PatchMatch: A Fast Randomized Matching Algorithm with Application to Image and Video

Connelly Barnes

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Adam Finkelstein

May 2011

Abstract

This thesis presents a novel fast randomized matching algorithm for finding correspondences between small local regions of images. We also explore a wide variety of applications of this new fast randomized matching technique.

The core matching algorithm, which we call PatchMatch, can find similar regions or “patches” of an image one to two orders of magnitude faster than previous techniques. The algorithm is motivated by statistical properties of nearest neighbors in natural images. We observe that neighboring correspondences tend to be similar or “coherent” and use this observation in our algorithm in order to quickly converge to an approximate solution. Our algorithm in the most general form can find k-nearest neighbor matchings, using patches that translate, rotate, or scale, using arbitrary descriptors, and between two or more images. Speed-ups are obtained over alternative techniques in a number of these areas. We analyze convergence both empirically and theoretically for many of these image matching algorithms.

We have explored many applications of this matching algorithm. In computer graphics, we have explored removing unwanted objects from images, seamlessly moving objects in images, changing image aspect ratios, and video summarization. Because our technique for removing unwanted objects from photographs is both high quality and interactive, due to the fast matching algorithm, it has been included in Adobe Photoshop CS5 as a new feature “content aware fill.” In computer vision we have explored denoising images, object detection, detecting image forgeries, and detecting symmetries. We also apply our algorithm to large collections of images. We conclude by discussing the limitations of our algorithm and areas for future research.

Acknowledgements

I would like to thank my advisor, Adam Finkelstein, for supporting me and being a good role model.

I was fortunate to work with a number of collaborators: Jia Deng, Dan B Goldman, Hugues Hoppe, David Jacobs, Jacob Lewellen, Jason Sanders, Eli Shechtman, Szymon Rusinkiewicz, and Tim Weyrich. Special thanks go to Dan Goldman and Eli Shechtman for providing research mentorship during my three internships at Adobe Systems. From you and my advisor, I have learned a great deal about research, career, and life. Thanks to the other Princeton graphics lab members, who provided useful research advise, including Aleksey Boyko, Xiaobai Chen, Forrester Cole, Tom Funkhouser, Vladimir Kim, Jingwan Lu, Linjie Luo, and Corey Toler-Franklin. Thanks also to the reviewers of papers at TIGGRAPH and Adobe CTL Retreat.

Special thanks to our collaborator Jacob Lewellen who worked on PatchWeb for his undergraduate thesis at Princeton.

I thank the following funding sources for sponsoring this work: National Science Foundation under grant IIS-0511965 and 0937139, Adobe Systems Inc, and Microsoft.

We would like to thank a number of artists for their original works. We thank Lorelay Bove for the color script. The short film *Kind of a Blur* is Copyright Skunk Creek Productions. The films *Elephants Dream* and *Big Buck Bunny* are licensed as Creative Commons by the Blender Foundation. The film *Star Wreck* is licensed as Creative Commons by Energia Productions. We thank the following Flickr users for Creative Commons imagery: Moogs, Swami Stream, Laurence & Annie, Bill Liao, Badwsky, Cwalker71, Swamibu, Stuck in Customs, Xymox, Chris Penn, Eric Brumble, Kibondo, Mazzaq Mazzacurati, Slack12, Thomashawk, Bex in Beijing, Paul Keleher, CarbonNYC, SteveWhis, Arranging Matches, Professor Bop, Whirling Phoenix, Cindy47452, Sevenbrane, Wili, Moi of Ra, Celie, and Bart van der Mark.

Finally, I thank everyone who supported me during my graduate school. Particularly, my parents, brother, and grandparents have provided me support and encouragement. Thanks to my friends that I have made at activities such as tango, rock climbing, skiing, and broomball.

To my parents for their love and encouragement.

Contents

Chapter 1

Introduction

1.1 Motivation

As digital photography has matured, researchers have developed a number of techniques for manipulating and understanding image and video at a high level. For example, recent algorithms for *image retargeting* allow images to be resized to a new aspect ratio – the computer automatically produces a good likeness of the contents of the original image but with new dimensions [94, 116]. Other algorithms for *image completion* let a user simply erase an unwanted portion of an image, and the computer automatically synthesizes a fill region that plausibly matches the remainder of the image [29, 60].

Many of these algorithms use as a core observation that an image can be broken down into small patches, that is, rectangles of fixed size, say 7x7 pixels. The images can then be manipulated by *patch-based synthesis*: the algorithm rearranges these patches, similar to a human rearranging a jigsaw puzzle. In the general case, however, manipulating images by patches previously carried considerable overheads in time and memory, because searching for similar patches was difficult.

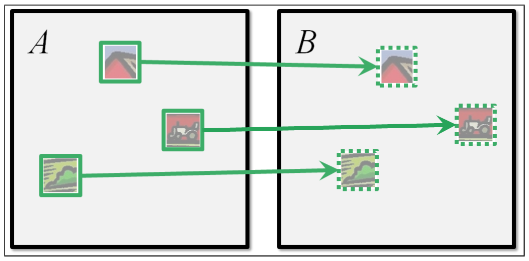


Figure 1.1: The matching problem we solve: for each patch in image A, find the nearest neighbor patch in image B.

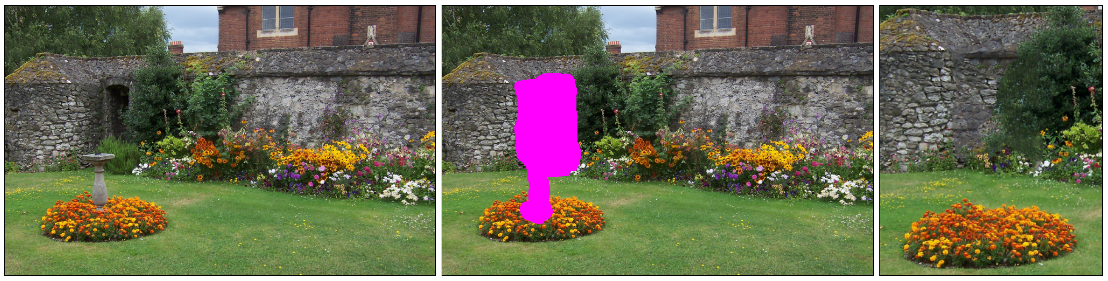

(a) Input image (b) Hole selected (c) Output image

Figure 1.2: Illustration of image completion. An input image (a) is provided; the user selects the region (b) to be removed; then the output image (c) is constructed by matching similar textural patterns elsewhere in the image. The selected object is thus removed.

In this thesis, we describe a novel algorithm, PatchMatch [8], and its generalizations [10], which greatly accelerate the search problem for image patches. For simple image matching problems, our algorithm is 20-100x faster than previous work, and uses substantially less memory. We solve the following problem: for every overlapping patch in image A, we wish to find the nearest neighbor patch in image B. This is shown in Figure 1.1. We solve the problem only for the case of *densely overlapping* patches: one patch is defined around every pixel. Our algorithm converges due to a prior assumption about *statistics of natural images*, specifically that the correspondences between spatially adjacent patches are highly correlated.

This accelerated algorithm in turn, permits new applications in image manipulation, video summarization, and computer vision, also described herein.

