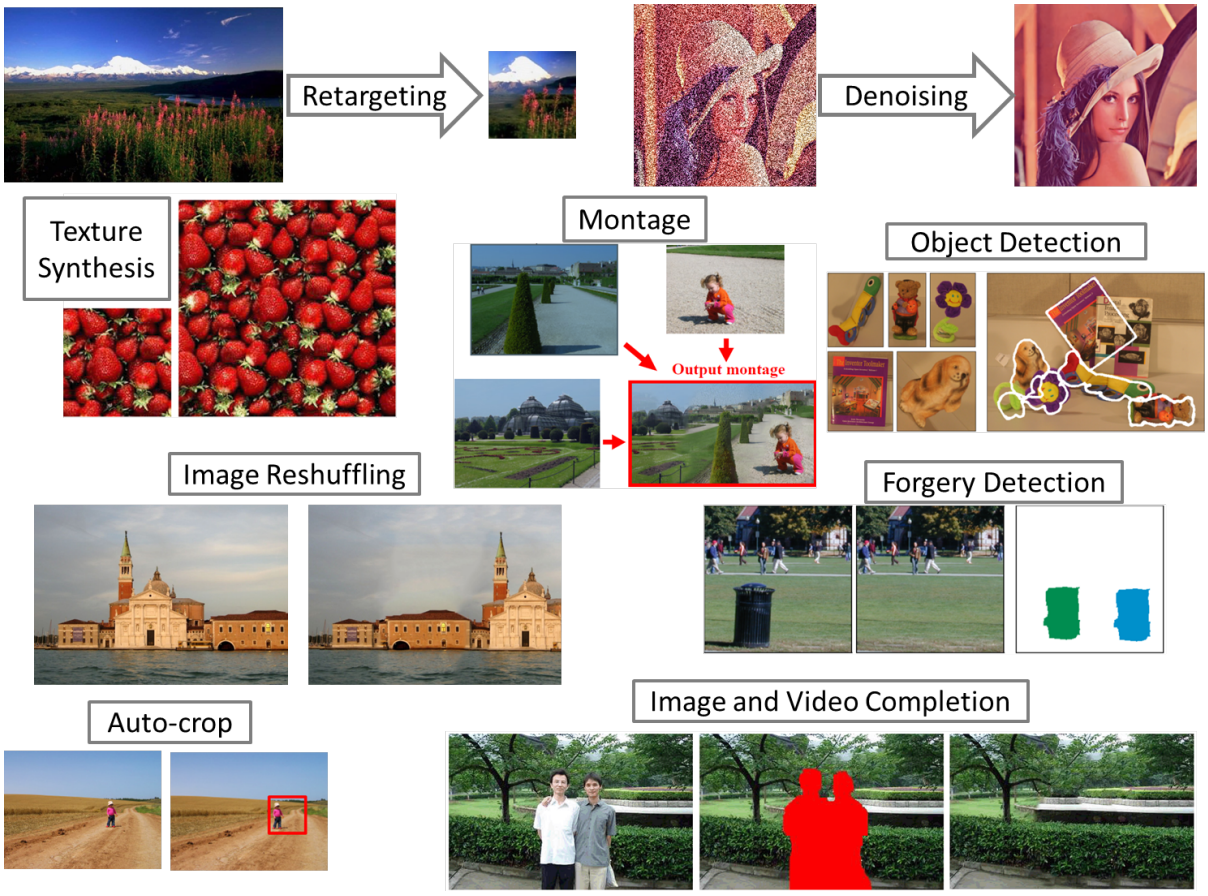
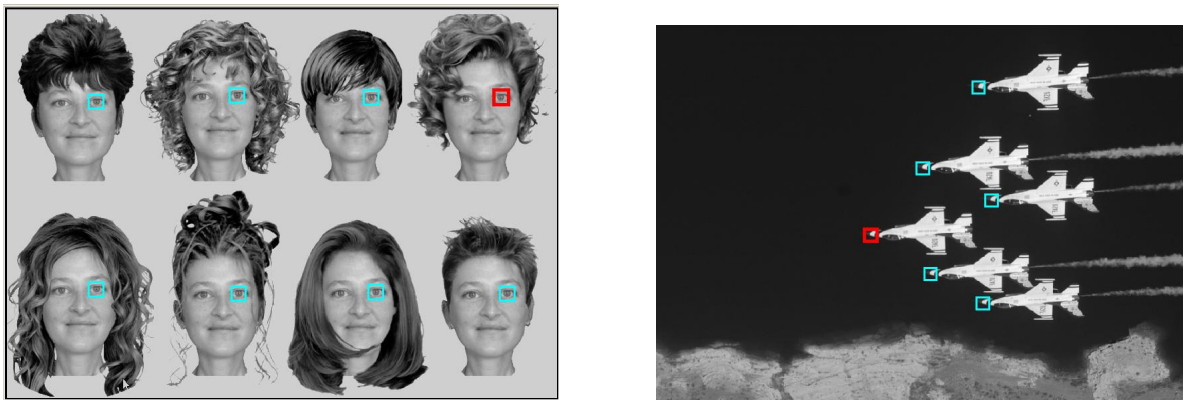



**Figure 1.2** The scope of patch-matching algorithms. A variety of problems are solved by methods that contain patch-matching algorithms at their core. (Image credits: Image Montage [55]. Object Detection, Forgery Detection and Image Reshuffling [13]. Texture synthesis and Image completion [67])

Figure 1.3 shows an example of patch-matching in two cases where a patch is searched for its nearest neighbors in the same image itself. We perform an experiment and observe that state-of-the-art approaches are almost 1400× faster than a brute force approach. With fast parallel algorithms now being developed, this speed-up increases many folds.

In the past, it was customary to compute patch correspondences with traditional approximate nearest-neighbor (ANN) tools such as Locality Sensitive Hashing (LSH) [24] and KD-trees [45, 29]. These tools perform well in terms of accuracy but are not as fast as one would hope. Images have specific structures and properties which can be taken advantage of to develop ANN algorithms specific to finding similar patches in images. Several such extensions to traditional ANN algorithms have been proposed and several new approaches have been developed to approach the problem. Existing literature is reviewed in Chapter 2 and a survey of these algorithms can also be found in [37, 20, 60]. Recently, a fast patch-matching algorithm motivated by coherency of patches was developed by Barnes et al. [12]. It performs a randomized, cooperative hill climbing search to calculate dense nearest neighbor matches



(a) (b)

**Figure 1.3** Two images ($600 \times 800$) describing the patch-matching problem. The *red* patch ($21 \times 21$) is matched with all other patches in the same image and the best matches are marked in *blue*. A brute force search for matching one patch of eyes in (a) takes 2125 ms while state-of-the-art approaches are able to find the matches in about 1.5 ms ($1400\times$ faster). (Image credit: [37])

quickly. This algorithm, termed "PatchMatch", is fast for many interactive applications, but is not very accurate. More iterations are required for higher accuracy, costing more computations. A hashing based algorithm, motivated by Locality Sensitive Hashing (LSH) [24] was developed by Korman and Avidan to extend PatchMatch. This algorithm is termed Coherency Sensitive Hashing (CSH) [36]. It 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 accurate patch correspondences. This method is up to 2 times faster than PatchMatch and finds significantly more accurate matches, making it the current state-of-the-art.

The required matches need to have certain properties which depend upon the application. These approaches focus on finding coherent matches which differ from optimal matches. Coherent matches are those which give higher weight to the matches which agree more with the matches of their neighbors. Optimal matches are points closest to the query point in the patch-space over the whole range. In addition, these approaches are iterative and are not easy to parallelize. With single-core processors almost outdated and many systems now being fit with a GPU, parallel algorithms are becoming the norm. Hence, there is a need to develop a fast and optimal, easy-to-parallelize patch-matching algorithm.

In this thesis, a novel pyramid based method for patch-matching is proposed. This algorithm is fast and searches for dense, global and near-optimal patch correspondences. We call it The "*Mixed-Resolution Patch-Matching (MRPM)*" algorithm. MRPM applies the coarse-to-fine framework to form a spatial image resolution pyramid which is processed in a novel manner by mixing pyramid resolutions. The pyramidal approach keeps the algorithm simple and mixing of resolutions helps in accurate localization of search. MRPM finds significantly more accurate matches than previous techniques with performance matching up to the state-of-the art approaches. It is most suitable for applications that require global and optimal matching. The simplicity of MRPM enables a straight forward parallel im-

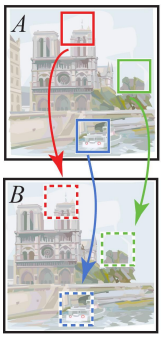



**Figure 1.4** Patches in image A (source image) are matched to patches in image B (target image). (Image credit: [12])

plementation which provides linear speed-up with respect to the number of cores on the CPU and an additional 70× speed-up on multipurpose GPUs.

The novel manner in which the pyramid is processed allows for exploration of this mixed-resolution approach in other multi-resolution tasks. Algorithms in several applications of image-processing and computer vision, down-sample original data for fast processing or to obtain an initial guess of the result. The mixed-resolution approach can be used in such problems to improve the accuracy of the initial guess, eventually improving overall accuracy. For other classification tasks also, using mixed-resolution vectors can increase the matching accuracy. We present the Mixed-Resolution approach for the specific problem of Patch-Matching but strongly believe that it can be applied to benefit a variety of other problems.

## 1.1 The Patch-Matching Problem

A patch is a group of continuous pixels in the image plane. In a patch-space, each patch can be represented as a point. Patch-matching is a nearest-neighbor problem which searches for one or more nearest patches in the patch space. This is illustrated in Figure 1.4. More formally, it can be defined as finding the set of points $P_c$ within a dataset $P$ that are "closest" to a query point $q$, as measured by some distance function $d(p, q)$ in the patch-space.

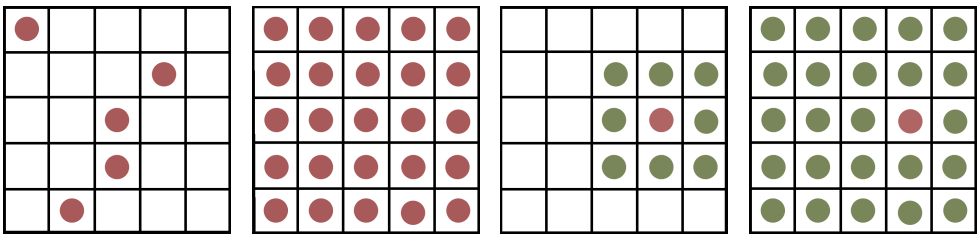
$$P_c = \underset{K}{\operatorname{argmin}}\, d(p, q), \quad p \in P \tag{1.1}$$

The distance function $d(.,.)$ is often defined over the patch elements but could also be defined to calculate distances between metrics like feature descriptors, defined over image patches. $d(.,.)$ is often one of $L_1$, $L_2$ or $L_\infty$ but the only requirement typically imposed is that $d$ be a true distance that satisfies $(i)$Symmetry ($d(a, b) = d(b, a)$), $(ii)$Non-Negativity ($d(a, a) = 0$ and $d(a, b) > 0, a \neq b$) and $(iii)$Triangle Inequality ($d(a, b) \leq d(a, c) + d(b, c)$).

In particular, the triangle inequality allows nearest-neighbor (NN) algorithms to compute bounds on various distances without having to evaluate the distance function itself, saving redundant calculations. A patch-matching algorithm finds for patches in image A, the nearest neighbors in B under this patch distance metric $d(.,.)$. In the literature, this mapping is referred to as the Nearest-Neighbor Field (NNF). Computing an exact-NNF is computationally expensive and several approaches find approximate-NNF (ANNF) for efficiency. Computing ANNFs is still challenging because the number of patches in an image is in the millions and for interactivity, one needs to find accurate patch correspondences for each patch in real or near-real time. Patch-matching is an important building block in many computer vision and graphics applications [67, 1]. General purpose NN algorithms [5, 45] have been modified to take advantage of the specific properties and structures in images. Properties required of the computed ANNF are dependent on the application it will be used for. Hence, patch-matching algorithms are often developed generally and provide parameters to tune their search.

Various types of search are performed by different patch-matching algorithms. In the case of overlapping patches in which every pixel in the image represents a patch, the search problem is more critical. If matches for every patch in image $A$ are to be found in image $B$, then the required matching is called *dense*. Some applications require matches to be found only for specific interest points and require only a *sparse* patch-matching. A *local* patch-matching uses a fixed window in the image plane, typically around the query patch, in which the matches are to be searched. But, far off optimal matches are found by performing a *global* search in image $B$. These can be seen in Figure 1.5.

Calculating dense and global matches is computationally intensive. The natural trade-off that exists between the accuracy of the matches and the computation time has been exploited by different applications in different ways. Many applications, such as object recognition or detection use sparse key-point correspondences which can be computed reasonably fast [40]. The local or sparse nature may provide approximate solutions for the given performance requirements, but the optimal solution may require dense or global matching. Several interactive applications such as image editing/re-targeting need fast matching to maintain user interest [56, 34, 64]. Sparse matching is used by them as dense matching



(a) Sparse (b) Dense (c) Local (d) Global

**Figure 1.5** Types of Patch-Matching: (a): A sparse patch-matching finds correspondences for only a few interest points. (b): If matches for all patches are required, then the matching is called Dense. (c): If the search for similar patches is restricted within a search range around the query patch, the matching is said to be local. (d): When every patch of the target image is considered a candidate for a match, the matching is global. Query patches are shown in red and candidate matches are shown in green.

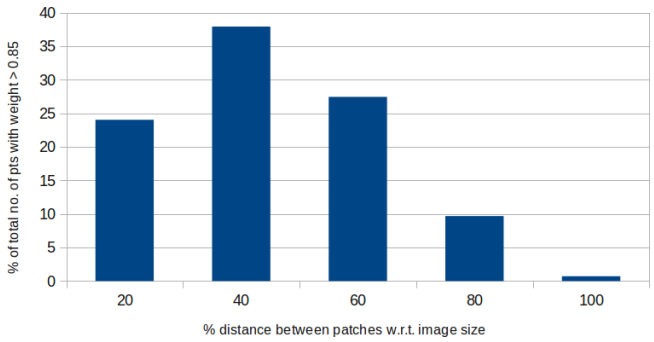



**Figure 1.6** Similar points maybe found far-away from the current patch. Large number of similar patches are found at a distance of 40-60% of image-size.

is slow. Local matching combined with multi-resolution refinement is used for optical flow, but large displacements of small objects are often missed [9].

We perform an experiment to understand where most similar patches in an image typically lie. The assumption of local patch-matching algorithms that best fitting patches are spatially close is true for most images but several counter-examples exist. In textured images specially, similar patches are found far-away in the image. This is also true for natural images. Figure 1.6 shows the percentage of average number of similar patches plotted against percentage distance w.r.t image size. It is the average of all $7 \times 7$ patches for images shown in Figure 5.2. The patches were assigned weights based on similarity score and only patches with $weight > 0.85$ were included in the plot. As seen in the figure, a significant fraction (40-60%) of similar patches lie at large distances. For most patches that are in the center, this covers the whole image. Hence, a local matching would not suffice to find all the optimal matches. Our Mixed-Resolution Patch-Matching (MRPM) approach searches globally for best matches and quickly finds near-optimal matches.

### 1.1.1 Parameters of Patch-Matching Algorithms

Due to the many ways in which the matches can be found, patch-matching algorithms have several parameters which can be controlled by the user and depend mostly upon the application. Here, we introduce these parameters and assign notations to them, to be used throughout the remainder of this thesis. Parameters common to all matching schemes are described below.