1.2 Why Accelerate Patch-Based Methods?

To illustrate the need for a fast search algorithm for image patches, consider the problem of image completion. As shown in Figure 1.2, the user selects an unwanted object to be removed from a photograph, shown in magenta. Using the algorithm of Wexler et al.[119], the hole to be filled is first replaced with an initial guess, smoothly interpolated from the boundary. The image is divided into overlapping patches. The algorithm then iteratively finds good correspondences from patches in the partially completed result inside the hole to the image contents outside the hole. Next, the matched features outside the hole are pasted to the corresponding location inside the hole, averaging "votes" for the overlapping patches. The inner loop is iterated, and repeated at multiple scales from a coarse resolution to a fine resolution, eventually converging to a seamless result.

In this process, the bottleneck is the search for good matching patches from inside the hole to outside the hole. Similar synthesis frameworks can be used to also automatically retarget images, that is, change their aspect ratio, among other image applications. Therefore we wish to accelerate this bottleneck in the synthesis process. Particularly, for these applications, we show that using our algorithm can speed up the synthesis to interactive rates, allowing new user interactions and controls. Thus our algorithm allows for more than simple speed ups and memory reduction, it also permits qualitative changes in the controls available to the user.

Moreover, there is a large and growing body of work in patch-based synthesis and analysis, for images, video, computational photography, 3D geometry, and computer vision. We will review this work in detail in other chapters, however we note that our accelerated search method can be used as a drop-in tool in many of these methods. Even in cases where our algorithm cannot be applied as a black-box tool, we hypothesize that it can be adapted to the particular problem, with similar observations as we used.

1.3 Overview

Here we give a brief overview of the remainder of the thesis.

1.3.1 The PatchMatch Algorithm

We first motivate the core matching algorithm with statistical observations about nearest neighbors in natural images (Chapter 2). We then present PatchMatch, a fast randomized algorithm for finding approximate nearest neighbors on densely sampled descriptors such as patches (Chapter 3). For simplicity, we first describe the algorithm in a simplified form that finds a single nearest neighbor for patches that translate only. Then we present the generalized algorithm, which can match k-nearest neighbors, across arbitrary scales and rotations, and optionally match arbitrary descriptors, such as densely sampled SIFT features.

The core algorithm can be used as-is, however, there are a number of implementation and search strategies that can accelerate the algorithm. We discuss these in Chapter 4. First, we discuss strategies for parallelizing the algorithm on the CPU and GPU. Next, we discuss details in optimizing the algorithm by for example avoiding redundant computations. Then we discuss alternative search strategies that help constrain the search space especially when matching large numbers of images. Building on this, we finally discuss PatchWeb, an algorithm for finding dense matches between thousands or millions of images. PatchWeb tackles scalability concerns when all images cannot fit in core memory, and also scales on to cluster architectures.

The remainder of this chapter discusses three applications of PatchMatch in our own research: image editing, video summarization in the form of "tapestries," and applications in computer vision.

1.3.2 Image Editing Applications

In Chapter 5, we look at methods of editing photographs using high-level controls to meet user goals. First, *image retargeting* algorithms allow images to be resized to a new aspect ratio, while the computer avoids stretching or other distortions. Second, *image completion* algorithms allow a user to simply erase an unwanted portion of an image, and the computer automatically synthesizes a plausible fill region. Third, *image reshuffling* algorithms make it possible to grab portions of an image and move them around — the computer automatically synthesizes the remainder of the image.

In each scenario, user interaction is essential for best results. This is for two reasons: first, the algorithms sometimes require user intervention to obtain best results, and second, because the goals needed cannot always be articulated *a priori* by the user, an artistic process of trial and error is needed to repeatedly optimize the result. Our fast matching algorithm can be used to greatly accelerate algorithms that perform these image editing operations, reducing their running times from minutes to seconds. This permits for the first time new interactive algorithms for high quality, intuitive image editing. Because our technique for removing unwanted objects from photographs is both high quality and interactive, due to the fast matching algorithm, it has been included in Adobe Photoshop CS5 as a new feature called "content aware fill."

1.3.3 Video Tapestries

In Chapter 6, we present a novel approach for summarizing video in the form of a multiscale image that is continuous in both the spatial domain and across the scale dimension: There are no hard borders between discrete moments in time, and a user can zoom smoothly into the image to reveal additional temporal details. We call these artifacts tapestries because their continuous nature is akin to medieval tapestries and other narrative depictions predating the advent of motion pictures.

We propose a set of criteria for such a summarization, and a series of optimizations motivated by these criteria. These can be performed as an entirely offline computation to produce high quality renderings, or by adjusting some optimization parameters the later stages can be solved in real time, enabling an interactive interface for video navigation. The efficiency of both the offline and the real time renderings is due to the use of the matching algorithm, PatchMatch, used under the hood for patch-based image synthesis.

Our video tapestries combine the best aspects of two common visualizations, providing the visual clarity of DVD chapter menus with the information density and multiple scales of a video editing timeline representation. In addition, they provide continuous transitions between zoom levels. In a user study, participants preferred both the aesthetics and efficiency of tapestries over other interfaces for visual browsing.

1.3.4 Computer Vision Applications

In Chapter 7, we explore several applications in computer vision that use the full power of the generalized matching algorithm. We investigate *non-local means denoising*: removing noise from images by finding and averaging repeating non-local patterns. We also investigate detecting digitally forged images where the forgery uses a clone tool. We present a simple method for detecting translational symmetries in the form of regular lattices. Finally, we investigate object detection using patches that rotate and are compensated for lighting, as well as SIFT features.

1.3.5 Discussion and Future Work

We conclude by discussing future work that could be done on PatchMatch, and potential future applications. Based on the uses that have been found so far, we believe the core algorithm can be widely useful as a tool for solving problems in image, video, computer vision, and potentially other domains, particularly when the search strategies are adapted for the problem domain.

1.4 Contributions of This Thesis

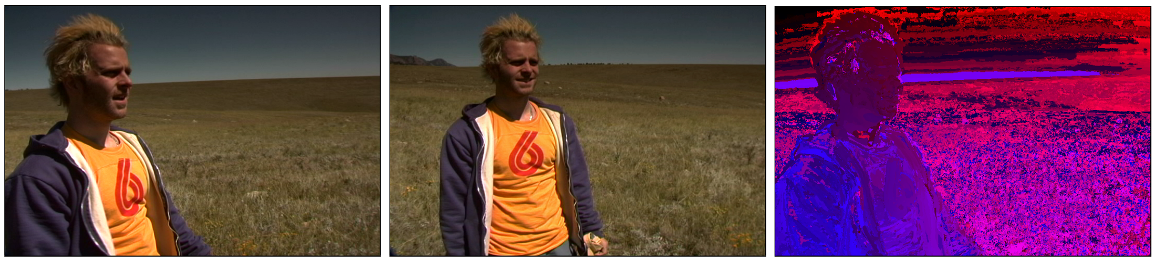
This thesis presents our previously published work describing the simple matching algorithm [8], the generalized matching algorithm [10], as well as vision and graphics applications, and video summarization applications [9]. The novel contributions made in this thesis are matching algorithms for large image datasets (Section 4.3), as well as observations about the statistics of natural images (Chapter 2) that motivate our algorithm.

Chapter 2

Statistics of Nearest Neighbors

Recall that our problem is to find, for every overlapping patch in image A, the nearest neighbor patch in image B. Nearest neighbor can be defined in terms of any patch distance metric D that measures how similar two patches are. For example, we commonly use an L^2 patch distance metric, which measures the distance between two patches by taking the L^2 norm of the difference of corresponding RGB values in the patch. We note that our nearest neighbors are not the same as those used for an optical flow field — our nearest neighbors use no smoothness constraints and find the best match independent of neighboring matches.