**Patch-size**: This is the most basic parameter which defines the dimensions of the patches that are to be matched. We represent each patch to be a $d \times d$ block of pixels. An ideal patch size depends upon the local scale of objects in the image. Using a large patch allows a more robust discrimination between areas that are not actually similar, which is particularly interesting in the presence of noise. Hence, they are more tolerant. On the other hand, small patches are more selective and take into account the importance of little contrasted small details. Large patches are also expensive to match and small patch-sizes (e.g. $7 \times 7$) generally perform well for most applications.

**Number of Nearest-neighbors:** The number of matches that are required for each patch is another key parameter for patch-matching algorithms. We denote it by $K$ and the search problem is referred to as finding $K$-nearest-neighbors ($K$-NN). For applications such as super resolution, image summarization and tracking just the best match per query patch is required. Searching only for the most optimal match is fast but applications such as object detection, denoising, and symmetry detection require multiple nearest neighbors. In such cases, instead of inefficient insertion sort, optimized heap algorithms are often used but complexity inevitably increases with increasing $K$. Some applications even require an $\epsilon$-NN search which finds all matches with $d(p, q) < \epsilon$.

**Search Range:** For local patch-matching, a search range $u \times u$, defines the restricted region in the target image which is searched for similar patches. This is often done to save computations. Local algorithms assume that good matches are found around the current patch in the image plane. This assumption is not valid for textured images and even in natural images, a similar patch can be found far-away from the query patch (as shown in Figure 1.6). Hence, global patch-matching may be required where the search range is the entire target image. Some initial processing might be performed to locate regions with potential candidates. These form discrete search ranges spread around the image to be searched. Our MRPM approach uses matches up-sampled from lower image resolutions to form these windows. Search ranges can be set to favor time or accuracy with larger search ranges being more accurate but expensive.

In addition to these, each algorithm introduces some of its own parameters. For example, Patchmatch [12] and CSH [36], being iterative approaches have the number of iterations as a parameter. Tree-based approaches require a threshold parameter to prevent further splitting. Our Mixed-Resolution Patch-Matching (MRPM) approach introduces parameters for number of levels in the pyramid and the number of resolution levels mixed to form mixed-resolution vectors.

## 1.2 Contributions of this Thesis

The thesis presents our previously published work describing the simple Mixed-Resolution Patch-Matching (MRPM) algorithm [59]. It also shows MRPM at work in applications. The main contributions of this thesis are:

- Pyramid Patch-Matching (PPM, Chapter 3): Motivated by the wide-spread use of pyramids in image processing and computer vision, we propose to use this coarse-to-fine framework for the problem of patch-matching. This finds more accurate matches than state-of-the-art efforts at speeds similar to them.

- Mixed-Resolution Vectors (Section 4.2 and 4.3): We propose a novel way of mixing resolution levels in pyramid processing to improve accuracy. For patch-matching, we find better matches than PPM at marginal cost of time. We strongly believe that these have a lot of scope in other research areas too.

- Efficient Implementation: We provide an efficient multi-core and GPU implementation of our MRPM algorithm with the GPU code being up to $250\times$ faster than a single-core implementation.

.

## 1.3 Organization of this Thesis

This thesis is organized as follows. The next chapter looks at the varied uses of pyramids in image processing and reviews existing literature on nearest-neighbor and patch-matching algorithms. It discusses the more recent PatchMatch [12] and CSH [36] algorithms in detail.

Chapter 3 introduces the use of pyramids for the problem of patch-matching intuitively called Pyramid Patch-Matching (PPM). Some improvements to PPM are proposed. We show that in terms of RMS error of matched patches, PPM finds more accurate matches than previous techniques. Its performance is similar to state-of-the-art approaches on multi-core CPUs. The simplicity of PPM enables a fast parallel implementation on the GPU which is up-to $70\times$ faster than a multi-core implementation run on a 4-core CPU.

To minimize the effect of smoothing due to downsampling in the pyramid, Chapter 4 introduces Mixed-Resolution (MR) vectors which extend PPM to Mixed-Resolution Patch-Matching (MRPM). Including information from finer resolution levels further improves the match accuracy with a marginal increase in time. Various parameters involved in this approach are introduced and experiments to understand their effect on matching accuracy and time are performed. MRPM is compared against previous techniques where its benefits come out clearly.

Various image processing and vision applications that use a patch-matching algorithm at their core are reviewed in Chapter 5 and the benefits of using MRPM are shown for some of them. Chapter 6 concludes the thesis and encourages researchers to explore the applicability of MR approach in other domains.

*Chapter 2*

# Background and Related Work

In this chapter, we review the use of image pyramids for efficiency, data-compression and accuracy. Our Mixed-Resolution Patch-Matching (MRPM) algorithm extends the utility of pyramids to the problem of patch-matching. We also present a literature review of the several nearest-neighbor algorithms that have been proposed for finding similar patches in images, discussing the more recent PatchMatch [12] and the state-of-the-art Coherency Sensitive Hashing (CSH) [36] algorithm in detail. Patch-matching algorithms are at the core of several image processing and vision applications which we describe in Chapter 5. Their background and related work is presented there.

## 2.1 Pyramids in Image Processing

A pyramid is a multiresolution representation of an image which separates information into frequency bands. Repetitive down-sampling of an image to represent it at multiple scales forms the image pyramid. Figure 2.1 shows a sample image pyramid.

Pyramids have been in use for over three decades and have made several image processing and compression tasks efficient [3]. Applications of image pyramids can be found in most domains of image processing and computer vision. Simple problems of data compression, mosaicking and image

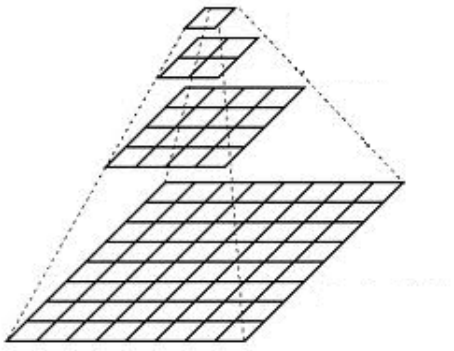


**Figure 2.1** A sample image pyramid. Higher levels are down-sampled versions of the original image.

alignment to complex problems of optical flow make use of this multi-scale representation of images [15]. While some problems such as object tracking, use pyramids to include large motions of objects in their search, others use them to represent data at a reduced scale for better matching accuracy for image and video classification tasks [54]. The downsampled image or video sample is almost noise-free and contains only higher level information required for matching. Pyramids, therefore are used in a variety of ways and have evolved to be inseparable from the tasks of manipulating and analyzing images [28, 26].

Motivated by this general applicability of image pyramids, we explore their use for the problem of patch-matching and show how they enable near optimal matches to be found quickly. Instead of performing an expensive exhaustive search at the original resolution level, we perform it at a reduced resolution level by forming a pyramid and restrict searches in finer resolution levels of the pyramid to windows defined by upsampling matches found at the coarser resolution levels. Pyramids are generally processed one level at a time. We extend this idea to perform *mixed-resolution pyramid processing* to include cues from finer resolution levels while processing coarser pyramid resolutions. We show how this increases the accuracy of the matches found at coarser resolutions leading to better match localization at the finer resolutions and believe that this mixed-resolution approach has applicability in several other domains.

## 2.2 Methods for Finding Similar Image Patches

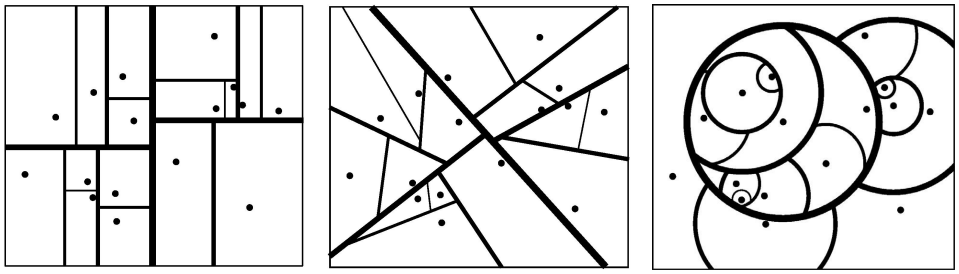
Several nearest-neighbor(NN) algorithms have been developed for finding similar patches between images. This problem has also been a subject of many surveys [20, 60, 37]. The matching that is required depends significantly on the application it will be used by. Different applications require different properties in the found matches. Hence, patch-matching algorithms are developed generally and have several parameters that can be tuned to favor time or accuracy. Parameters can be set to perform a *range search* which finds all matches within a distance metric threshold ($d(p, q) < \epsilon$) or to perform a *K-NN search* which finds the best K matches for a given query patch. Patches of different sizes can be matched using various distance functions in the patch space. A brute-force approach for finding 1-NN matches in an image takes $O(nd^2)$ where patches to be matched are $d \times d$ and $n$ is the number of pixels in the image. This is computationally very expensive and patch-matching algorithms strive to reduce the complexity of this problem without compromising much on the accuracy of matched patches.

In applications like texture synthesis [25], the texture example is usually a small image. In other applications such as patch-based completion [21], retargeting and reshuffling [55, 12], the input image is typically much larger so the search problem is even more critical. Speed and accuracy are hence desired properties of patch-matching algorithms. To achieve this, one might intuitively use general-purpose NN algorithms [68, 58, 14]. However, these algorithms would not take advantage of structures and properties specific to images and applying them directly to find similar patches would not achieve best results. Motivated by these, patch-matching is generally posed as a search problem in a high-dimensional data

space. Several NN approaches for similar patches in images organize the dataset into a hierarchical tree based structure These structures store data only at leaf nodes and use internal nodes to prune the list of leaves returned by a query. Most of the these methods perform both exact as well as approximate NN search, as demanded by the application. More recently, methods based on randomization [12] or hashing [24, 36] have become popular. These perform fast approximate searches for similar patches between images and have opened up the space for several new patch-based applications to be developed by making interactive applications for image-editing possible.

### 2.2.1 Tree-based Methods

Figure 2.2 gives an intuitive idea of how different methods organize data in a hierarchical structure. $k$d-trees [29, 34] generally perform splitting along a data dimension, usually the one with maximum variance. The split value is generally chosen to be the median value along the split direction. These are still one of the most popular NN methods after they were first proposed over three decades ago by Bentley, J. [14]. However, node divisions in $k$d-trees are always axis-aligned, regardless of the data-distribution. This is a simple representation but leads to poor search performance in several cases. Recently, an extension to $k$d-trees was proposed by Adams et al. [1] called the Gaussian-$k$d-tree structure. It combines effects of storing more than one element per node and making local reorganizations during insertions. It is memory efficient and performs fast search.



(a) $k$d-Tree (b) PCA Tree (c) $vp$-Tree

**Figure 2.2** Different trees for partitioning a 2D point space. The thickness of the line is proportional to the partition order (thicker lines were partitioned first). Even though all methods are tree-based, the difference between the structures they create is notable. Image courtesy: [37]

PCA trees [58, 38] try to relax the axis-aligned assumption of $k$d-trees by performing a Principal Component Analysis (PCA) [31] to reduce dimensionality of data. They perform splitting along the eigenvector corresponding to maximum variance. An example to illustrate the partition process for PCA-trees is shown in Figure 2.2(b). Particularly in synthesis applications, approximate search is often used in conjunction with a dimensionality reduction technique like PCA [38] to make the NN method more time and memory efficient.

Other methods such as Ball Trees [49] and Vantage Point Trees ($vp$-trees) [68] partition the data points based on some metric defined on pairs of points. The $vp$-tree method, in contrast with other

metric-based approaches, splits data points using the absolute distance from a single center. The center for each node is often chosen randomly and various heuristics are employed for setting other parameters.

Clustering methods such as the k-means method [41] are also used to cluster data in the high-dimension space and form the hierarchy. Another method called Tree-Structure-Vector-Quantization (TSVQ) [65] exists which is essentially a data-compression technique. It takes a set of training vectors as input, and generates a binary-tree-structured codebook by using the centroids of the set of training vectors. Kumar et al. [37] perform several optimizations on tree based methods and claim that the $vp$-tree structure, which is not well-known in the vision community, gives the best overall performance for finding similar image patches.

### 2.2.2 Other Methods

Although tree based methods are most common in the literature for NN search, several other approaches exist. Nene and Nayar [47] mention a simple approach to find local correspondences of a point in high-dimensional space using maximum norm. This method works well for several applications, but is less suited for cases where the true $L_2$ distance is more important. Other techniques such as those of Arya et al. [5] and Datar et al. [24] perform approximate-NN (ANN) search and provide fast matches to applications that do not require exact matching. In contrast to all these approaches that represent data in higher dimensions, Xiao et al. propose an algorithm for exact NN search by performing optimizations on the brute-force search [67]. They eliminate redundant similarity computation of sequential overlap between patches to find exact nearest patches fast. This produces better results for applications that require exact matching. However their algorithm is designed specifically for local matching and has an increased memory overhead. On the other hand, we optimize the brute-force search using a pyramidal framework and propose novel ways of mixing resolution levels to achieve near-exact matching. We use heuristics like early-termination to avoid redundant distance calculations and quickly find accurate matches.

Ashikhmin [6] proposed a local propagation technique exploiting local coherence in the synthesis process by limiting the search space for a patch to the source locations of its neighbors in the exemplar texture. This motivated the work of a patch-matching algorithm which is popularly known as PatchMatch [12].

### 2.2.3 PatchMatch

PatchMatch is a dense and global ANN method which performs a randomized, cooperative hill climbing search and calculates dense nearest neighbor matches quickly. It relies on the coherence between patches of an image for speedup. 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. The algorithm has three main components, illustrated in Figure 2.3.

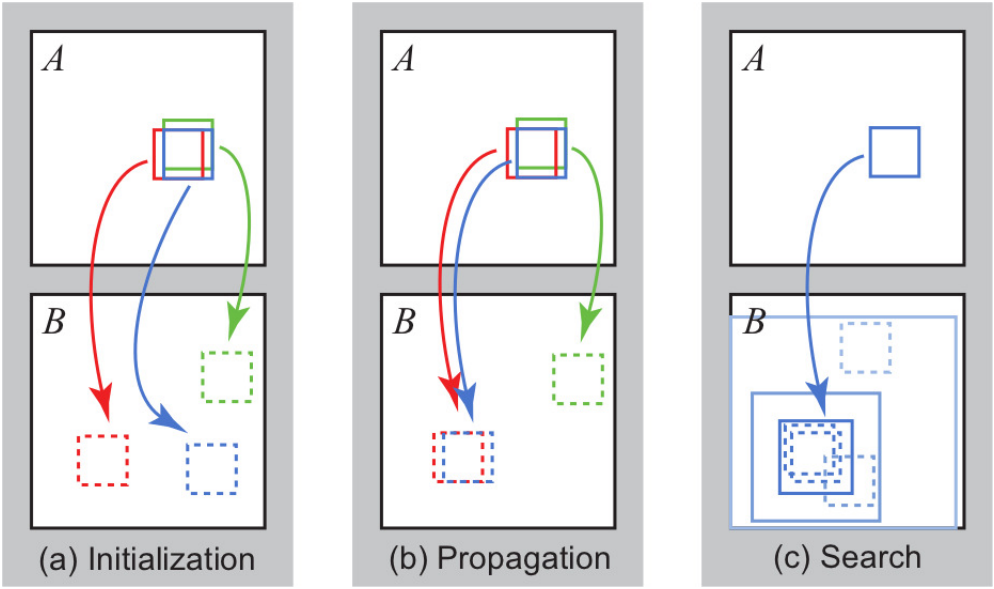



**Figure 2.3** The three stages of the PatchMatch [12] algorithm. (a): Initialization is done randomly for all the patches. (b) A patch searches the matches of its neighbors and propagates good matches. For example, the blue patch in (b) checks above/green and left/red neighbors. (c) the patch searches randomly for improvements in concentric neighborhoods. (Image credit: [12])

1. **Initialization:** The nearest-neighbor field (NNF) is initially filled with either random offsets or some prior information. Applications that use a coarse-to-fine gradual resizing process provides PatchMatch an option to use an initial guess upscaled from the previous level in the pyramid. Often, some early iterations with a random initialization are coupled with guesses upscaled from coarser pyramid levels to achieve a *good* initial guess and escape from local minima. After the initialization, the next two steps are performed iteratively.

2. **Propagation:** In each iteration, matches of a patch at $(x, y)$ are improved using known matches of $(x-1, y)$ and $(x, y-1)$, translated by one pixel to get the corresponding match. In even iterations, the scan-line processing is reversed and matches of $(x+1, y)$ and $(x, y+1)$ are checked for improving the match of the patch at $(x, y)$.

3. **Random Search:** In each iteration, the matches of a patch at $(x, y)$ are further improved by testing a sequence of candidate offsets picked uniformly at random with an exponentially decreasing distance from the current match.

PatchMatch was initially proposed to find single nearest-neighbors of image patches searching only across translations. It was generalized to include $K$ nearest matches and searching for various rotations and scale [13]. PatchMatch is a magnitude of order faster than previous approaches [5, 45, 46] and enables several interactive applications. However, it is not as accurate as one would hope. More iterations are required for higher accuracy, costing more computations. In addition, the main assumption

that patch-match relies on (i.e. coherency of image patches) becomes invalid in some cases, particularly in images with highly textured regions.

Unlike PatchMatch, our Mixed-Resolution Patch-Matching algorithm (MRPM) algorithm focuses on finding optimal matches and maintains coherency by upsampling of matches from coarser resolution levels of the coarse-to-fine framework we employ. On CPUs, we are able to achieve speeds similar to PatchMatch significantly achieving lower RMS error values of the matched patches. Due to the iterative nature of PatchMatch, its GPU implementation is only $7\times$ faster than the CPU implementation [12]. Our simple and direct approach is highly suited for parallel implementation and we are able to achieve linear speed-up with number of cores on the CPU and an additional $70\times$ speed-up on a GeForce GTX 580 graphics card, compared to our CPU implementation run on a quad-core machine.

### 2.2.4 Coherency Sensitive Hashing

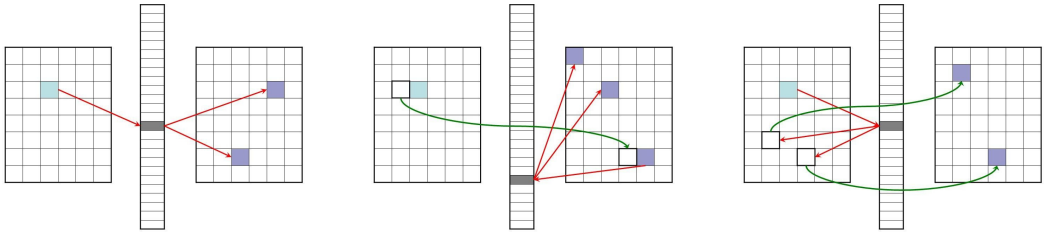
Locality sensitive hashing (LSH) first introduced by Indyk and Motwani [32] employs a hashing scheme to map patches to bins and uses it to find patch correspondences. Given a set of points in a metric space, LSH function families have the property that points that are close to each other have a higher probability of being hashed to the same bin compared to points that are far apart.

Coherency Sensitive Hashing (CSH) [36] motivates from PatchMatch [12] and LSH [24] and replaces the random search step of PatchMatch with a hashing scheme, similar to the one used in LSH. This has two benefits. First, it leads to better initialization of the matches and Second, information can be propagated to nearby patches in the image plane (as also done in PatchMatch) and also to similar patches that were hashed to the same bin. Hence, CSH combines cues of appearance and coherence (of location) in a novel manner which gives a rich set of candidate patches for matching.

Indexing is the first step of the algorithm where CSH builds a set of hash tables. These tables have a property that similar patches (disregarding their image location) are likely to be hashed to the same entry. Let $f()$ be the hash-function and $a_i$ and $b_i$ be patches in image A and image B respectively. Then, for search, instead of only using the patches of image B matched to the same bin as the query patch of image A, CSH makes the following observations to expand its set of potential candidates:

1. If $f(a) = f(b)$ then, $b$ is a good candidate for $a$
2. If $b$ is a candidate for $a_1$ and $f(a_1) = f(a_2)$, then $b$ is a candidate for $a_2$
3. If $b_1$ is a candidate for $a$ and $f(b_1) = f(b_2)$, then $b_2$ is a candidate for $a$
4. If $b$ is a candidate for $Left(a)$, then $Right(b)$ is a candidate for $a$ (also true for $top/bottom$ pairs)

Observations 1-3 are based on appearance and observation 4 is based on coherence of images. This gives three new types of candidate matches. Observation 1 and 3 combined give the first type (Figure 2.4(a)), 3 and 4 combined give the second type (Figure 2.4(b)) and the the third type is derived from observation 2 (Figure 2.4(c)).



(a) Type 1 (b) Type 2 (c) Type 3

**Figure 2.4** 3 different types of candidate matches generated for the search step of CSH. The red arrows are for mapping to the hash-table bin and the green arrows are for depicting the current best match of a patch. (Image credit: [36])

Since CSH performs targeted propagation of information, it achieves lower error values in early iterations compared to PatchMatch. Hence, CSH is a magnitude of order faster than PatchMatch and more accurate, especially in textured regions which do not follow the coherency assumption strictly. Unlike PatchMatch, CSH does not depend entirely on coherency and achieves lower RMS error values of matched patches and improved results on image reconstruction (reconstructing image A given image B and a dense patch-matching from image A to B), which is at the basis of many patch based methods. Hence, CSH outperforms PatchMatch in terms of both, accuracy and speed and is the current state-of-the-art.

## 2.3 Discussion

Our MRPM approach is quite different from the tree-based, randomized and hashing based techniques discussed above. Instead of the hash-table or tree data structure, we use the image pyramid to represent the image at multiple scales and perform exhaustive matching at the coarsest resolution level. Our search is initialized using matches from lower resolution levels as opposed to random sampling or hashing and is restricted to regions upsampled from matches at the previous level for finer resolution levels. Currently, we perform a exhaustive search over this range for finding matches. However, it is also possible to use other techniques of patch-matching for search here. The candidate space is small, but techniques mentioned above such as [5, 45, 68], Patchmatch [12] and CSH [36] can be used in conjunction with our approach for finding the best matches. We extend traditional pyramid approaches by using mixed-resolution vectors for matching. MRPM is not iterative and given its simplicity, fast parallel implementations on multiple cores (linear speed-up w.r.t number of cores) and on GPUs (further $70\times$ speed-up) are possible while it finds near-optimal patch correspondences.

*Chapter 3*

# Pyramid Patch-Matching

The patch-matching problem has been approached in several ways. Most common among these are traditional tree-based approaches. Algorithms based on randomization and hashing have been also been proposed. In contrast to these, motivated by the widespread use of spatial image resolution pyramids, we propose to apply this coarse-to-fine framework for efficiently finding accurate patch correspondences. We call this The *Pyramid Patch-Matching* (PPM) algorithm.

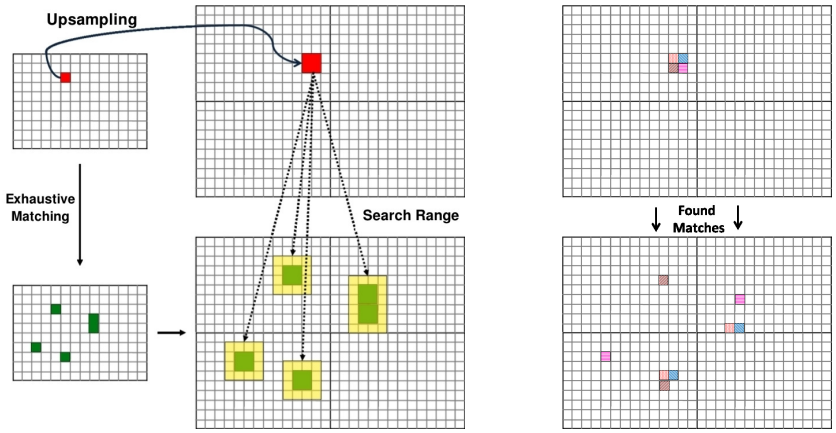



**Figure 3.1** (a): Exhaustive search is done at a reduced resolution level. The search range for the pixels of the up-sampled source region at the next level is an expanded window around the up-sampled target patches. (b): A search in this range gives the best matches

The first step of the algorithm is pyramid construction. Once the pyramid is constructed, PPM computes dense and global patch correspondences at the lowest resolution level using exhaustive search. However, it is also possible to use other patch-matching or nearest-neighbor algorithms like PatchMatch [12], CSH [36] or KD-trees [45] to find the matching at this level. The nearest $K$ matches are stored at each level for each patch. The matches are transferred to the next higher 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 process repeats until matches in the original image are found.

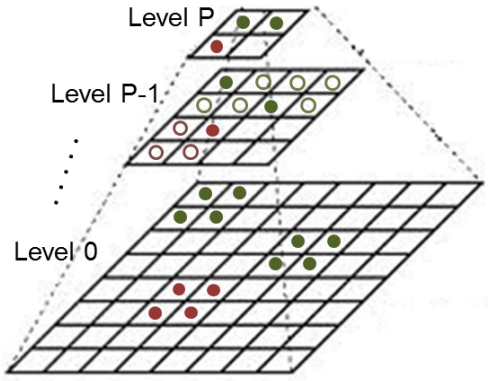



**Figure 3.2** PPM: The pyramid is constructed as shown. At the highest level, exhaustive search gives the matches for every patch. For a *red* (query) patch, matches are shown in *green*. These are then up-sampled and provide candidate matches, shown in same color. A search within this range gives the best matches at the current level (color filled). These are further up-sampled. The process continues until Level 0, the original resolution level.

We also propose some techniques to improve match accuracy and speed of the algorithm. A schematic representation of the algorithm is presented in Figure 3.1.

The upsampling of matches naturally brings in coherency, while maintaining the search targeted. This ensures that neighboring pixels share the space of candidate matches allowing for the coherency model to function. For coherent matching, neighboring patches of the source image are required to match similar neighboring patches in the target image. PatchMatch [12] described in Chapter 2 takes coherency to the extreme and exploits it to gain speed. CSH [36] combines cues of location within the coherency framework of PatchMatch and propagate information to patches that are close in the image plane (coherent) or are similar in appearance. This leads to achieving lower error values in early iterations. However, they miss far-off optimal matches due to their focus on finding coherent matches. PPM initializes the candidate space using matches from a down-sampled image. This keeps the search directed and coherent in the natural sense, without explicitly enforcing coherency.

## 3.1 Pyramid Patch-Matching

The Pyramid Patch-Matching (PPM) begins with the construction of a pyramid. The number of levels in the pyramid are controlled by using a threshold for the smallest image allowed in the pyramid. At this coarsest level, exhaustive global search is performed. Matches found here are up-sampled to provide a search range at finer resolution levels. A search within this range gives the best matches for that level. This process continues until original resolution level is reached. Individual steps are described in detail below.

**Table 3.1** A table showing effects of various down-sampling methods on RMS error of matched patches. Error is the average $L_2$ patch distance between the patch and its matches averaged over all patches in all images of Figure 3.6.

| Method | Error |
|---|---|
| Gaussian Averaging | **60.69** |
| Area Interpolation | 60.75 |
| Cubic Interpolation | 61.12 |
| Nearest-Neighbor Interpolation | 63.53 |

### 3.1.1 Pyramid Construction

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

$$I_p = \downarrow_s (I_{p-1}), \tag{3.1}$$

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. The observations are noted in Table 3.1. It can be seen that except the Nearest-Neighbor interpolation method, all perform well on accuracy of matched patches. Moreover, as will be discussed in Chapter 4, mixing of resolutions reduces the effect caused by down-sampling. 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 a size at least $Pthold$. The 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 $I_{A(p)}$. A sample pyramid is shown in Figure 3.2.

### 3.1.2 Coarsest Level Matching

The processing begins at the coarsest resolution level of the pyramid: level $P$. Here, images $I_{A(P)}$ and $I_{B(P)}$ are small. The small size of images at this highest level in the pyramid makes it feasible to perform a global search for similar patches over them quickly. Each patch in $I_B(P)$ is a candidate match for every patch in $I_A(P)$. An exhaustive search is performed to find the best matches. However, it is also possible to use other techniques, such as CSH [36], PatchMatch [12] and other approximate nearest neighbor methods to find the matching at this level, based on the application. A match is stored as a structure containing $(row, col)$ coordinates and a distance with the source patch ($similarity \propto \frac{1}{distance}$). Low distance and high similarity patches are good matches. The top $K$ such matches are stored for every patch at the end of processing of this level. For this highest level $P$ of the pyramid, all patches in the target image are candidates. Hence for every patch $a$ in $I_A$,

$$candidateList_{I_{A(P)}}(a) \leftarrow \text{all patches in } I_{B(P)} \tag{3.2}$$

and the top K matches are given by

$$matchList_{I_{A(P)}}(a) = \underset{b_i}{\text{argmin}}_K \; D(a, b_i), \quad b_i \in candidateList_{I_{A(P)}}(a), \tag{3.3}$$

This gives the Approximate Nearest Neighbor Field (ANNF) of the source image at the coarsest level to the target image at the coarsest level. This field then acts as an indicator to localize search for matches of patches at finer resolutions.

### 3.1.3 Finer Resolution Levels Matching

While processing a level $p$ of the pyramid, matches of level $p+1$ are already known. Matches from level $p+1$ are up-sampled and the candidate list for patch $a$ at level $p$ is given by

$$candidateList_{I_{A(p)}}(a) = \bigcup N(\uparrow_s (matchList_{I_{A(p+1)}}(a^*))), \tag{3.4}$$

where $a^*$ is the patch in $I_{A(p+1)}$ corresponding to $a$, $\uparrow_s$ represents up-sampling, and $N()$ represents a neighborhood operator. This is outlined in Algorithm 1. Matches from level $p$ for patch $a$ are given by

$$matchList_{I_{A(p)}}(a) = \underset{b_i}{\text{argmin}}_K \; D(a, b_i), \quad b_i \in candidateList_{I_{A(p)}}(a), \tag{3.5}$$

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.

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 $p$.

**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**

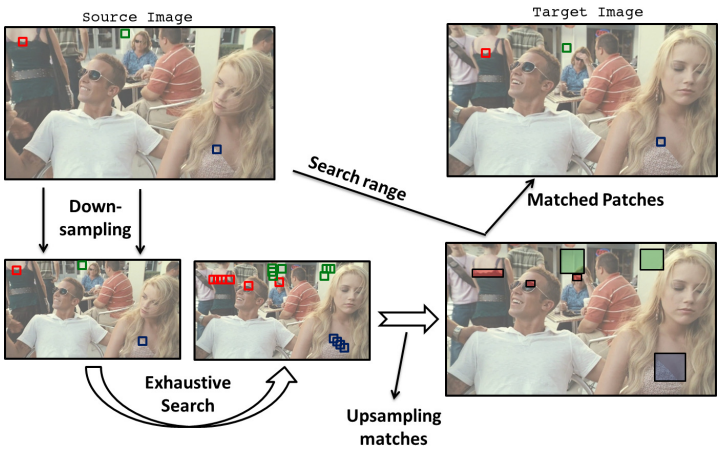



**Figure 3.3** Original images are downsampled and exhaustive search for matches is performed. More number of matches than required are located at coarser resolutions (We call this heuristic "Increase K"). The matches found here are transfered to the original resolution level by upsampling. Union of 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.

4. The union of $u \times u$ windows around each of these transferred matches form the search range for matching patch $a$.