We observe two properties of natural images: *coherence* of nearest neighbors, and a *peaked distribution* of where better nearest neighbors are located. These motivate two stages of our algorithm: *propagation*, and *random search*, respectively, which guide an iterative search process according to these prior assumptions about our image input, as will be described in Chapter 3.



(a) Image A (b) Image B (c) Match coordinates

Figure 2.1: Illustration of coherence. Two different images A and B are selected. In (c) we show the ground truth nearest neighbor coordinates. For each patch in image A, we visualize the nearest neighbor patch in image B with relative x coordinate as red and y as blue. Coherent regions have the same color. Note the effective coherence is actually higher, because many regions simply dither between one of several different possible nearest neighbors, each of which has the same low matching error.

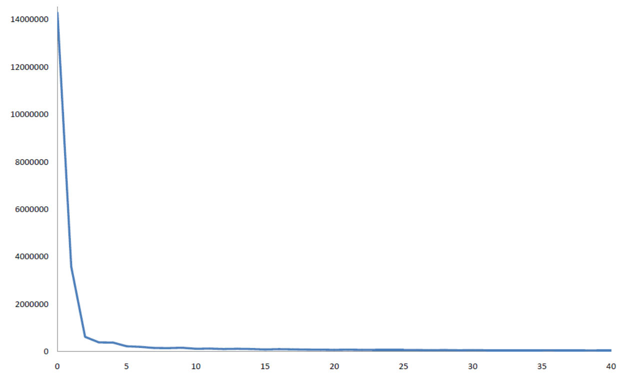


Figure 2.2: Histogram showing a correlation between adjacent patches' nearest neighbors. On the horizontal axis, we measure how far apart are the nearest neighbors of adjacent patches, in terms of Euclidean 2D distance. On the vertical axis, we count the number of patches with the given distance. Note that most patches have zero distance, indicating perfect coherence, but there is a tail, generally decreasing, with high likelihoods of distance $1, 2$, or 3 pixels.

First we discuss coherence. In general, we expect patch appearances to change slowly as a patch is moved one pixel in any direction, particularly for large patches. Thus we expect there to be a high amount of *coherence* in the nearest neighbor matches. We say two correspondences are coherent if they have the same relative coordinates, that is, their position in image B minus position in image A is the same. For any two images A and B, this coherence can be visualized by calculating the correct ground truth nearest neighbors from A to B, and visualizing the relative coordinates of the nearest neighbors, as shown in Figure 2.1. Coherent regions have the same color. Our algorithm takes advantage of this coherence property by *propagating* good matches to spatially adjacent pixels.

However, even when nearest neighbors of adjacent patches are not exactly coherent, they are often spatially proximate to each other, and follow a *peaked distribution*. This property is not true for all possible inputs, but it is true for the set of *natural images* — photographs taken by people. This is a very wide class of images that is commonly input to algorithms. Thus we wish to accelerate our algorithm by taking advantage of these peaked distributions.

We can visualize the peaked distribution in a few ways. First, we can ask for horizontally and vertically adjacent patches, how far apart are their nearest neighbors? We visualized this in Figure 2.2, for wide baseline stereo pairs in our large dataset of natural images (other categories in our dataset result in similar graphs). We find that many of the patches have spatial distance zero from their horizontally or vertically adjacent neighbor, indicating exact coherence. However, of the remaining patches, many have small spatial distances of say $1, 2$, or 3 pixels, and the distribution is generally decreasing, with a long tail, so very few adjacent patches have nearest neighbors that are all the way across the image from each other. Our full dataset of image pairs is shown in Figure 2.3.

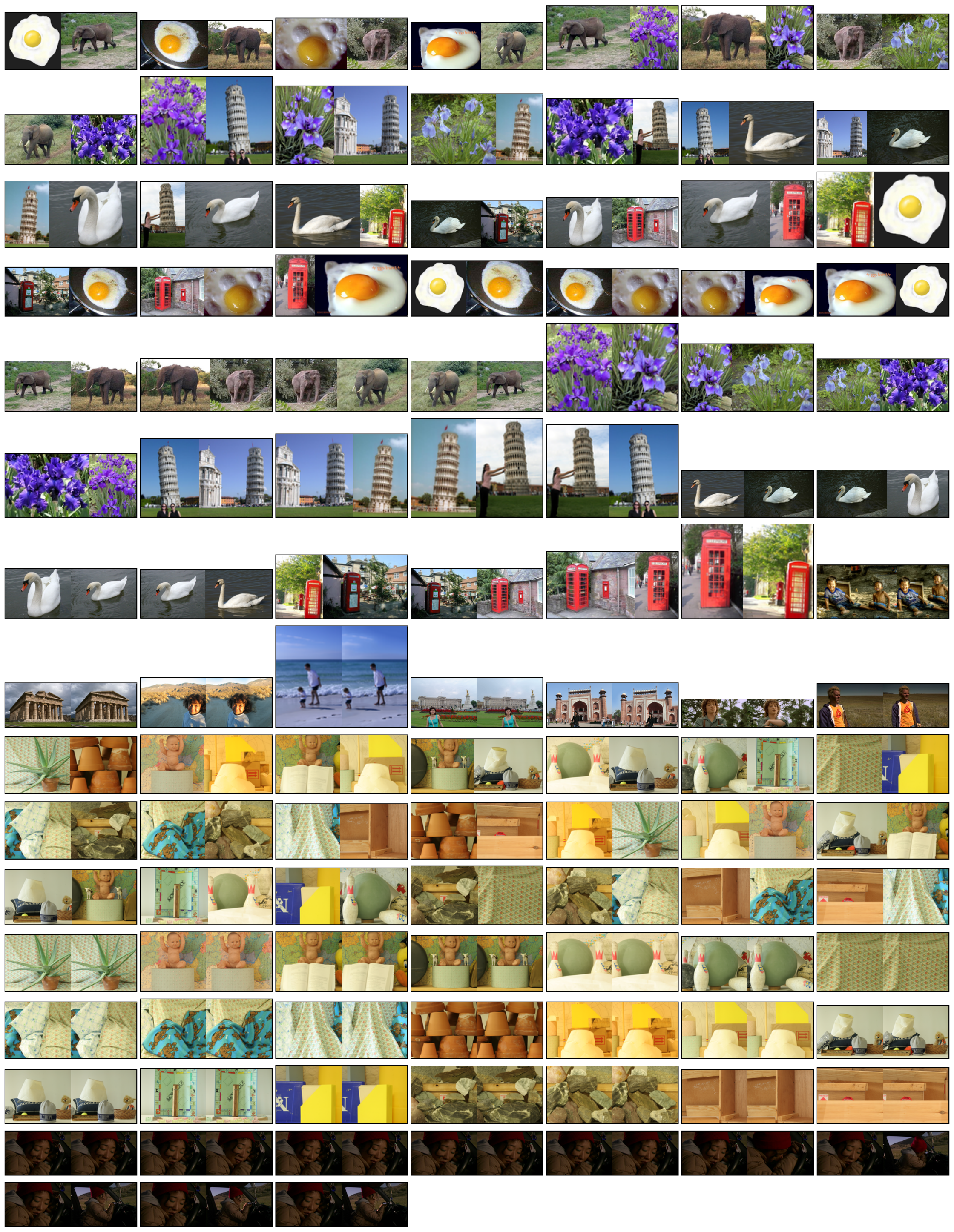


Figure 2.3: Dataset of 108 image pairs used to produce the histograms in this chapter. This consists of stereo pairs, similar images, dissimilar images, and frames taken from near and far times in videos. Images are from the Caltech-256 dataset, the Middlebury stereo pair dataset, and from the short film *Kind of a Blur* by Jon Goldman. There are 24 image pairs from different classes, 24 pairs from the same class, 6 similar pairs of input and output from our image editing applications, 21 wide baseline stereo pairs, 21 mismatched image pairs from the stereo dataset, and 12 pairs from near and far times in video.