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 down-sampled 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 from the CSH dataset (Figure 3.6), an example run of the algorithm is shown describing a simple two level pyramid matching process in Figure 3.3. 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 down-sampling 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 vectors $V(a)$ and $V(b_i)$
  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** $matchList_{(P)}$

## 3.2 Improvements to PPM

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

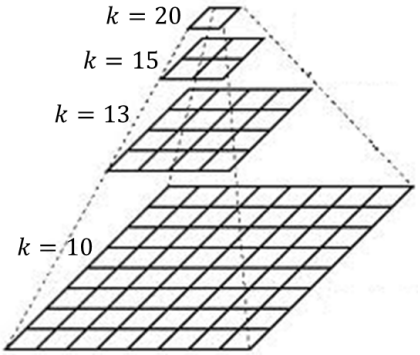



**Figure 3.4** Increase K: More number of matches than required are stored at coarser resolution levels of the pyramid. This number is gradually reduced to the desired number of matches as the algorithm moves towards original resolution level.

### 3.2.1 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 required number of matches as the algorithm proceeds. 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. This is illustrated in Figure 3.4.

This idea can be seen at work in Figure 3.3. Notice that only 1 patch is originally required, but at the coarser resolution, more number of patches are stored, providing a wider search range for better

**Table 3.2** Reduction in error values and slight increase in time before and after applying the *Increase K* condition. Values shows below are for matching image pairs. Parameters fixed were Search Range($u \times u$)=$5 \times 5$ and $Pthold = 32$ px. Error is the average RGB distance between patches averaged over all patches.

| **Image Resolution** | **KNN** | **Patch-Size** | **Before Increase K** | | **After Increase K** | |
|---|---|---|---|---|---|---|
| | | | Time | Error | Time | Error |
| $256 \times 256$ | 5 | $7 \times 7$ | 3 sec | 69.23 | 4.5 sec | 67.08 |
| $256 \times 256$ | 10 | $5 \times 5$ | 4 sec | 54.78 | 4.75 | 53.12 |
| $512 \times 512$ | 2 | $7 \times 7$ | 5 sec | 47.64 | 6.25 sec | 46.72 |
| $512 \times 512$ | 10 | $5 \times 5$ | 16 sec | 45.81 | 23 sec | 44.73 |
| $1024 \times 1024$ | 2 | $9 \times 9$ | 23 sec | 76.33 | 25 sec | 74.32 |

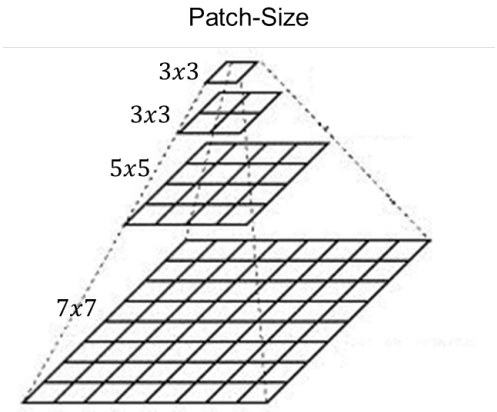



**Figure 3.5** Reduce Patch-Size: At coarser resolution levels of the pyramid, the size of patches to be matched is reduced. At a down-sampled level, a smaller patch would capture and match information from desired patch-size at a finer resolution level.

accuracy. The effect of Increase-K on time and accuracy can be seen in Table 3.2. We observe that with a marginal increase in time, we are able to increase the accuracy of matched patches. The table shows the values of time taken and error measures for various image and patch sizes. This technique is most helpful when less number of matches are originally required. In such a case, keeping more number of matches at coarser resolution levels preserves the best match within the extended search range.

### 3.2.2 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. Hence, reduced size patches are matched at higher levels of the pyramid. This is illustrated in Figure 3.5. The reduced patch-size makes matching more efficient.

Intuitively, the smallest patch would be $3 \times 3$. However, we notice that, at the coarsest resolution, even using pixel values ($1 \times 1$ patch) improves accuracy in some cases. This behavior is justified due to similar neighborhoods of the original image being mapped to a similar color after down-sampling. Patches which are similar in finer resolutions are generally down-sampled to similar pixel values in the coarser resolutions. A single pixel here, corresponds to a bigger patch in the finer resolution level. However, this behavior was not seen across all cases and we use $3 \times 3$ patches as the minimum patch-size. Reduce-patch-size used with the increase-K improvement provides efficient and accurate matches.

### 3.2.3 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

**Table 3.3** Timings before and after applying the Early Termination (ET) condition. Values shown below are for matching Image pairs. Parameters fixed were Search Range($u \times u$)=$5 \times 5$ and $Pthold = 32$ px

| Image Resolution | KNN | Patch-Size | Before ET | After ET |
|---|---|---|---|---|
| $256 \times 256$ | 5 | $7 \times 7$ | 5 sec | 3 sec |
| $256 \times 256$ | 10 | $5 \times 5$ | 6 sec | 4 sec |
| $512 \times 512$ | 2 | $7 \times 7$ | 10 sec | 5 sec |
| $512 \times 512$ | 10 | $5 \times 5$ | 25 sec | 16 sec |
| $1024 \times 1024$ | 1 | $9 \times 9$ | 32 sec | 19 sec |

speeds up the matching process by eliminating dissimilar patches quickly. This technique has been used by other approaches like those of Li et al. [39] and Daniel et al. [10]. Quantitative benefits for using the Early Termination approach are found in Table 3.3. By avoiding these redundant calculations we are able to achieve much better times for the same accuracy. The table lists the time taken before and after using the "early termination" approach on various image and patch sizes with different number of required number of matches.

## 3.3 Parallel Implementation

Our approach works in the pixel space, which is inherently parallel. Matching of each patch is done independently. We take advantage of this by using the multiple cores of the CPU (with OpenMP [50]) and the architecture of a graphics processing unit (GPU) [48]. The multi-core implementation achieves near-linear time speed-up with number of cores. The GPU code provides an additional $70\times$ speed-up compared to the multi-core implementation run on a quad-core machine.

### 3.3.1 Multi-core Implementation

The first step of our algorithm, pyramid construction is a problem suited for parallel implementation. Using this fact, we down-sample images by processing groups of pixels in parallel. Each thread handles a block of pixels, averages them according to the down-sampling function used and assigns the new value to the pixel in the next higher level of the pyramid. After pyramid construction, for the matching problem, we have query patches and candidate search patches. At the coarsest resolution level exhaustive matching is done and the candidate space is same for all source patches. At finer resolution levels however, the candidate patches are different for different query patches. In either case, each query patch searches the space of its candidate patches independently. This enables us to find matches of every patch in parallel.

OpenMP [50] is a multi-platform shared-memory parallel programming API which provides directives for control over how the compiler handles parallel processing in a program. We use a directive ("# pragma omp parallel for") that define a parallel region to be run by multiple threads in parallel. It provides a work-sharing construct identifying the iterative for-loop whose iterations should be run in

parallel. While processing each level of the pyramid, we run a for-loop over all query patches which searches for their matches in parallel. It reads from a shared array containing candidate patch lists, calculates distances and writes the best matches to the output array. Then, blocks of this array are assigned to threads. A thread up-samples the matches to generate candidate lists for the finer resolution level searches. At the original resolution level, this output array is returned and contains the final patch correspondences found by the algorithm. Our algorithm's design is motivated by parallel constructs and is not efficient when run in a single-core configuration. We use the multi-core CPU implementation presented above for all our experiments performed in this thesis.

### 3.3.2 GPU Implementation of PPM

General Purpose computing on GPUs (GP-GPU) [51] has become common. The graphics hardware is being exploited to efficiently perform several computation intensive tasks. We describe how various stages of our Pyramid Patch-Matching (PPM) approach make use of this in our GPU implementation of PPM.

#### 3.3.2.1 Forming the Pyramid

Pyramid construction is a standard problem for high-end parallel and GPU processors. We use the standard in built down-sampling function of the GPU - Computer Vision Library [2]. To understand this better, let us say we have an image $I_o$ to be downsampled with a scale factor $s$. An image $I_d$ is created with dimensions $I_o(width)/s$ and $I_o(height)/s$. Every pixel of $I_d$ is processed by a thread which maps the pixel to it's corresponding pixels in $I_o$. These pixels are then averaged according to the interpolation technique used. Such high parallelism generates the pyramid for the image in microseconds. Only once, the original image is copied from Host (CPU) to Device (GPU). The pyramid then formed is sent for assisting in patch-matching to other kernels.

#### 3.3.2.2 Coarsest Level Matching

At the coarsest level, each overlapping patch in the target as well as source image is represented as a 1D vector containing information of all patches in the image. Each thread is responsible for one vector

**Table 3.4** GPU and CPU timings of PPM. Values for $K = 5$, Patch = $7 \times 7$ and $Pthold = 32$ px, except in global exhaustive search

| Image(s) | Search | CPU | GPU | Speed-up |
|---|---|---|---|---|
| Dataset Images (Fig. 3.6) | $3 \times 3$ | 13.46 sec | 278 msec | 48× |
| | $5 \times 5$ | 20.45 sec | 684 msec | 30× |
| | $7 \times 7$ | 29.68 sec | 1113 msec | 26× |
| Lena ($256 \times 256$) | Brute-Force | 356.63 sec | 4.89 sec | 72× |

distance calculation. At this level, distances of all patches in image $A$ are calculated with all patches in image $B$. These distances are stored along with their locations. This array is sorted in parallel (using the Thrust library [30]). The top elements with the lowest distance are picked and assigned as matches to that patch.

#### 3.3.2.3 Finer Levels Matching

At these levels, search for matches for every patch is fixed in regions determined by matches of the patch in the down-sampled image. This region is formed by taking a union of search ranges around matches transferred from the previous level. Unlike the coarsest level here, for matching every patch, first a vector representation of all the candidate patches is made and the source patch is represented as a vector. Now distance calculation is performed in parallel. The distance values are sorted and $K$ top matches are picked. These are then used to form the search range for the next level, until the original resolution level is reached. At the original resolution level, the final matching is done similarly which gives the desired output of $K$ top match locations in image $B$ for every patch (pixel) in image $A$.

For fast access, we use the shared memory to store the per-pixel source patch. Table 3.4 shows the performance of our GPU code compared against the multi-core implementation. 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.

## 3.4 Effect of 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

**Table 3.5** 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. 3.6.

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

**Table 3.6** 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 $32 \times 32$ 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 |

between time and accuracy also. Effects of varying the number of required matches $K$ and the patch-size can be seen in Table 3.5.

### 3.4.1 Pyramid Threshold and Pyramid Levels

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 3.6 shows the running time and average error using different values of $Pthold$ for 10-NN using $5 \times 5$ patches. Error values increase as we allow smaller images in the pyramid. Search range used is $3 \times 3$ around upsampled matches. It can be seen that stopping at a resolution near $32 \times 32$ gives best performance and accuracy.

### 3.4.2 Search Range

The search range parameter controls the area around up-sampled matches that is included in the candidate list for searching. A wider search range provides more search area and improves accuracy. The effect of this parameter on time and accuracy can be seen in Table 3.7. It can be seen that $3 \times 3$ and $5 \times 5$ are optimal in terms of accuracy as well as speed. Having a really wide search range marginally improves accuracy but costs significant computation time.

**Table 3.7** A table showing the effects of the search range parameter $u \times u$ on error and time. Error is the RMS distance between matches patches averaged over all patches of images. Larger search ranges are expensive to compute and do not provide much gain in accuracy.

| Search Range | Error | Time |
|---|---|---|
| $3 \times 3$ | 109.47 | 13 sec |
| $5 \times 5$ | 99.92 | 22 sec |
| $7 \times 7$ | 96.81 | 30 sec |
| $9 \times 9$ | 94.8 | 41 sec |

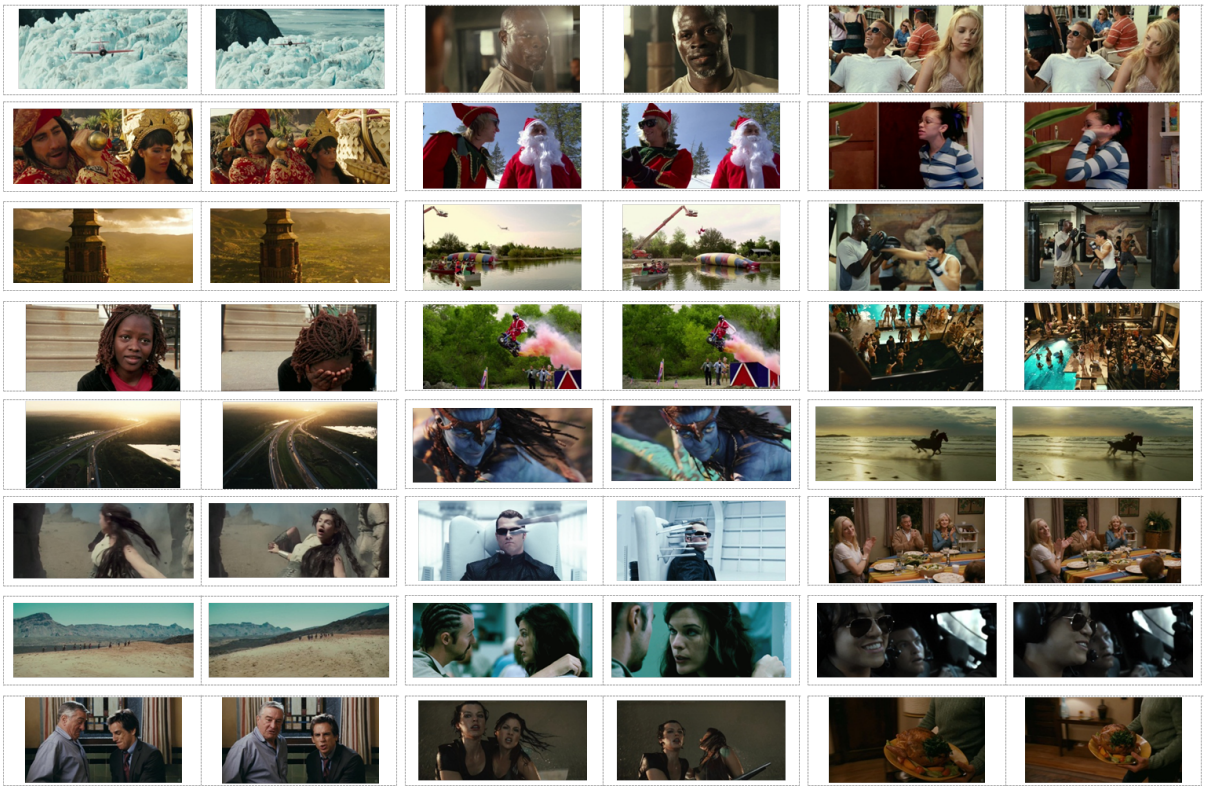


**Figure 3.6** CSH Dataset: Showing 24 of 133 image-pairs from the dataset provided by CSH [36]. Each pair consists of images of the same scene of a movie with usually some motion of both camera and subjects in the scene. (The images are between 1 and 30 frames apart in the video)

## 3.5 Experiments

In this chapter, we described our Pyramid Patch-Matching (PPM) algorithm. Here, we show the potential of this approach. Various methods for patch-matching were reviewed in Chapter 2. The PatchMatch algorithm by Barnes et al. discussed there compares to several techniques and performs matching an order of magnitude faster than previous approaches [12]. However, it is not as accurate. It was extended by Korman and Avidan who propose Coherency Sensitive Hashing (CSH) [36], motivated by Locality Sensitive Hashing (LSH) [24]. CSH compares against PatchMatch and is slightly more efficient while being significantly more accurate than PatchMatch [36]. Hence, CSH is the current state-of-the-art algorithm for finding similar patches in images.