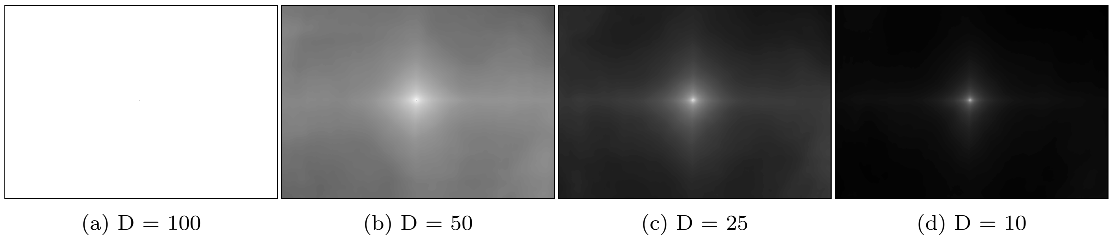

(a) D = 100 (b) D = 50 (c) D = 25 (d) D = 10

Figure 2.4: 2D histogram showing peaked distribution of where better matches are located. The center denotes the current match's location. For correspondences with high patch distance D (a) better matches are uniformly distributed throughout the image. As patch distance is lowered, (b) the better matches become more peaked, with more good matches nearby the current match. Even lower patch distance (c), (d) causes further peaking behavior. These histograms are measured from a dataset of 108 pairs of matched images, both similar and dissimilar.

Another way to visualize this peaked distribution is to ask: suppose we have a current, suboptimal correspondence for a single patch, which has some distance D associated with it. Then keeping the source location in image A fixed, where are the better target positions for the correspondence in image B relative to our current target position? Integrating over all possible correspondences, we can plot a 2D histogram of intensity versus x, y relative position, where intensity represents the probability that a given relative offset in the target image B will yield a better correspondence. These plots are shown in Figure 2.4, after integrating over our full dataset of 108 natural images.

Note that the 2D histograms of where better neighbors are located are not uniform, but instead follow a peaked distribution. When the distance D of our correspondence is high, the better locations to look at are uniformly distributed across the image. As the distance D is lowered, the better locations become more clustered around the current location (origin), until eventually, for very low distances, it is good to search very close to the current position.

Observe also that these priors do not hold for all possible input images, but only the large set

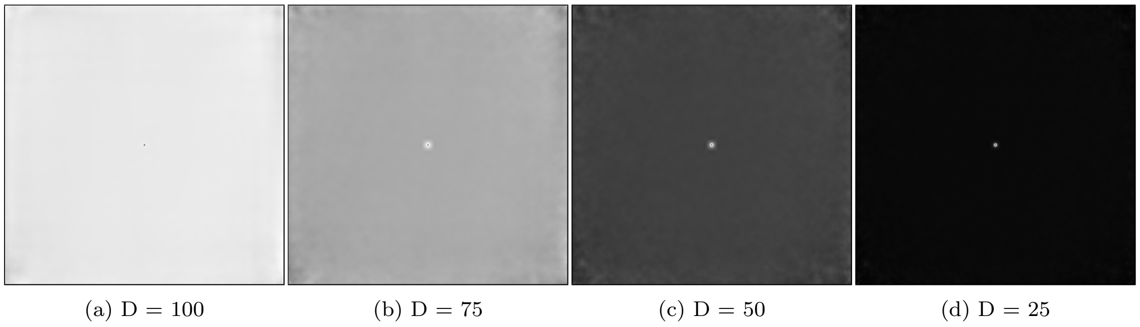
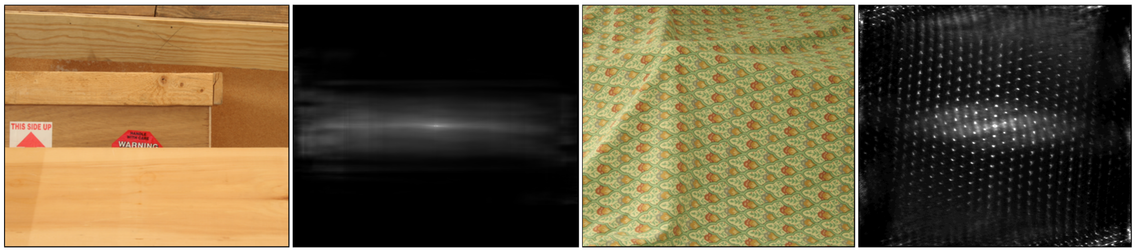

(a) D = 100 (b) D = 75 (c) D = 50 (d) D = 25

Figure 2.5: 2D histogram of where better matches are located, matching random images, with Gaussian, uniform, or octave random noise. Unlike natural images, there is little peaking, and better matches are distributed uniformly across the image. The small peak in the center is due to the 7x7 patch size, which causes a small amount of coherence.



(a) Edges image (b) Edges histogram (c) Patterns image (d) Patterns histogram

Figure 2.6: 2D histogram showing where better matches are located, for specific image pairs. The first stereo image pair, edges (a), contains strong horizontal edges. In (b), a 2D histogram shows where better matches are found relative to a current match. Note the horizontal bias, indicating it would be more efficient to sample along the direction of the edge to find better matches. Stereo image pair (c) has repetitive patterns, giving a 2D histogram (d) of better matches, that indicates once a good match is found, better matches might be found by sampling along the lattice spacing.

of natural images that users tend to be interested in. For example, Figure 2.5 shows the same 2D histograms of where better neighbors are located, averaged across a set of images that contain Gaussian, uniform, and octave random noise. The distributions are nearly uniform, with almost no peaks, indicating that better matches in a random image are not necessarily spatially nearby. Therefore our algorithm will converge more slowly on this input, because it does not follow our prior assumptions.

Finally, we note for future work that many other statistical properties of natural images could be investigated. For example, the 2D histograms of Figure 2.4 are averaged over a large dataset, but when we examine individual pairs of images, we find great variation. Image pairs that have strong horizontal edges cause a horizontal bias in the distribution, and images with repetitive patterns cause a repetitive pattern in the distribution, as shown in Figure 2.6. For other domains such as video or 3D geometry, the statistics may exhibit even more subtle properties.

Chapter 3

The PatchMatch Algorithm

In this chapter we describe our core matching algorithm, which accelerates the problem of finding nearest neighbor patches by 20-100x over previous approaches. Our algorithm is a randomized approximation algorithm: it does not always return the exact nearest neighbor, but returns a good approximate nearest neighbor. We develop our algorithm in the context of image search, but it generalizes and can also be applied to at least 1D or 3D signals.

For simplicity we first present a special case of the algorithm that finds only a single nearest neighbor using L^2 patch distance between patches that only translate. Later we present the full generalized algorithm that can match across rotations, scales, find k-nearest neighbors, and match arbitrary descriptors. We analyze convergence of these algorithms both empirically and theoretically.