We compare our PPM approach with CSH. We perform exhaustive matching on GPU and calculate the ground truth 10 nearest neighbors in the target image of each patch for image pairs in the dataset. CSH restricts the patch-dimensions to be powers of 2. Hence, in order to achieve a fair comparison, we slightly modify our algorithm and calculate $8 \times 8$ patches. We show that PPM finds more accurate matches while its performance matches with CSH. Our approach is highly efficient on parallel architectures compared to CSH which is an iterative approach. We use the dataset provided by Korman and

**Table 3.8** Percentage of ground truth matches captured by PPM and CSH. PPM captures up to 4% more matches when the top 10 matches are required.

| Best K Matches | CSH | PPM |
|---|---|---|
| Best Match | 92.43% | 92.48% |
| Best 5 Matches | 88.06% | 89.7% |
| Best 10 Matches | 81.72% | 85.69% |

Avidan for CSH available on their web-page [35]. It contains 133 image pairs resized to 0.4MP some of which are shown in Figure 3.6. The CPU code is implemented using OpenMP [50] and the GPU code is implemented in CUDA [22, 30]. All experiments were performed on an Intel i7 920 quad-core processor running at 2.67GHz with 3 GB of RAM. The GPU used was a Nvidia GeForce GTX 580 card.

### 3.5.1 Proximity to Ground Truth

In this experiment, we compute how many of the ground truth matches are found by PPM and CSH. Code for CSH was obtained from their web-page [35] and default parameters (5 iterations) were set. For PPM, a $3 \times 3$ search range with a 32-pixel pyramid threshold was used. Table 3.8 records the observations. When only the best-match is required, PPM and CSH both perform equally. The benefits of PPM present strongly when more number of matches are calculated. PPM captures up to 4% more ground truth matches than CSH when top-10 matches are required.

### 3.5.2 Error

In this experiment, we set the parameters of the algorithms such that they achieve similar computation times. We then compare the error values they achieve. Figure 3.7 shows the average error of PPM, CSH compared with ground truth for 10 images picked uniformly at random from the CSH dataset. We achieve lower error values than CSH and are closer to the ground truth. CSH focuses on finding coherent matches and misses far-off optimal matches increasing their error. We use their default setting

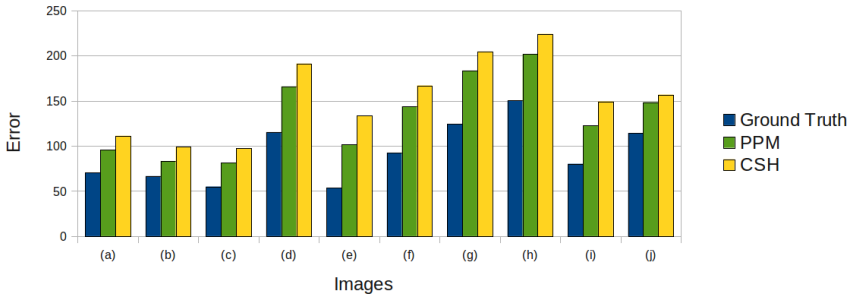



**Figure 3.7** PPM v/s CSH: Comparing error values achieved by PPM and CSH for the top-10 matches found for each image pair of the CSH image dataset. Parameters were set such that computation time for both algorithms was similar.

of 5 iterations for calculating their matches. Our parameters at $Pthold = 32$ pixels and search range $3 \times 3$ achieve similar computation times as CSH for this problem.

## 3.6 Conclusion

In this chapter, we showed that pyramids can be used for the problem of patch-matching by our Pyramid Patch-Matching (PPM) algorithm. We find matches at high speeds outperforming the accuracy of state-of-the-art algorithm Coherency Sensitive Hashing (CSH). Our pyramid representation is a simple structure and is traversed quickly, allowing for fast parallel implementations. Several techniques such as, increase-K, reduce-patch-size and early termination were proposed to improve the PPM process were proposed. However, due to down-sampling, lot of information is lost and matching at coarse resolutions is not confident. We propose mixed-resolution vectors described in the next chapter to further improve the accuracy of PPM.

*Chapter 4*

# Mixed-Resolution Patch-Matching

## 4.1 Motivation

In the previous chapter, we propose Pyramid Patch-Matching (PPM) which uses a spatial image resolution pyramid to represent images at multiple scales. It performs a coarse-to-fine search over this pyramid and was shown to find matches with state-of-the-art accuracy maintaining speed. However, higher levels of the pyramid contain smooth or approximate versions of the original image and matching at these coarse resolutions is not confident.

The matching in PPM is performed independently using patches at each level of the pyramid. Down-sampling reduces the amount of information in the higher levels of the pyramid which leads to smooth distance functions and poor localization of matches. In order to improve the accuracy of matches at this reduced resolution level, we observe that certain information from finer levels can be included in patch vectors of coarser levels. This would match more relevant information improving matching accuracy. We extending usual vectors that contain patch information from a single level and propose Mixed-Resolution (MR) vectors that mix information from various resolution levels to form patch vectors.

## 4.2 Mixed-Resolution Vectors

Our PPM approach resolves to brute-force search when the number of pyramid levels is only 1, i.e. the original resolution image. Matching patches at this resolution would achieve the best accuracy but has impractical computational costs. PPM constructs the pyramid to achieve localization of matches, which provides it speed. This localization comes through search windows which are formed by up-sampling the matches found at coarse resolutions. Hence, the accuracy of matching at coarse resolutions affects the overall accuracy of PPM. As the number of levels in the pyramid increase, the matching becomes faster but less accurate. PPM represents patches as vectors and solves the $K$-NN problem over this space at each level. However, these vectors contain information from only a single resolution level. At fine resolutions, these match correctly. At coarse resolutions, matching is not as accurate. The coarse

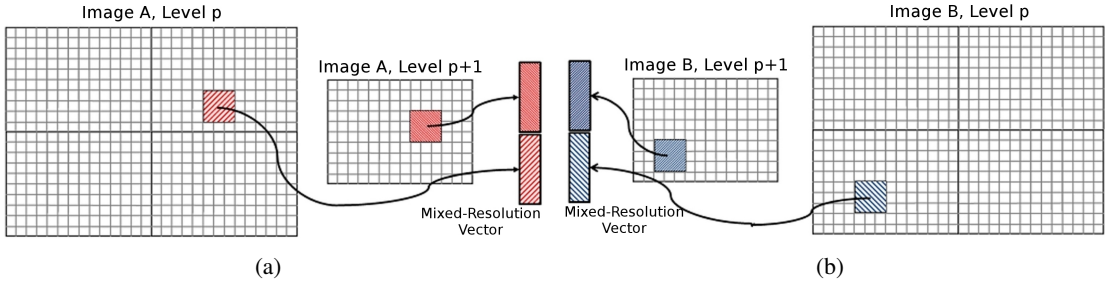



**Figure 4.1** Forming Mixed Resolution (MR) vectors. Information from finer resolution levels is included in the patch-vectors of coarse resolutions.

resolution patch vectors need to contain more and relevant information to improve match scores. We mix different resolution levels to provide this information and propose *Mixed-Resolution Vectors*.

Mixing of resolutions refers to combining information from different resolution levels to form patch vectors. For including relevant information, we must map patches at coarser resolutions to those at finer levels. Simple up-sampling provides this mapping. PPM processes each resolution level starting from the coarsest. Including all the information from the mapped patch at the finer level would be redundant because the finer levels would anyway be processed in future stages of the algorithm. In such a case, excluding the coarsest resolution level would have the same effect. Hence, we wish to include only certain information from the finer resolution.

There are several ways in which this information can be picked. One could use some prior knowledge to decide upon the pixels whose information will be included. Some fixed coordinates of pixels within the patch can also be used and their information be included. The benefits of using different ways depend upon the specific properties of an image and an overall best way is not defined. In our experiments, we use a simple approach to do this. The information at the center of the corresponding patch in the finer resolution is picked to be included in the patch-vector of the coarse resolution. We keep the size of this center to be equal to the patch size at the current level. This is shown in Figure 4.1

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. The finer level patches corresponding to the coarsest level patch are much larger. Figure 4.2 shows for a coarsest level patch, the finer level patches corresponding

**Algorithm 3** 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** $matchList_{(P)}$

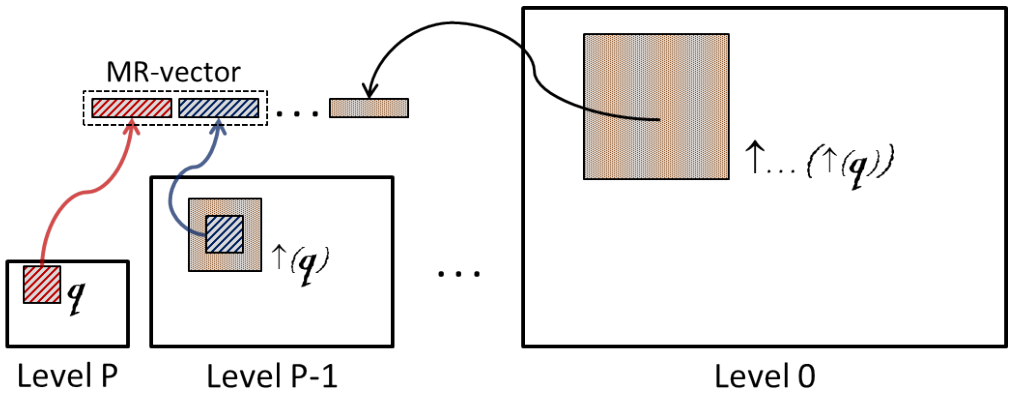



**Figure 4.2** A patch $q$ at the coarsest resolution level maps to larger patches at finer levels, as shown. A part of the upsampled patch $\uparrow (q)$ is included in the patch-vector of this coarse patch to form the MR-vector. This can be upsampled further to include information from more finer resolutions.

to it from which only some information is included in its vector. These MR-vectors contain more and relevant information about the patch. This reduces the effect of down-sampling and makes matching more accurate. Mixing resolutions improves match localization and makes PPM more accurate. We follow the framework of PPM and change only patch-vectors replacing them with MR-vectors. This changes the search step of the algorithm where vectors are formed. This is illustrated in Algorithm 3. We call this The *Mixed-Resolution Patch-Matching* Algorithm.

## 4.3 Effects of Mixed-Resolution and Search Range

Mixed-Resolution Patch-Matching (MRPM) improves the accuracy of matches with a marginal cost in time. Increasing the search range around up-sampled matches also has the same effect. We experiment with various search ranges and mix different number of resolution levels to compare their effects on match accuracy and time. The observations are recorded in Table 4.1. As expected, increasing the

**Table 4.1** 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. 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) | $l = 2$ | $l = 3$ |
| $3 \times 3$ | 109.47 (13 sec) | 96.71 (19 sec) | 93.93 (24 sec) |
| $5 \times 5$ | 99.92 (22 sec) | 90.38 (30 sec) | 87.74 (35 sec) |
| $7 \times 7$ | 96.81 (30 sec) | 88.62 (45 sec) | 86.18 (53 sec) |
| $9 \times 9$ | 94.8 (41 sec) | 87.36 (63 sec) | 85.03 (73 sec) |

**Table 4.2** A table showing speed up of MRPM on the GPU. Values for $K = 5$, Patch = $7 \times 7$, $Pthold = 32$ px and $l = 2$.

| Image(s) | Search | CPU | GPU | Speed-up |
|---|---|---|---|---|
| Dataset Images (Fig. 3.6) | $3 \times 3$ | 19 sec | 442 msec | 43× |
| | $5 \times 5$ | 30 sec | 937 msec | 32× |
| | $7 \times 7$ | 45 sec | 1.73 sec | 26× |

search range reduces the error with extra cost in time. Mixing resolutions achieves even lower error values with less cost in time. The extended search range may or may not contain relevant matches. It only increases the candidate space for search. Finer resolutions surely contain more significant information for matching than areas around upsampled matches. Hence, including information from these in patch-vectors guarantees an improvement in the matching score making MR-matching more effective than matching with an increased search range. We observe that 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 $5 \times 5$ gives the best of time and accuracy.

## 4.4 GPU Implementation of MRPM

Similar to Pyramid Patch-Matching (PPM), Mixed-Resolution Patch-Matching (MRPM) also works in the pixel space and processes patches independently allowing for fast GPU implementation. In Chapter 3, the GPU implementation of PPM was described in detail. Most of the process for MRPM remains similar except for the patch vectors that are formed by mixing resolution levels instead of a single resolution level.

As the pyramid is formed on the GPU, data of each resolution is already in the GPU memory. While traversing the levels of the pyramid and performing matching at each level, the source and target patches pick relevant information from current and finer resolution levels to form 1D representative vectors of the patch. For each source patch, number of threads equal to the number of candidate patches are invoked. Each thread calculates the distance between MR-vectors and stores it in an array along with the coordinates of the candidate match. This array is then sorted in parallel using key-value pairs. The top $K$ matches are picked for transfer to finer resolution level by up-sampling. Patches in a window around these up-sampled matches are candidate matches for the up-sampled query patch. This process repeats until original resolution level is reached. The sorted output array containing match scores and locations is returned.

Table 4.2 shows the performance gain by comparing the CPU and GPU implementation of MRPM. Algorithms were run on an Intel i7 920 quad-core processor with 3GB RAM. The GPU used is an NVIDIA GTX 580 card with 512 cores.

**Table 4.3** Table showing the effects of varying KNN and Patch-size. Values are for search range $3 \times 3$ and $Pthold = 32$px with $l = 2$ for MRPM.

| KNN | Patch-size | PPM | | | MRPM | | |
|---|---|---|---|---|---|---|---|
| | | Error | CPU | GPU | Error | CPU | GPU |
| 1 | $5 \times 5$ | 82.1 | 9.5 sec | 127 msec | 68.63 | 11.46 sec | 174 msec |
| 1 | $7 \times 7$ | 130.9 | 9.81 sec | 142 msec | 109.06 | 12.08 sec | 225 msec |
| 1 | $9 \times 9$ | 176.76 | 10.21 sec | 155 msec | 147.30 | 12.9 sec | 294 msec |
| 5 | $5 \times 5$ | 65.94 | 10.85 sec | 225 msec | 58.72 | 14.58 sec | 353 msec |
| 5 | $7 \times 7$ | 109.467 | 13.42 sec | 304 msec | 96.71 | 18.73 sec | 528 msec |
| 5 | $9 \times 9$ | 157.327 | 17.7 sec | 402 msec | 138.76 | 25.04 sec | 731 msec |
| 10 | $5 \times 5$ | 63.16 | 13.66 sec | 392 msec | 57.25 | 19.48 sec | 680 msec |
| 10 | $7 \times 7$ | 105.54 | 20.21 sec | 569 msec | 94.91 | 29.3 sec | 921 msec |
| 10 | $9 \times 9$ | 152.34 | 29.31 sec | 793 msec | 137.04 | 42.12 sec | 1007 msec |

## 4.5 Experiments

In Chapter 3, PPM was shown to outperform the state-of-the-art CSH algorithm in terms of accuracy. In this chapter, we introduced Mixed-Resolution (MR) vectors and described the Mixed-Resolution Patch-Matching (MRPM) algorithm. Here, we perform experiments to show benefits of using MR-vectors compared to traditional patch-vectors. We also compare MRPM with PPM and CSH [36]. We use the 133 image-pairs dataset provided by Korman and Avidan [36, 35] for our experiments. Some of the 133 image-pairs are shown in Figure 3.6. Our implementation of MRPM is in C++ using OpenMP. Code for CSH was obtained from CSH-web-page [35]. The computation platform was an Intel i7 920 processor running at 2.67GHz with 3 GB of RAM. The GPU used was an Nvidia GeForce GTX 580 card with 512 cores. In these experiments, we mix pixels from two resolution levels in the matching windows. The *Avg. Error* used to quantify matching quality is the average $L_2$ distance between a patch and its matches in RGB space, averaged over all patches and all images.

### 4.5.1 PPM v/s MRPM

In this experiment, we vary $K$ and patch-size and record observations for time and accuracy using traditional (PPM) and MR-vectors (MRPM). Average values for images of the CSH dataset are shown in Table 4.3. The table also shows time comparisons on the CPU and GPU. With a marginal increase in time, MRPM is able to achieve lower error values. We request the reader to note that the error values are not averaged over patch-size and depend upon it with the error being higher for larger patches.

### 4.5.2 Comparison with PPM and CSH

In these experiments, we compare the proximity of matches found by PPM, MRPM and CSH with the ground truth matches calculated using exhaustive search. We show that MRPM finds up to 4% and 8% more ground truth matches than PPM and CSH respectively. In another experiment, we plot the

**Table 4.4** 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 [36] | PPM | MRPM |
|---|---|---|---|
| Best Match | 92.43% | 92.48% | 93.57% |
| Best 5 Matches | 88.06% | 89.7% | 91.51% |
| Best 10 Matches | 81.72% | 85.69% | 89.87% |

error values for 10 images from the CSH dataset picked uniformly at random and show that MRPM significantly outperforms PPM and CSH and achieves much lower error values.

#### 4.5.2.1 Proximity to ground-truth

We compute how many of the ground truth matches are captured by PPM, MRPM and CSH. Table 4.4 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, PPM approach and the CSH approach. The best match is mostly captured by all the algorithms and MRPM holds only a slight edge over them. However, MRPM finds around 4% and 8% more ground-truth matches among the top-5 and top-10 matches than PPM and CSH respectively.

#### 4.5.2.2 Error

In this experiment, we set parameters of algorithms such that they achieve similar computation times. We then plot the error values achieved by them. Figure 4.3 compares the average error of MRPM with CSH, PPM and the ground truth. MRPM achieves error rates much closer to the ground truth error significantly outperforming CSH and PPM. Matching of simple patch-vectors in PPM leads to poor localization of matches which gives bad accuracy. CSH focuses on finding coherent matches and misses far-off optimal matches increasing their error.

We also perform an experiment by fixing an error value and monitoring the time taken by MRPM and CSH 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. The type of matches required and choice of algorithm depends highly upon the application.

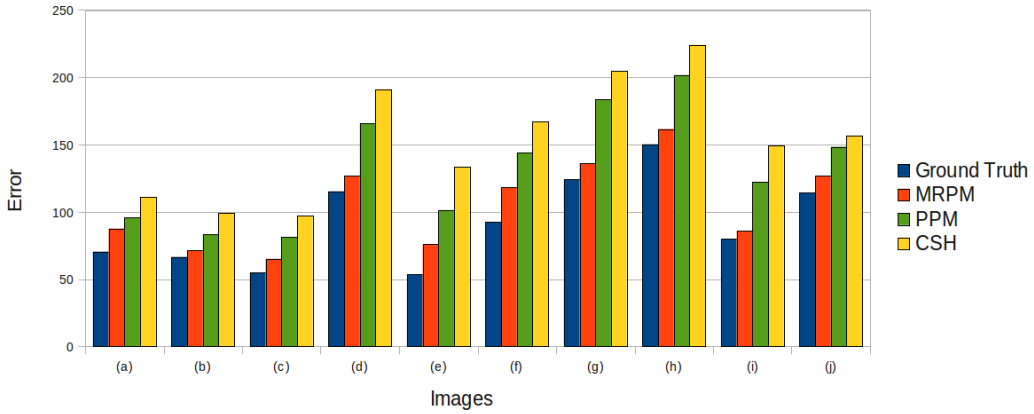



**Figure 4.3** Average error values of MRPM compared with CSH, PPM and the ground truth. MRPM consistently achieves lower error values than the other approaches and is much closer to the ground truth.

## 4.6 Conclusion

In this chapter we introduced Mixed-Resolution Vectors and matched patches using these in the framework of Pyramid Patch-Matching. We show that by mixing information from finer resolution levels, better localization of matches is achieved which improves matching accuracy. A GPU version which follows from the GPU implementation of Pyramid Patch-Matching (PPM) was presented which provides a gain of up to 50$\times$ over a quad-core CPU implementation. We show our Mixed Resolution Patch-Matching (MRPM) outperforms PPM and CSH in terms of match accuracy with a marginal cost of time by matching MR-vectors. The concept of MR-vectors is generic and may also be applied to several other image processing and computer vision tasks. We hypothesize that several problems can achieve better accuracy with the use of these MR-vectors.

*Chapter 5*

# Applications

## 5.1 Vision Applications

Finding matching patches between image regions is a core issue in many computer vision problems, from classical problems like optical flow, to low-level image processing such as non-local means denoising. Synthesis tasks such as texture synthesis and image in-painting to high level image analysis tasks like object detection, image segmentation and classification also require match correspondences between images and image regions. Correspondence searches can be classified as either *local*, where a search is performed in a limited spatial window, or *global*, where all possible displacements are considered. These correspondences can also be classified as *sparse*, determined only at a subset of key feature points, or *dense*, determined at every pixel or on a dense grid in the input.

Computing dense and global matches are computationally expensive and several approaches use local sparse correspondences for efficiency. Multi-resolution refinement is often used with local correspondence search because it captures only small displacements (e.g. in optical flow [9]). Sparse keypoint correspondences [40, 44] find application in alignment, 3D reconstruction and recognition tasks. These and other sparse to dense iterative methods [16, 57] perform well in textured high-resolution images but are not so accurate in other cases. These can benefit from relaxing the locality and sparseness assumptions. Many other analysis [8] and synthesis [21] applications inherently require a dense global matching for good performance. Some applications require matches to be found at different scales. For example, super-resolution from a single image [27] represents the image at multiple scales and searches throughout for matches.

In previous chapters, we proposed such an algorithm which can find dense and global matches more accurately than previous randomized and hashing based techniques. Our algorithm, Mixed-Resolution Patch-Matching (MRPM) can be used at the core of these applications to provide speed and improved quality of results. We show the benefits of using MRPM for four applications. First, for the problem of image denoising, we refer to the solution proposed by Buades et al. [18] termed *non-local means* and use MRPM to globally find similar patches that are used to denoise. Second, for the task of image summarization (or retargeting), we improve efficiency by using MRPM in the framework of the solution

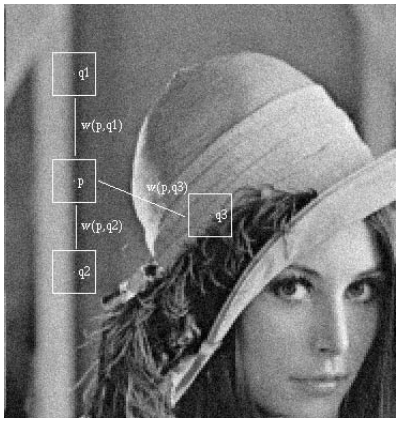



**Figure 5.1** The NL-means strategy: To denoise the pixel at the center of a patch $p$, similar patches would contribute with a higher weight than dissimilar patches. $w(p, q1)$ and $w(p, q2)$ are both much greater than $w(p, q3)$.

proposed by Simakov et al. [55]. We also show how mixed-resolution vectors improve quality over traditional patch-vectors. Third, for auto-crop, which is the problem of detecting a small region (fixed-size) of an image containing the most information, a patch-matching based solution empowered by MRPM is proposed. Finally, we measure performance of MRPM on the problem of image reconstruction which aims at reconstructing image A, given image B and a dense patch-matching from A to B.

### 5.1.1 Non Local Means Denoising

Denoising refers to the recovery of a digital data corrupted with noise. Many algorithms have been proposed to solve this problem and a review of these can be found in [19]. Denoising methods also degrade or remove the fine details and textures in an image. In order to better understand this removal, Buades et al. [18] introduced *method noise*, defined as the difference between the noisy image and its denoised version. For a good denoising method, the method noise would contain little or no structure, only random noise.

Recently, Buades et al. [18] showed that high-quality results could be obtained by non-local means (NLM) denoising: finding similar patches within an image and then averaging these. The weights depend upon the patch-similarity around the pixels as shown in Figure 5.1. Subsequent works [23, 43] motivated by NLM, showed that this patch-based method could be extended to obtain state-of-the-art results by performing additional filtering steps. Buades et al. [18] searched for similar patches only within a limited search window to save computations. Because every pixel in the search window contributes to denoising and a dissimilar patch also gets a non-zero weight, restricting the search range was also shown to improve accuracy by Duval et al. [61]. Optimizations to the non-local means method were proposed [62, 42] making it up to $50\times$ faster on the CPU and implementations on the GPU achieved real-time denoising. Brox et al. [17] showed that a tree-based method could be used to obtain better

**Figure 5.2** The collection of 36 images used for evaluating our denoising performance.

quality for some inputs. They do increase the distance to include far away patches but searching is still limited to some local region.

The Non-Local Means (NLM) algorithm can be formally defined as follows: Given a noisy image $I(x_i, y_i)$, NLM computes the denoised value $I'(x_i, y_i)$ for a pixel $(x_i, y_i)$ by a weighted average of all pixels in a search range $S$.

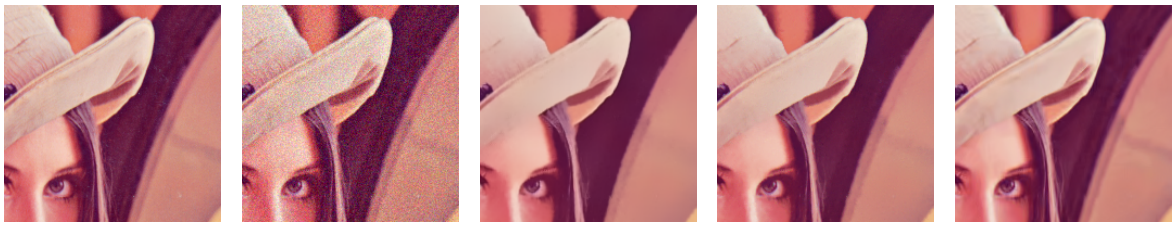
$$I'(x_i, y_i) = \sum_{(x_i, y_i) \epsilon S} w(i, j) I(x_j, y_j), \tag{5.1}$$

where the weight $w(i, j)$ of two pixels $(x_i, y_i)$ and $(x_j, y_j)$ depends on their patch-similarity, defined as:

$$w(i, j) = \frac{1}{Z(i)} e^{-\frac{\left\| P_i - P_j \right\|_2^2}{h^2}} \tag{5.2}$$

where $Z(i)$ is the normalizing constant, $h$ is a constant proportional to the noise deviation $\sigma$ and $P_i$ and $P_j$ are patches around $(x_i, y_i)$ and $(x_j, y_j)$ respectively.

This basic algorithm achieves high-quality results. However, contributing patches come only from a local window. Patch-matching algorithms like our Mixed-Resolution Patch-Matching (MRPM) [59] can be used to extend the scope of NLM by efficiently searching for similar patches throughout the image. We observe that the original non-local means algorithm denoises with more and not necessarily



(a) Original Image (b) Noisy Image (c) NLM (d) MRPM denoising (e) MR-NLM

**Figure 5.3** Denoising Results: The original image (a) is added with AWGN with $\sigma = 15$ in (b). In (c) it is denoised with NLM [18] (PSNR: 33.12) and with MRPM (PSNR: 33.05) in (d). The overall best result (PSNR: 33.93) is obtained in (e) denoised by MR-NLM: a combination of the two algorithms.

**Table 5.1** Denoising performance: Comparing the PSNR values for denoising our 36 image dataset with the Non-local means (NLM) [18] and our MRPM algorithm shows that MRPM performs better on self-similar images. Best results are obtained by combining the matches found by MRPM and NLM.

| **Average PSNR** | **NLM** | **MRPM** | **MR-NLM** |
|---|---|---|---|
| Over all Images | 31.82 | 31.45 | 32.17 |
| Self-Similar Images | 31.64 | 31.97 | 32.08 |

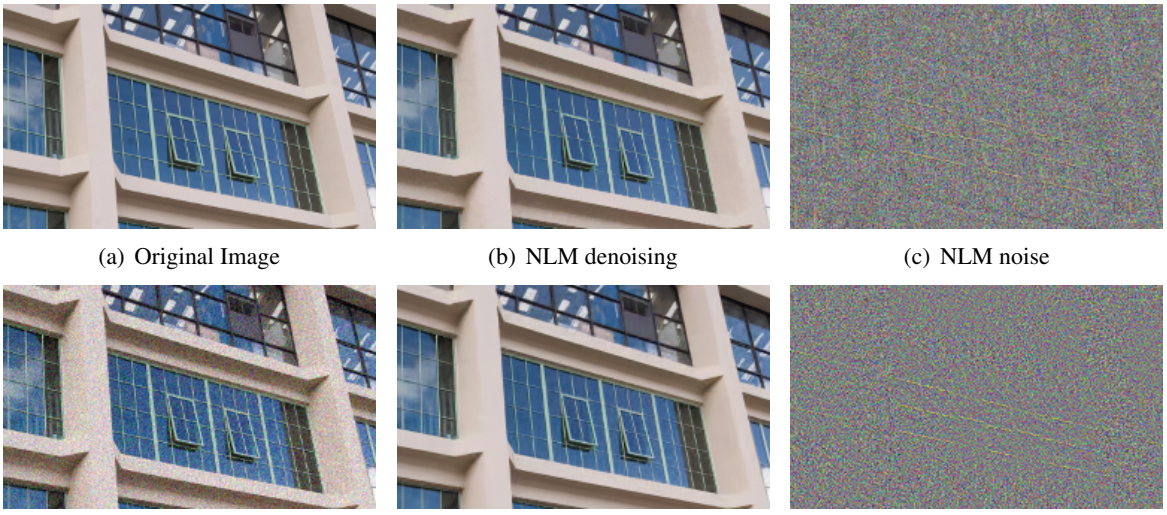
similar patches. Instead, denoising with fewer but mostly similar patches can provide better results. We implemented the simple algorithm of Buades et al. [18] and included an additional patch matching step using our MRPM algorithm. In contrast to local patches contributing to the denoising process, MRPM finds discrete windows spread globally that contain similar patches.

For evaluation, we collected 36 images to form our dataset shown in Figure 5.2. We corrupted these images with Additive White Gaussian Noise (AWGN) with zero variance and a standard deviation $\sigma = 15$. The source code for Non-Local Means (NLM) was taken from their web-page [4] and was executed with parameters of search window set to $11 \times 11$ and patch size $7 \times 7$. With MRPM, $K = 20$ most similar patches for each patch in the image were found and a weighted averaging over them was performed to obtain the denoised image. Both algorithms were run on an Intel i7 920 processor running at 2.67GHz with 3 GB of RAM. Counter-intuitively, when averaged over all images, the PSNR value achieved by our MRPM algorithm is worse, because it finds better matches. This occurs because our algorithm can search the entire image for a good match, therefore in uniform regions, the patch's noise pattern simply matches similar noise. Global search is particularly helpful for images with repeating patterns (self-similar images). We handpick such images from our dataset and calculate the average PSNR values over them. In these cases, we achieve better denoising quality as seen in Table 5.1.

We observe that instead of executing these algorithms against each other, they can also complement each other. We implement a combined version of these in which we list the patches found by both these algorithms and perform a weighted averaging over them. This new version of mixed-resolution with non-local means called **MR-NLM**, gives overall better denoising performance achieving a PSNR value of 32.17 as shown in Table 5.1. It can also be seen in Figure 5.3, which shows a crop of the lenna image denoised with the non-local means (NLM), mixed-resolution patch-matching (MRPM) and a mix of the two (MR-NLM). This performs best over all as well as over self-similar images that contain repeating elements. For a self-similar image from our dataset, Figure 5.4 shows the images denoised with NLM and MR-NLM. The method noise images for these are shown as well. It can be seen that our algorithm removes much less structure from the image during the denoising process.