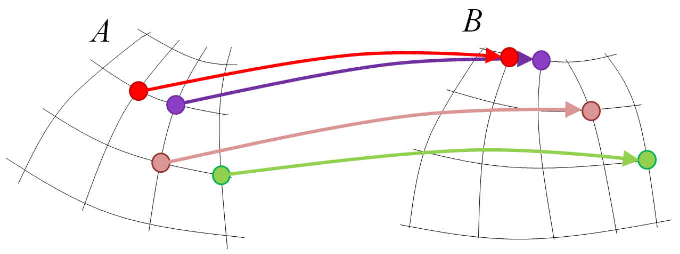


Figure 3.1: High level motivation behind the algorithm. We match descriptors computed on two manifolds, visualized as color circles. Each descriptor independently finds its most similar match in the other manifold. Coherence here could be indicated by the red descriptor being similar to the purple descriptor, and that their nearest neighbors are spatially close to each other.

3.1 High Level Motivation

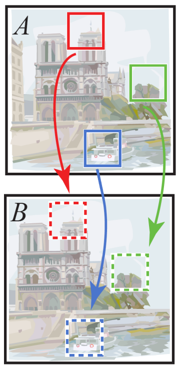
The high level intuition behind our algorithm is shown in Figure 3.1. We have two manifolds A and B with descriptors computed at lattice points, visualized as colored circles. We wish to find for each descriptor in A, the most similar descriptor in B. We do this by taking advantage of spatial locality properties observed in the previous chapter: when we have a good match, we can *propagate* it to adjacent points on the lattice, and if we have a reasonable match, we can try to improve it by *randomly searching* for better matches around the target position. The first stage — propagation — takes advantage of the property that many matches are *coherent*, or have the same relative matching coordinates, as discussed in the previous chapter (Figure 2.1 and Figure 2.2). The second stage — random search — looks for better correspondences relative to the current correspondence's target position, according to a peaked search distribution, similar to the measured distributions in Figure 2.4.

In this chapter, we develop our algorithm for 2D images, which have a regular lattice of the positions of all pixels. However, our algorithm has also been applied to 1D contours and 3D geometric data, so we can imagine generalizing these propagation and random search operations to any space that locally is Euclidean or nearly so.

3.2 Introduction

Many methods recently have been developed for manipulating images at a high level, such as *retargeting* algorithms that change image aspect ratios, or *image completion* algorithms that remove unwanted objects from photographs. Many of the most powerful of these methods are *patch based*: they divide the image into many small, overlapping rectangles of fixed size, called patches.

To understand our matching algorithm, we must consider the common components of patch based algorithms: The core element of nonparametric patch sampling methods is a repeated search of all patches in one image region for the most similar patch in another image region. In other words, given images or regions A and B, find for every patch in A the nearest neighbor in B under a patch distance metric such as L^p. We call this mapping the *Nearest-Neighbor Field* (NNF), illustrated schematically in the inset figure. Approaching this problem with a naïve brute force search is expensive – $O(mM^2)$ for image regions and patches of size M and m pixels, respectively. Even using acceleration methods such as *approximate nearest*



neighbors [78] and dimensionality reduction, this search step remains the bottleneck of nonparametric patch sampling methods, preventing them from attaining interactive speeds. Furthermore, these tree-based acceleration structures use memory in the order of $O(M)$ or higher with relatively large constants, limiting their application for high resolution imagery.

To efficiently compute approximate nearest-neighbor fields our new algorithm relies on three key observations about the problem:

Dimensionality of offset space. First, although the dimensionality of the patch space is large (m dimensions), it is sparsely populated ($O(M)$ patches). Many previous methods have accelerated the nearest neighbor search by attacking the dimensionality of the patch space using tree structures (e.g., *kd*-tree, which can search in $O(mM \log M)$ time) and dimensionality reduction methods (e.g., PCA). In contrast, our algorithm searches in the full 2-D space of possible patch offsets, achieving greater speed and memory efficiency.

Natural structure of images. Second, the usual independent search for each pixel ignores the natural structure in images. In patch-sampling synthesis algorithms, the output typically contains large contiguous chunks of data from the input (as observed by Ashikhmin [3]). Thus we can improve efficiency by performing searches for adjacent pixels in an interdependent manner.

The law of large numbers. Finally, whereas any one random choice of patch assignment is very unlikely to be a good guess, some nontrivial fraction of a large field of random assignments will likely be good guesses. As this field grows larger, the chance that no patch will have a correct offset becomes vanishingly small.

Based on these three observations we offer a randomized algorithm for computing approximate NNFs using incremental updates (Section 3.4). The algorithm begins with an initial guess, which may be derived from prior information or may simply be a random field. The iterative process consists of two phases: *propagation*, in which coherence is used to disseminate good solutions to adjacent pixels in the field; and *random search*, in which the current offset vector is perturbed by multiple scales of random offsets. We show both theoretically and empirically that the algorithm has good convergence properties for tested imagery up to 2MP, and our CPU implementation shows speedups of 20-100 times versus *kd*-trees with PCA. Moreover, we propose a GPU implementation that is roughly 7 times faster than the CPU version for similar image sizes. Our algorithm requires very little extra memory beyond the original image, unlike previous algorithms that build auxiliary data structures to accelerate the search. Using typical settings of our algorithm's parameters, the runtime is $O(mM \log M)$ and the memory usage is $O(M)$. Although this is the same asymptotic time and memory as the most efficient tree-based acceleration techniques, the leading constants are substantially smaller.

Our generalized matching algorithm is presented in Sections 3.5-Section 3.7, and remains both simple and fast, while offering some facilities not found in previous work. For example, in computer vision, the difficulty of performing a 4D search across translations, rotations, and scales had previously motivated the use of sparse features that are invariant to some extent to these transformations. Our algorithm efficiently finds dense correspondences despite the increase in dimension, so it offers an alternative to sparse interest point methods.

3.3 Related Work

Patch-based sampling methods have become a popular tool for image and video synthesis and analysis. Applications include texture synthesis, image and video completion, summarization and retargeting, image recomposition and editing, image stitching and collages, new view synthesis, noise removal and more. We will review these applications in detail in Chapter 5. We discuss the common search techniques they use here, as well as previous work in nearest neighbor search.

Patch based methods can be largely divided into greedy and optimization methods, the latter of which can overcome artifacts and inconsistencies introduced during the greedy fill process. We also discuss these in more detail in Chapter 5. Optimization methods produce the highest quality results, as well as being easiest to use with our algorithm, so we focus on them.

The high synthesis quality of patch optimization methods comes at the expense of more search iterations, which is the clear complexity bottleneck in all of these methods. Moreover, in applications like texture synthesis, the texture exemplar is usually a small image, in other applications such as patch-based image completion, retargeting and reshuffling, the input image is typically much larger so the search problem is even more critical.