**Table 5.2** Comparing GPU and CPU times for MR-NLM Image Denoising

| **CPU Time** | **GPU Time** | Speed-up |
|---|---|---|
| 49 sec | 2.58 sec | 19× |



(a) Original Image (b) NLM denoising (c) NLM noise

(d) Noisy Image (e) MR-NLM denoising (f) MR-NLM noise

**Figure 5.4** Denoising of images with repeating elements: Original image (a) is added with AWGN with $\sigma = 15$ in (d). It is denoised with NLM (PSNR: 33.35) in (b) and with MR-NLM (PSNR: 34.03) in (e). The method-noise image for MR-NLM in (f), removes much less structure from the image in the denoising process compared to NLM in (c).

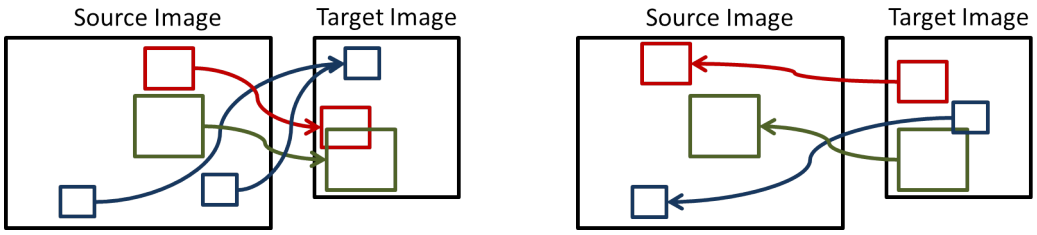
.

We also implement a parallel version of this MR-NLM denoising algorithm using our GPU implementation of MRPM described in Chapter 4. The images of our dataset were all resized to 0.3MP for the experiment and $K$ was set to $K = 20$. Table 5.2 compares the CPU times with those achieved on the GPU while quality was maintained. The GPU used was a NVIDIA GeForce GTX 580 card with 512 cores. A speed-up of $19\times$ is achieved on the GPU. We kept our GPU algorithm simple and optimizations to it can provide additional gain in speed.

### 5.1.2 Content Aware Image Resizing (Image Retargeting)

For the problem of image retargeting, or image summarization Simakov et al. [55] showed that high-quality summaries can be created by optimization of a well-defined bi-directional similarity measure. Contrary to the elegant method of Seam Carving [7] which removes uniform regions scattered throughout the image by carving out vertical and horizontal pixel-wide seams with low gradient content, Simakov et al. [55] proposes to use this bi-directional similarity measure to quantify how *good* the retargeted image is. This measure is based on the observation that two signals $S$ (input source signal) and $T$ (output target signal) are visually similar if as many as possible patches of $S$ are contained in $T$, and vice versa. For every patch $p \in S$, a search for the most similar patch $q \in T$ is done and the patch distances are computed, and vice-versa. This is illustrated in Figure 5.5

The bi-directional similarity equation has two terms: completeness and coherence. The completeness term ensures that $T$ represents all the visual data in $S$ by finding a good match for every source patch




(a) Completeness (b) Coherence

**Figure 5.5** The bidirectional similarity measure consists of the Completeness and the Coherence term. (Image courtesy: [55])

in $T$. The coherence term ensures that $T$ does not introduce new visual artifacts that were not observed in $S$. This is checked by finding a good match for every target patch in $S$. The overall value is given by adding the completeness and the coherence term.

$$d(S,T) = d_{complete}(S,T) + d_{coherent}(S,T) \tag{5.3}$$

Distances between patches are computed with a desired distance function $D(.,.)$. $D$ is small for similar patches and large for dis-similar patches. The similarity equation can thus be written to measure the **dis**similarity as follows:

$$d(S,T) = \frac{1}{N_S}\sum_{P \subset S} \min_{Q \subset T} D(P,Q) + \frac{1}{N_T}\sum_{Q \subset T} \min_{P \subset S} D(Q,P) \tag{5.4}$$

where $P$ and $Q$ are patches of $S$ and $T$, $N_S$ and $N_T$ are the number of patches in the source and the target image respectively. $D(.,.)$ is often the $L^2$ distance in the RGB space between a pair of patches.

Depending upon the application, relative weights can be added to the completeness and coherence terms of the bidirectional similarity measure.

$$d(S,T) = (\alpha)d_{complete}(S,T) + (1-\alpha)d_{coherent}(S,T) \tag{5.5}$$

Setting $\alpha = 0$ would employ only the coherence term and can be used for applications such as texture synthesis. It is similar to the objective function optimized in the data completion work of [66]. If the value of $\alpha$ is set to 1, only the completeness term would be effective. This can be used in applications where coherence is unimportant or is included in some other form. We discuss one such application "auto-crop" in the next section.

For the summarization problem, $\alpha$ is set to $0.5$. The algorithm aims to minimize $d(S,T)$ of equation 5.4. This is achieved by having two main components in the algorithm: 1) Iterative update rule and 2) Gradual resizing. Given a *good* initial guess of $T$, the iterative update rule finds the nearest patch in $T$ for every patch in $S$ and vice-versa. The new pixel value is given by a weighted average of the corresponding locations of a pixel in the patches it is matched to and in the patches that are matched to it. This process is repeated for a fixed number of iterations or until saturation, refining the initial guess.

(a) Original Image (b) Scaling

(c) PPM summary (d) MRPM summary

**Figure 5.6** PPM v/s MRPM summary. (a): The original image and which is directly down-sampled in (b). (c): Summary created by using the PPM algorithm described in Chapter 3. (d): A visually more pleasing summary is obtained by using our MRPM method which finds more accurate patch correspondences.

The gradual resizing process aims at finding this good initial guess. It shows that a gradually down-sampled version of the original image ($|T| = 0.95|S|$) can serve as a good starting point for the iterative refinement process. Hence, the algorithm first finds a target image larger than the one required. That target image is further downsampled to serve as the initial guess for the next iteration. This process continues until the desired image size and ratio are achieved.

Simakov et al. [55] show that good summaries are formed by their algorithm and they are able to change aspect ratios while preserving image detail. They observe that the nearest patch-matching step in their algorithm is the bottle-neck for efficiency. We implement a image summarization algorithm based on the framework of [55] by replacing the patch matching step with our MRPM algorithm. Figure 5.6 shows the results we achieve with our PPM and MRPM algorithms. For the summary generated with pyramidal patch-matching without mixing resolution levels, important features in the image are also averaged out (e.g. the man). MRPM finds more accurate matches which provides for generating a better summary. The matching was done by mixing 2 resolution levels ($l = 2$). Figure 5.7 shows how this algorithm can be used to change the aspect ratio of the image while generating perceptually good results. The aspect ratio in the figure is changed from $3 : 2$ to $1 : 1$. The summary result with the new aspect ratio is visually more pleasing than a directly scaled version.

(a) Original Image (b) Scaling (c) Summary

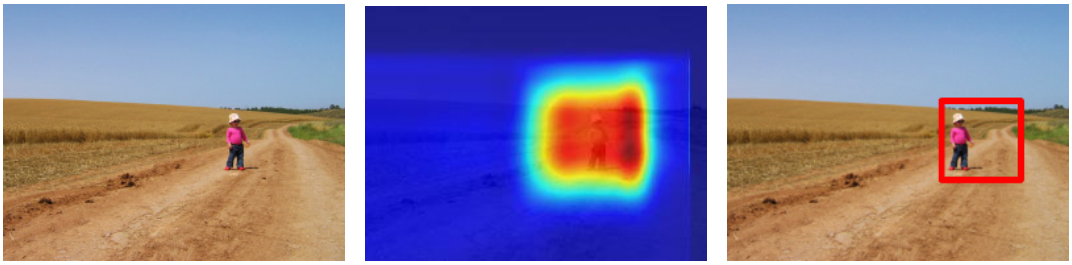
**Figure 5.7** Resizing with different aspect ratios. (a) shows the original image which is scaled to a different aspect ratio in (b). The resizing done using the bi-directional similarity algorithm [55] fitted with our MRPM method gives a visually more pleasing result as shown in (c).

To generate a summary half the size of the $250 \times 200$ image shown in Figure 5.6, Simakov et al. [55] take 5 minutes. Their algorithm fitted with our MRPM [59] for nearest patch matching is able generate this summary in just 34 seconds ( $9\times$ faster).

Certain parts (e.g. faces) of an image may be more important than the rest of the image. Simakov et al. [55] observe this and extend their algorithm to include weights per pixel for the summary in which a mask indicating areas of high importance can be given as input along with the original image. This idea can be further extended for object-removal and reshuffling of objects by providing a suitable importance mask and providing some user interaction. Taking the idea of bi-directional similarity further, Simakov et al. [55] show how their algorithm can also be used for texture synthesis and creating image montages. MRPM can act as the patch-matching engine for all these applications and quickly provide accurate matches for better quality of results.

### 5.1.3 Auto-Crop

Automatically detecting the most important part of an image is a problem that has been approached in several ways. The problem is often posed as a cropping problem which requires the algorithm to detect and crop this window which can be used as an enhanced version or as a thumbnail. Zhang et. al. [69] approach the problem by defining an objective function which composes of three sub-models of composition, conservation and penalty. They maximize this function to obtain an optimal crop. Santella et. al. [53] take an eye-tracking approach and use fixation data to identify important content in the image. This is then used to achieve a crop of desired aspect-ratio. Santella et. al. [53] also propose to take user opinion to verify the quality of the crop, providing a quality measure unavailable for this problem.



(a) Original Image (b) Heat-Map (c) Cropping Window Detected

**Figure 5.8** Auto-Crop: A window of the desired cropping size is moved around the original image (a). A saliency map giving the bi-directional similarity score for each window position is obtained as shown in (b). The window around the peak of this score is detected as most suitable for cropping, shown in (c).

Simakov et. al. [55] show how patch-matching can also be used for this problem. They define information content in a window to be high if all small patches in the original image can find a good match within the window. Since each window is a part of the original image, all patches are coherent. This is equivalent to setting $\alpha = 1$ in equation 5.5 where only the *completeness* term is important for this problem. The algorithm follows a sliding window approach. A window $W$ of size $m \times n$ (user defined) is moved over all possible locations in the image. All possible $W \subset S$ are candidates for the cropping window. Using the dissimilarity equation 5.4 with the coherence term muted, dissimilarity of every window position is calculated. The coherence term can be safely ignored since every window contains a continuous region from the image and is coherent with that part of the image.

Nearest match for every patch in the image are found within the window using MRPM. The dis-similarity score is calculated for every pixel by adding the distances of every patch with its match inside the window, centered around that pixel. This leads to a dis-similarity map with the least dis-similar window position getting the highest value, as shown in 5.8(b). Following the definition, the window around this pixel contains the most information. An example is shown in Figure 5.8. In case of the

(a) (b)

**Figure 5.9** Original images and cropped regions obtained by including MRPM in the framework of the algorithm by Simakov et. al. [55]. All small patches from the original image find a good match in the cropped window.

auto-crop application, this window would be chosen for cropping. Examples of regions cropped by the algorithm are shown in Figure 5.9. In contrast to previous approaches [53, 69], this approach makes no assumption of the importance of objects in the image. For example, faces and other small objects are treated similarly. However, it still achieves good results because its assumption that unique patches contain important information is often valid. The search for the nearest match for all patches is the bottle-neck of the algorithm. It is often implemented with recent patch-matching algorithms [12, 36]. However, these are not easy to parallelize. We develop an implementation for this approach which is up-to 70× faster than previous approaches on a GTX 580 NVIDIA graphics card.

### 5.1.4 Image Reconstruction

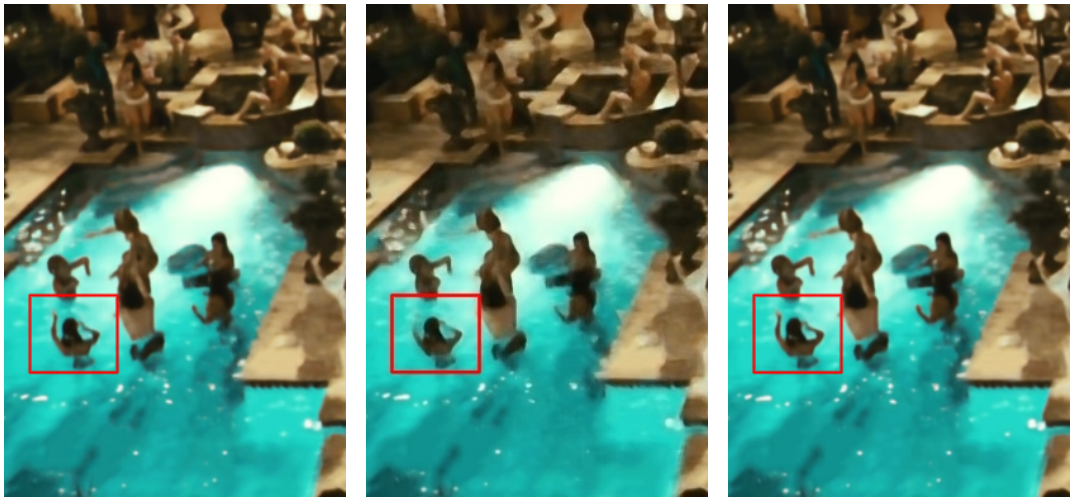
We also calculate the performance of our MRPM algorithm in a direct manner, using the re-construction of a source image A, given a target image B and a dense patch map from A to B. Original image A is available from which the dense patch-match is calculated using a patch-matching algorithm. The matching is then used to re-construct the image A from patches of image B. This kind of reconstruction requires coherent matches to be found and is a part of several patch-based methods. We perform the experiment as performed by Korman and Avidan [36]. It simply replaces each pixel with the average of the corresponding pixels that it is mapped to by all patches that contain it. Such averaging was shown to maximize the (Bi-)Directional Similarity from A to B [55]. For this experiment we used all images from the CSH Dataset [36], resized to 0.4 MP.

Each pixel is replaced by the average of corresponding pixels in each patch that was matched to all the patches that contain this pixel (for a $5 \times 5$ patch, a pixel belongs to 25 patches). The performance is quantified using an error which is the RMS distance (in RGB space) between original and reconstructed pixels. Table 5.3 shows the error values achieved by PatchMatch [12], CSH [36] and our approach. The base-line for comparison is set by executing CSH until saturation, referred to as $\text{CSH}_\infty$. Patchmatch [12] achieves an error value of 7.62 and CSH [36] of 6.29. Our MRPM approach obtains a average error value of 6.47 which is slightly worse than that of CSH. The baseline $\text{CSH}_\infty$ error value is 5.81. Please note that this base-line is different from the ground-truth which would be achieved by global exhaustive matching. Error values are lower if the matching algorithm enforces coherency. An exhaustive search might do even worse for this experiment.

Qualitative results are shown in Figure 5.10. The image shows the base-line reconstruction along with the reconstructed images using CSH and MRPM. The primary difference region is marked with a red box. It can be seen that CSH generates a good reconstruction, close to the base-line. MRPM also performs well, but fails to properly reconstruct a hand of the girl, which CSH reconstructs effectively.

**Table 5.3** Reconstruction error values of PatchMatch, CSH and MRPM compared with $\text{CSH}_\infty$.

| | PatchMatch | CSH | MRPM | $\text{CSH}_\infty$ |
|---|---|---|---|---|
| Reconstruction Error | 7.62 | 6.29 | 6.47 | 5.81 |



(a) CSH (b) MRPM (c) Base-Line

**Figure 5.10** Reconstructed images using CSH and MRPM are shown alongside the base-line reconstruction. The primary difference area is marked with a red box. CSH and MRPM both perform a good reconstruction but MRPM fails to reconstruct a hand of the girl.

Overall, MRPM performs slightly worse than CSH as it focuses on finding optimal matches which are different from coherent matches, a property required for this experiment. It performs reasonably well even without enforcing coherency.