Various speedups for this search have been proposed, generally involving tree structures such as TSVQ [117], kd-trees [52, 61, 79, 119], and VP-trees [63], each of which supports both exact and approximate search (ANN). In synthesis applications, approximate search is often used in conjunction with dimensionality reduction techniques such as PCA [52, 61, 67], because ANN methods are much more time- and memory-efficient in low dimensions. Ashikhmin [3] 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. Our propagation search step is inspired by the same coherence assumption. The *k-coherence* technique [109] combines the propagation idea with a precomputation stage in which the k nearest neighbors of each patch are

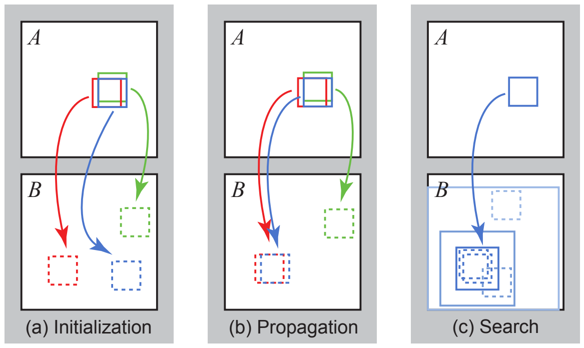


Figure 3.2: Phases of the randomized nearest neighbor algorithm: (a) patches initially have random assignments; (b) the blue patch checks above/green and left/red neighbors to see if they will improve the blue mapping, propagating good matches; (c) the patch searches randomly for improvements in concentric neighborhoods.

cached, and later searches take advantage of these precomputed sets. Although this accelerates the search phase, k-coherence still requires a full nearest-neighbor search for all pixels in the input, and has only been demonstrated in the context of texture synthesis. It assumes that the initial offsets are close enough that it suffices to search only a small number of nearest neighbors. This may be true for small pure texture inputs, but we found that for large complex images our random search phase is required to escape local minima. In this work we compare speed and memory usage of our algorithm against kd-trees with dimensionality reduction, and we show that it is at least an order of magnitude faster than the best competing combination (ANN+PCA) and uses significantly less memory. Our algorithm also provides more generality than kd-trees because it can be applied with arbitrary distance metrics, and easily modified to enable local interactions such as constrained completion.

A number of works have used our algorithm. First, there are our own applications in computer vision [10] and video summarization [9] discussed in this thesis. Other researchers have also incorporated our algorithm in research projects for finding repetitions in images [23], synthesizing hybrid images [90], enhancing images using correspondences [47], morphing between different images [96], determining where people look when comparing images [53], video denoising [70], motion estimation [16], and texture by numbers [85].

3.4 Matching Across Translation Only

The core of our system is the algorithm for computing patch correspondences. We define a *nearest-neighbor field* (NNF) as a function $\mathbf{f} : A \mapsto \mathbb{R}^2$ of nearest neighbors, defined over all possible patch coordinates (locations of patch centers) in image A, for some distance function of two patches D. Given patch coordinate $\mathbf{a}$ in image A and its corresponding nearest neighbor $\mathbf{b}$ in image B, $\mathbf{f}(\mathbf{a})$ is simply $\mathbf{b}$.[1] We refer to the values of $\mathbf{f}$ as *nearest neighbors*, and they are stored in an array whose dimensions are those of A.

This section presents a randomized algorithm for computing an approximate NNF. As a reminder, the key insights that motivate this algorithm are that we search in the space of possible coordinate offsets, that adjacent patches search cooperatively, and that even a random coordinate assignment is likely to be a good guess for many patches over a large image.

The algorithm has three main components, illustrated in Figure 3.2. Initially, the nearest-neighbor field is filled with either random assignments or some prior information. Next, an iterative update process is applied to the NNF, in which good patch nearest neighbors are propagated to adjacent pixels, followed by random search in the neighborhood of the best nearest neighbor found so far. Sections 3.4.1 and 3.4.2 describe these steps in more detail.

3.4.1 Initialization

The nearest-neighbor field can be initialized either by assigning random values to the field, or by using prior information. When initializing with random values, we use independent uniform samples across the full range of image B. In applications described in Chapter 5, we use a coarse-to-fine gradual resizing process, so we have the option to use an initial guess upscaled from the previous level in the pyramid. However, if we use only this initial guess, the algorithm can sometimes get trapped in suboptimal local minima. To retain the quality of this prior but still preserve some ability to escape from such minima, we perform a few early iterations of the algorithm using a random initialization, then merge with the upsampled initialization only at patches where D is smaller, and then perform the remaining iterations. We do this "merge and then iterate" process also during iterative algorithms such as hole filling that are very sensitive to the previous iteration, to avoid falling out of any good local minimum found on previous iterations.

3.4.2 Iteration

After initialization, we perform an iterative process of improving the NNF. Each iteration of the algorithm proceeds as follows: Nearest neighbors are examined in scan order (from left to right, top to bottom), and each undergoes *propagation* followed by *random search*. These operations are

[1] Our notation is in absolute coordinates, vs relative coordinates in Barnes et al. [8]

interleaved at the patch level: if P_j and S_j denote, respectively, propagation and random search at patch j, then we proceed in the order: $P_1, S_1, P_2, S_2, \ldots, P_n, S_n$.

Propagation. We attempt to improve $\mathbf{f}(x, y)$ using the known nearest neighbors of $\mathbf{f}(x-1, y)$ and $\mathbf{f}(x, y-1)$, assuming that the patch coordinates are likely to be the same, except for the relative translation one pixel to the right or down. For example, if there is a good mapping at $(x-1, y)$, we try to use the translation of that mapping one pixel to the right for our mapping at (x, y). Let $\mathbf{z} = (x, y)$. The new candidates for $\mathbf{f}(\mathbf{z})$ are $\mathbf{f}(\mathbf{z} - \mathbf{\Delta}_p) + \mathbf{\Delta}_p$, where $\mathbf{\Delta}_p$ takes on the values of $(1, 0)$ and $(0, 1)$. Propagation takes a downhill step if either candidate provides a smaller patch distance D.

The effect is that if (x, y) has a correct mapping and is in a coherent region R, then all of R below and to the right of (x, y) will be filled with the correct mapping. Moreover, on *even* iterations we propagate information *up and left* by examining patches in reverse scan order, and using candidates below and to the right. Propagation converges very quickly, but if used alone ends up in a local minimum. So a second set of trials employs *random search.*

Random search. A sequence of candidates is sampled from an exponential distribution, and the current nearest neighbor is improved if any of the candidates has smaller distance D. Let $\mathbf{v_0}$ be the current nearest neighbor $\mathbf{f}(\mathbf{z})$. We attempt to improve $\mathbf{f}(\mathbf{z})$ by testing a sequence of candidate mappings at an exponentially decreasing distance from $\mathbf{v}_0$:

$$\mathbf{u}_i = \mathbf{v}_0 + w\alpha^i \mathbf{R}_i \tag{3.1}$$

where $\mathbf{R}_i$ is a uniform random in $[-1, 1] \times [-1, 1]$, w is a large maximum search "radius", and α is a fixed ratio between search window sizes. We examine patches for $i = 0, 1, 2, ...$ until the current search radius $w\alpha^i$ is below 1 pixel. In our applications w is the maximum image dimension, and $\alpha = 1/2$, except where noted. Note the search window must be clamped to the bounds of B.

Halting criteria. Although different criteria for halting may be used depending on the application, in practice we have found it works well to iterate a fixed number of times. All the results shown here were computed with 4-5 iterations total, after which the NNF has almost always converged. Convergence is illustrated in Figure 3.3.

Efficiency. The efficiency of this naive approach can be improved in a few ways. In the propagation and random search phases, when attempting to improve an offset $\mathbf{f}(\mathbf{z})$ with a candidate offset $\mathbf{u}$, one can do *early termination* if a partial sum for the patch distance exceeds the current best known patch distance. Also, in the propagation stage, when using square patches of side length

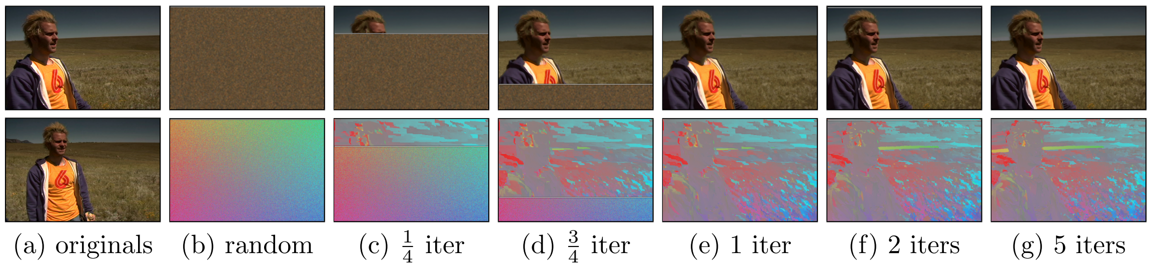


Figure 3.3: Illustration of convergence. (a) The top image is reconstructed using only patches from the bottom image. (b) above: the reconstruction by patch "voting" (each patch looks up its nearest neighbor's colors, and these are averaged for all overlapping patches), below: a random initial offset field, with magnitude visualized as saturation and angle visualized as hue. (c) 1/4 of the way through the first iteration, high-quality offsets have been propagated in the region above the current scan line (denoted with the horizontal bar). (d) 3/4 of the way through the first iteration. (e) First iteration complete. (f) Two iterations. (g) After 5 iterations, almost all patches have stopped changing.

p and an L_q norm, the change in distance can be computed incrementally in $O(p)$ rather than $O(p^2)$ time, by noting redundant terms in the summation over the overlap region. However, this incurs additional memory overhead to store the current best distances $D(f(x, y))$.

3.5 Matching Across Rotations and Scales

For some applications, such as detecting objects in the field of computer vision, it may be desirable to match patches across a range of possible rotations or scales. Without loss of generality, we compare upright unscaled patch a in image A, with patch b in image B that is rotated and scaled around its center.

To search a range of rotations $\theta \in [\theta_1, \theta_2]$ and a range of scales $s \in [s_1, s_2]$, we simply extend the search space of the original PatchMatch algorithm from (x, y) to (x, y, θ, s), extending the definition of our nearest-neighbor field to a mapping $\mathbf{f} : \mathbb{R}^2 \mapsto \mathbb{R}^4$. Here $\mathbf{f}$ is initialized by uniformly sampling from the range of possible positions, orientations and scales. In the propagation phase, adjacent patches are no longer related by a simple translation, so we must also transform the relative offsets by a Jacobian. Let $\mathbf{T}(\mathbf{f}(\mathbf{x}))$ be the full transformation defined by (x, y, θ, s): the candidates are thus $\mathbf{f}(\mathbf{x} - \boldsymbol{\Delta}_p) + \mathbf{T}'(\mathbf{f}(\mathbf{x} - \boldsymbol{\Delta}_p))\boldsymbol{\Delta}_p$. In the random search phase, we again use a window of exponentially decreasing size, only now we contract all 4 dimensions of the search around the current state.

The convergence of this approach is shown in Figure 3.4. In spite of searching over 4 dimensions instead of just one, the combination of propagation and random search successfully samples the search space and efficiently propagates good matches between patches. In contrast, with a kd-tree, it is nontrivial to search over all scales and rotations. Either all rotations and scales must be added

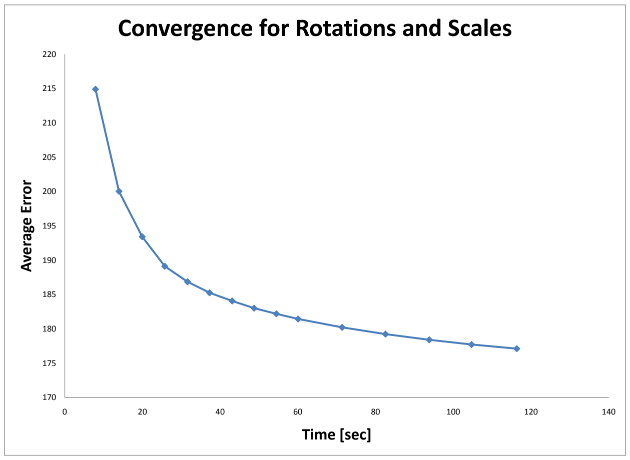


Figure 3.4: Searching across all rotations and scales. This is averaged across a dataset of 0.3 MP images. Both the time per iteration and iterations needed to converge (typically 20) are higher than for the translation only algorithm. The time per iteration is higher primarily because of the bilinear filtering used in our patch distance computation: if nearest neighbor filtering is used, each iteration is several times faster, but the patch distance is slightly higher due to aliasing.

to the tree, or else queried, incurring enormous expenses in time or memory.

As an implementation note, we note that the time per iteration in Figure 3.4 is higher than for the translation-only algorithm. This is primarily because of the bilinear filter we used for texture lookup in the rotated/scaled patch. Alternatively, nearest neighbor filtering can be used, which makes each iteration several times faster, but can introduce aliasing in the position of the best nearest neighbor. This may or may not be an issue depending on application. A third alternative is to upsample the target image, and then sample using a nearest neighbor filter. This gives high speed as well as some antialiasing. We suggest for future work that GPU algorithms could be quite efficient, due to bilinear filtering being implemented in hardware.

3.6 Matching Arbitrary Descriptors

The PatchMatch algorithm was originally implemented using the sum-of-squared differences patch distance, but places no explicit requirements on the distance function. The only implicit assumption is that patches with close spatial proximity should also be more likely to have similar best-nearest-neighbors, so that PatchMatch can be effective at propagating good nearest neighbors and finding new ones. This turns out to be true for a variety of descriptors and distance functions. In fact, the algorithm can converge even more quickly when using large-area feature descriptors than it does with small image patches, because they tend to vary relatively slowly over the image. In general, the "distance function" can actually be any algorithm that supplies a total ordering. It does not need to obey the properties of the usual mathematical metric distance function. For example symmetry can be broken: the "distance" from the source patch A to the target patch B could change if the two images are switched. Or the triangle inequality could not be obeyed. The matching can even be performed between entirely different images — the rate of convergence depends only on the size of coherent matching regions. Thus, our matching is quite flexible. In Chapter 7, we explore several examples in computer vision, such as patches that compensate for lighting changes, and matching SIFT descriptors [73].

3.7 Matching k-Nearest Neighbors

For certain applications such as removing noise by averaging similar patches, we may wish to compute more than a single nearest neighbor at every (x, y) position. This can be done by collecting k nearest neighbors for each patch. Thus the NNF $\mathbf{f}_k$ is a multi-valued map, with k values. The NNF thus can be stored as an array with size A and k channels. There are many possible modifications of PatchMatch to compute the k-NN. We have compared the efficiency of several of these against a standard approach: dimensionality reduction with PCA, followed by construction of a kd-tree [78] with all patches of image B projected onto the PCA basis, then an independent ϵ-nearest neighbor lookup in the kd-tree for each patch of image A projected onto the same basis.

Since each of these algorithms can be tuned for either greater accuracy or greater speed, we evaluated each across a range of settings. For PatchMatch, we simply computed additional iterations, and for kd-trees we adjusted the ϵ and PCA dimension parameters. The relative efficiency of these algorithms is plotted in Figure 3.5. We also compare with FLANN [79], a package that includes kd-tree, k-means tree, a hybrid algorithm, and a large number of parameters that can be tuned for performance.