As can be seen from Table 5.3, our error is marginally higher than CSH. Algorithms that enforce coherency perform better for image reconstruction. This is because the experiment averages pixels mapped to by neighboring patches that contain it. If the matching is more coherent, the same pixel location is averaged and reconstruction quality is higher. MRPM focuses on finding optimal matches and enforces coherency only by up-sampling. We do not give higher weights to coherent matches and find more accurate optimal matches (Section 4.5.2.2). Our performance on image reconstruction is comparable even without this facility of coherence. However, certain applications explicitly require coherent matches to be found. Hence, the experiment reveals that, as future work, we need to explore options of including coherency cues in our framework.

### 5.1.5 Discussion

In the previous section, we described only a few applications that perform a nearest-neighbor search and benefit by a fast and accurate patch-matching algorithm. The use of patch-matching is widespread in computer vision and image editing applications. Simakov et al. [55], Barnes et al. [13] and Xiao et al. [67] are some works which show these and several other patch-based vision applications accelerated by fast patch-matching algorithms. Figure 1.2 shows a collection of these, showcasing the wide scope for patch-matching algorithms in image editing and computer vision applications.

*Chapter 6*

# Conclusion and Areas of Future Research

We proposed a novel solution to the problem of patch-matching which matches K patches in image B for every patch of image A. We approached the problem with a simple coarse-to-fine multi-resolution framework. Unlike previous approaches which use complex tree-based, randomized or hashing based approaches, a simple pyramid structure was used to represent images at multiple resolution levels. The algorithm starts with the aim to find matches at the coarsest resolution. A global exhaustive search was performed there. The matches from the coarse resolutions are transferred to the next higher resolution by up-sampling. The search range for the up-sampled source region at this level was set by an expanded window around up-sampled candidate patches. This brought in the element of coherency by ensuring that neighboring query patches search the same regions for their matches. This process was carried out for every next-higher resolution level until the original resolution level. A final search here finds the required K matches. We called this the Pyramid Patch-Matching (PPM) algorithm. We used the exhaustive search in our experiments but our algorithm is flexible to accommodate other patch-matching and nearest-neighbor algorithms to perform the search step at each level. The choice of which depends upon the application.

Three techniques empower PPM. First, we observe that fine details are often missed due to early commitment in the pyramid based approach. To overcome this problem, we proposed the 'increase K' technique through which we store more than required number of matches at coarser resolutions. This ensures a wider search range at lower image resolutions which gradually decreases towards higher resolution levels. Increase-K improves the accuracy of PPM by up-to 4%. Second, following the down-sample rule, we matched small patch-sizes at coarser levels which maintained desired distinctiveness of patches and made matching efficient. Third, we note that the bottleneck of our algorithm is the distance-calculation step to measure similarity of patches. We significantly reduce this time by terminating the distance calculation early, if it exceeds the distance of the farthest match currently stored. This early-termination approach leads to a significant speed-up of up-to 1.7×. Overall, these improvements make PPM more accurate and efficient.

PPM found more accurate matches at similar speeds compared to state-of-the-art efforts. We performed experiments that demonstrate the benefits of PPM. It captures up-to 4% more of the ground truth

matches and achieves lower error values than Coherency Sensitive Hashing (CSH), the previous state-of-the-art. The experiments also reveal that the benefits of PPM are more pronounced when multiple and global matches are required.

Given the simplicity of our algorithm, we showed that fast parallel implementations of PPM are possible. Exploiting multiple cores of the CPU with openMP, we achieved a near-linear time speed-up with respect to the number of cores on the CPU. Each patch was handled by a CPU thread and was matched independently. Following this, we implemented the algorithm in CUDA to run on a Graphics Processing Unit (GPU). Since large number of threads are available on the GPU, we modified the parallel approach and assigned a patch-vector distance calculation to every thread. The distance array obtained for each patch was sorted in parallel and top K matches were picked. This GPU implementation achieved an additional $70\times$ speed-up compared to the CPU implementation run on a quad-core machine.

We studied the effects of various parameters involved in our algorithm. The pyramid-threshold used as the stopping criteria for down-sampling images in the pyramid showed that stricter thresholds (less number of pyramid levels) are more expensive to compute but provide accurate matching whereas lenient thresholds (more number of pyramid levels) are efficient but compromise accuracy. A balance of both time and accuracy could be achieved by setting the dimensions of the smallest image to be around 32-pixels. We show that the window around up-sampled matches (search-range) can be expanded further to increase accuracy. However, we observe that matching at lower-resolution levels suffers from smoothing due to down-sampling and patch-vectors contain significantly low information. We focus on including more information in these vectors and show that this increases accuracy more than increasing the search range. This was done by mixing information from finer resolution levels into patch-vectors of coarse levels. We called these Mixed-Resolution vectors.

We replaced patch-vectors with Mixed-Resolution vectors in the framework of Pyramid Patch-Matching and achieved an improvement in matching accuracy. The computation time increased marginally due to larger dimension vectors being matched but accuracy significantly improved. Information from several resolution levels can be combined and various ways could be chosen to pick relevant information from the finer levels. In our experiments, we chose to include information from the center of the up-sampled patch and showed that the increase in computational cost is acceptable only up-to mixing two resolution levels, above which the accuracy gain is not significant. We called this the Mixed-Resolution Patch-Matching (MRPM) algorithm. MRPM finds up-to 8% more ground-truth matches compared to previous approaches and consistently achieves lower error values. Following the parallel implementations of PPM, efficient multi-core and GPU implementations of MRPM were also presented.

From the variety of applications that use patch-matching algorithms at their core, we presented four. First, for image denoising we proposed to use MRPM to find the matches in the framework of non-local means (NLM) [18]. Counter-intuitively it achieved worse denoising performance than local NLM because it found better matches, matching similar noise to itself in uniform regions. We observed that both algorithms can complement each other and averaged all matches found by both algorithms achieving better denoising performance than either alone. We showed this combined algorithm removes much

less structure from the image in the denoising process. Second, for image summarization we follow the framework of Simakov et al. [55]. We replaced their patch-matching step with PPM and MRPM respectively. With PPM, the matching became up to $9\times$ faster and visual quality of results improved significantly when using Mixed-Resolution vectors. Similarly, we proposed a faster solution for auto-crop detecting the cropping window with accuracy. Finally, for the problem of image reconstruction, we showed that MRPM performs slightly worse than CSH [36]. This is justified since CSH enforces coherency which is vital for reconstruction. Up-sampling maintains coherency in MRPM but the experiment revealed that, in the future, more coherency cues can be included in MRPM's framework.

There are several directions for future work. Currently, the center region of the up-sampled patch is directly chosen to be concatenated with the patch-vector at the current level. The manner in which this information is included from the finer resolution level can be explored in-depth further, perhaps directing it by using some prior knowledge about images or image patches. The algorithm can also be extended to match patches across all rotations and scale (currently it only matches across translation). MRPM can naturally be extended to videos where 3-D patches can be matched efficiently. Efficient construction of the pyramid for a 3D space needs to be explored. Application specific cues can be included in the MRPM framework to improve accuracy for certain applications. The effects of using other search techniques for matching in the pyramid framework can also be explored.

We hypothesize that the Mixed-Resolution approach, which we presented for patch-matching, has scope in other domains also. Problems that make use of the multi-resolution framework are wide spread. We strongly believe that using Mixed-Resolution vectors for these problems can improve their accuracy. For example, stereo/optical-flow match patches are multiple scales to include large displacements of objects. Mixing resolutions while matching at coarser resolutions can include small scale objects in the search, which might get smoothed out due to down-sampling. Feature-descriptors like SIFT etc. often work at a single resolution level at a time. Mixed-Resolution descriptors can be developed which would store information from finer levels and make matching more accurate. These applications are numerous. Mixed-Resolution vectors also opens up the space for several new problems to use the multiresolution approach which earlier suffered early commitment and low confidence matching at coarse resolutions. We are excited about the opportunities this would create.

# Related Publication(s)

- Harshit Sureka, P. J. Narayanan. Mixed-Resolution Patch-Matching. In the 12th European Conference on Computer Vision, **ECCV 2012**, 7th to 13th October 2012, Florence, Italy.

# Bibliography


[1] A. Adams, N. Gelfand, J. Dolson, and M. Levoy. Gaussian kd-trees for fast high-dimensional filtering. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 21:1–21:12, New York, NY, USA, 2009. ACM.

[2] Y. Allusse, P. Horain, A. Agarwal, and C. Saipriyadarshan. GpuCV: A GPU-Accelerated Framework for Image Processing and Computer Vision. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, F. Porikli, J. A. Peters, J. Klosowski, L. Arns, Y. Chun, T.-M. Rhyne, and L. Monroe, editors, *Advances in Visual Computing*, volume 5359 of *Lecture Notes in Computer Science*, chapter 42, pages 430–439–439. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2008.

[3] C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. Pyramid methods in image processing, 1984.

[4] J. M. Antoni Buades, Bartomeu Coll. Nlm webpage: `http://www.ipol.im/pub/art/2011/bcm_nlm/`.

[5] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45:891–923, November 1998.

[6] M. Ashikhmin. Synthesizing natural textures. pages 217–226, 2006.

[7] S. Avidan and A. Shamir. Seam carving for content-aware image resizing. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[8] S. Bagon, O. Boiman, and M. Irani. What is a good image segment? a unified approach to segment extraction. In *Proceedings of the 10th European Conference on Computer Vision: Part IV*, ECCV '08, pages 30–44, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *In Proceedings of the IEEE International Conference on Computer Vision*, 2007.

[10] D. I. Barnea and H. F. Silverman. A class of algorithms for fast digital image registration. *IEEE Transactions on Computers*, 21:179–186, 1972.

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

[12] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. Patchmatch: a randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 28, 2009.

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

[14] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[15] J. Y. Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker: Description of the algorithm. `Jean-YvesBouguet`, 2002.

[16] T. Brox, C. Bregler, and J. Malik. Large displacement optical flow. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 41 –48, june 2009.

[17] T. Brox, O. Kleinschmidt, and D. Cremers. Efficient nonlocal means for denoising of textural patterns. *TIP*, 17(7), july 2008.

[18] A. Buades and B. Coll. A non-local algorithm for image denoising. In *CVPR*, 2005.

[19] A. Buades, B. Coll, and J. M. Morel. A review of image denoising algorithms, with a new one. *Multiscale Model. Simul*, 4, 2005.

[20] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, sep 2001.

[21] A. Criminisi, P. Perez, and K. Toyama. Object removal by exemplar-based inpainting. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–721 – II–728 vol.2, june 2003.

[22] N. CUDA:. `http://www.developer.nvidia.com/cuda`.

[23] K. Dabov. Image denoising with block-matching and 3d filtering. *SPIE*, 6064(30), 2006.

[24] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, SCG '04, pages 253–262, 2004.

[25] A. Efros and T. Leung. Texture synthesis by non-parametric sampling. In *In International Conference on Computer Vision*, pages 1033–1038, 1999.

[26] B. Fisher. `http://homepages.inf.ed.ac.uk/rbf/CVonline/`. CV Online:.

[27] D. Glasner, S. Bagon, and M. Irani. Super-resolution from a single image. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 349 –356, 29 2009-oct. 2 2009.

[28] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1992.

[29] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 327–340, New York, NY, USA, 2001. ACM.

[30] J. Hoberock and N. Bell. Thrust: A parallel template library, 2010. Version 1.3.0.

[31] H. Hotelling. Analysis of a Complex of Statistical Variables with Principal Components. *Journal of Educational Psychology*, 24:498–520, 1933.

[32] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. 1998.

[33] N. Komodakis and G. Tziritas. Image completion using efficient belief propagation via priority scheduling and dynamic pruning.

[34] J. Kopf, C.-W. Fu, D. Cohen-Or, O. Deussen, D. Lischinski, and T.-T. Wong. Solid texture synthesis from 2d exemplars. In *ACM SIGGRAPH 2007 papers*. ACM, 2007.

[35] S. Korman. `http://www.eng.tau.ac.il/~simonk/CSH/index.html`. CSH Webpage:.

[36] S. Korman and S. Avidan. Coherency sensitive hashing. *International Conference on Computer Vision*, 2011.

[37] N. Kumar, L. Zhang, and S. Nayar. 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, pages 364–378, 2008.

[38] S. Lefebvre and H. Hoppe. Parallel controllable texture synthesis. *ACM TRANSACTIONS ON GRAPHICS*, pages 777–786.

[39] W. Li and E. Salari. Successive elimination algorithm for motion estimation. *Trans. Img. Proc.*, 4(1):105–107, jan 1995.

[40] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov. 2004.

[41] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[42] M. Mahmoudi and G. Sapiro. Fast image and video denoising via nonlocal means of similar neighborhoods. *Signal Processing Letters, IEEE*, 12, 2005.

[43] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Non-local sparse models for image restoration. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2272 –2279, 29 2009-oct. 2 2009.

[44] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1615–1630, Oct. 2005.

[45] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *In VISAPP*, 2009.

[46] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

[47] S. A. Nene and S. K. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1997.

[48] NVIDIA. *NVIDIA CUDA Programming Guide 2.0*. 2008.

[49] S. M. Omohundro. Five balltree construction algorithms. Technical report, 1989. Technical Report 89-063.

[50] OpenMP Architecture Review Board. OpenMP application program interface version 3.0, may 2008.

[51] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, May 2008.

[52] M. Rubinstein, A. Shamir, and S. Avidan. Improved seam carving for video retargeting. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, pages 16:1–16:9, New York, NY, USA, 2008. ACM.

[53] A. Santella, M. Agrawala, D. DeCarlo, D. Salesin, and M. Cohen. Gaze-based interaction for semi-automatic photo cropping. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 771–780, New York, NY, USA, 2006. ACM.

[54] A. Shamaie and A. Sutherland. Graph-based matching of occluded hand gestures. In *Applied Imagery Pattern Recognition Workshop, AIPR 2001 30th*, pages 67 –73, oct 2001.

[55] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani. Summarizing visual data using bidirectional similarity. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008. CVPR 2008*, pages 1–8, 2008.

[56] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani. Summarizing visual data using bidirectional similarity. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, june 2008.

[57] I. Simon and S. Seitz. A probabilistic model for object recognition, segmentation, and non-rigid correspondence. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1 –7, june 2007.

[58] R. F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6(4):579–589, 1991.

[59] H. Sureka and P. Narayanan. Mixed-resolution patch-matching. In *Computer Vision – ECCV 2012*, volume 7577 of *Lecture Notes in Computer Science*, pages 187–198. Springer, 2012.

[60] Trevor and P. Indyk, editors. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, 2006.

[61] J.-F. A. Vincent Duval and Y. Gousseau. On the parameter choice of the non-local means. *CMLA Preprint 2010-06*, March 2010.

[62] J. Wang, Y. Guo, Y. Ying, Y. Liu, and Q. Peng. Fast Non-Local Algorithm for Image Denoising. 2006.

[63] Y.-S. Wang, C.-L. Tai, O. Sorkine, and T.-Y. Lee. Optimized scale-and-stretch for image resizing. *ACM Trans. Graph.*, 27(5):118:1–118:8, Dec. 2008.

[64] L.-Y. Wei, J. Han, K. Zhou, H. Bao, B. Guo, and H.-Y. Shum. Inverse texture synthesis. *ACM Transactions on Graphics*, 27(3), 2008.

[65] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[66] Y. Wexler, E. Shechtman, and M. Irani. Space-Time Completion of Video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3):463–476, 2007.

[67] C. Xiao, M. Liu, N. Yongwei, and Z. Dong. Fast exact nearest patch matching for patch-based image editing and processing. *IEEE Transactions on Visualization and Computer Graphics*, 17:1122–1134, 2011.

[68] P. N. Yianilos. 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, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[69] M. Zhang, L. Zhang, Y. Sun, L. Feng, and W. Ma. Auto cropping for digital photographs. In *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*, page 4 pp., july 2005.