Heap algorithm. In the most straightforward variant, we associate k nearest neighbors with each patch position. During propagation, we improve the nearest neighbors at the current position by exhaustively testing each of the k nearest neighbors to the left or above (or below or right on even iterations). The new candidates are $\mathbf{f}_i(\mathbf{x} - \mathbf{\Delta}_p) + \mathbf{\Delta}_p$, where $\mathbf{\Delta}_p$ takes on the values $(1, 0)$ and $(0, 1)$, and $i = 1 \ldots k$. If any candidate is closer than the worst candidate currently stored at $\mathbf{x}$, that worst candidate is replaced with the candidate from the adjacent patch. This can be done efficiently with a max-heap, where the heap stores the patch distance D. The random search phase works similarly:

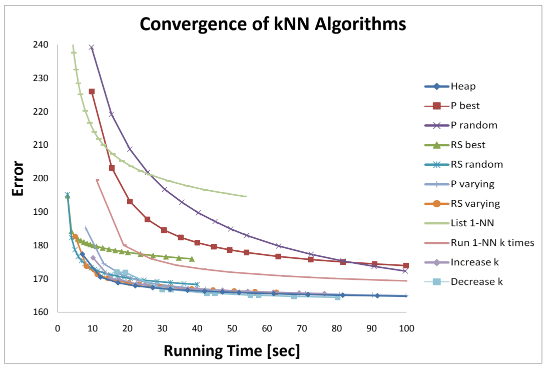
n samples are taken around each of the k nearest neighbors, giving nk samples total. The worst element of the heap is evicted if the candidate's distance is better. When examining candidates, we also construct a hash table to quickly identify candidates already in our k list, to prevent duplicate entries. We store the k nearest neighbors in heap order as a rectangular array $\mathbf{f}$. To reduce memory usage we rebuild the hash table as needed when we visit each patch.

Heap algorithm with sparse sampling of k. In our analysis of the heap algorithm, we found that the k neighbors often have tightly clustered offsets, because natural images feature a great deal of coherency. Therefore, one might wonder if the exhaustive search through all k candidates in the propagation and random search phase is really necessary, or if we can sample the k candidates more sparsely. For propagation, one can examine only the adjacent offset with smallest distance (denoted "P best" in Figure 3.5) or choose an element at random as a candidate for propagation (denoted "P random"). Likewise, in random search, one can randomly sample around the offset with smallest distance (denoted "RS best") or sample around a randomly chosen offset (denoted "RS random"). Finally, one could run propagation or random search on the top m offsets, where m is randomly chosen uniformly between 1 and k (denoted "P varying" and "RS varying"). Although they reduce the number of operations for each iteration, these strategies all proved to converge more slowly than the naïve heap algorithm, as shown in Figure 3.5.

Use the 1-NN algorithm to find k-NN. Another observation is that many candidates are considered over the course of the original 1-NN PatchMatch algorithm. Therefore, in another strategy we retain all candidate offsets sampled by the algorithm for each pixel, and then partially sort the list to find the top k (denoted "List 1-NN"). Similarly, we could run the 1-NN algorithm k times, with each run constrained so that offsets cannot be equal to any of the previously chosen offsets (denoted "Run 1-NN k times"). Like the sparse heap strategies, it is fast to compute an iteration but convergence over time remains slower than the original heap algorithm.

Changing k **during iterations**. One can start on iteration 1 with a number of nearest neighbors k_0, and after half of the iterations are completed, increase or decrease this to the final desired number of nearest neighbors k, either dropping the worst elements of the heap, or adding uniform random elements as needed. We tried increasing from a small number of nearest neighbors $k_0 = k/2$ (denoted "Increase k"), and decreasing from a large number of nearest neighbors $k_0 = 2k$ (denoted "Decrease k"). Again these algorithms are slower than the simple heap algorithm.

Some of these algorithms complete single iterations faster than the basic heap algorithm described above, but convergence is slower as they propagate less information within an iteration. In general, the original heap algorithm is a good choice over a wide range of the speed/quality curve.



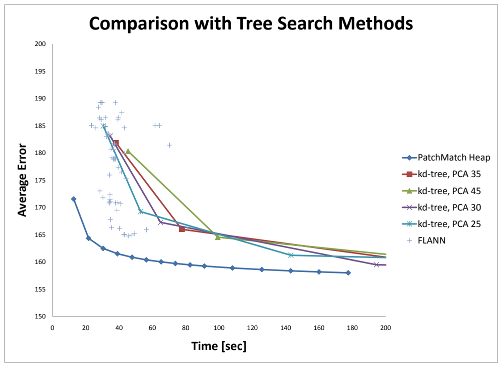


Figure 3.5: *Left:* Performance of k-PatchMatch variants, with $k = 16$ nearest neighbors, averaged over all images in Figure 7.2, resized to 0.2MP, and matched against themselves. Error is average L^2 patch distance over all k matches, for 7x7 patches. Points on each curve represent progress after each iteration. *Right:* Comparison with kd-tree and FLANN, at 0.3 MP, averaged over the same dataset. Again the matching is for $k = 16$ nearest neighbors. Our algorithm outperforms the tree methods.

We find the basic heap algorithm outperforms kd-tree over a wide range of k and image sizes: for example, our algorithm is several times faster than kd-tree, for $k = 16$ and input images of 0.1 to 1.0MP. In our comparisons to the kd-tree implementation of Mount and Arya [78] and FLANN [79], we gave the competition the benefit of the doubt by tuning all possible parameters, while adjusting only the number of iterations for our heap algorithm. FLANN offers several algorithms, so we sampled a large range of algorithmic options and parameters, indicated by the + marks in Figure 3.5. FLANN can also automatically optimize parameters, but we found the resulting performance always lies within the convex hull of our point-sampling. In both cases, this extensive parameter-tuning resulted in performance that approached – but never exceeded – our heap algorithm. Thus, we propose that the general k-PatchMatch heap algorithm is a better choice for a wide class of problems requiring image patch correspondence. With additional optimization of our algorithm, the performance gap might be even greater.

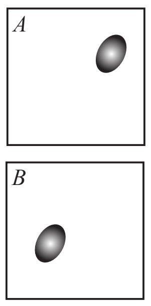
3.8 Analysis of Convergence

In this section, we analyze the convergence of our 1-NN algorithm for matching across translations only, for both synthetic and real examples. The translation only algorithm is easier to analyze than the generalizations described in Sections 3.5-Section 3.7.

3.8.1 Analysis for a Synthetic Example

Our algorithm converges to the exact NNF in the limit. Here we offer a theoretical analysis for this convergence. Our analysis here applies only to the translation-only single-nearest neighbor algorithm. We show that it converges most rapidly in the first few iterations with high probability. Moreover, we show that in the common case where only approximate patch matches are required, the algorithm converges even faster. Thus our algorithm is best employed as an approximation algorithm, by limiting computation to a small number of iterations.

We start by analyzing the convergence to the exact nearest-neighbor field and then extend this analysis to the more useful case of convergence to an approximate solution. Assume A and B have equal size (M pixels) and that random initialization is used. Although the odds of any one location being assigned the best offset in this initial guess are vanishingly small ($1/M$), the odds of at least one offset being correctly assigned are quite good ($1-(1-1/M)^M)$) or approximately $1-1/e$ for large M. Because the random search is quite dense in small local regions we can also consider a "correct" assignment to be any assignment within a small neighborhood of size C pixels around the correct offset. Such offsets will be corrected in about one iteration of random search. The odds that at least one offset is assigned in such a neighborhood are excellent: $(1-(1-C/M)^M)$ or for large M, $1-exp(-C)$.



Now we consider a challenging synthetic test case for our algorithm: a distinctive region R of size m pixels lies at two different locations in an otherwise uniform pair of images A and B (shown inset). This image is a hard case because the background offers no information about where the offsets for the distinctive region may be found. Patches in the uniform background can match a large number of other identical patches, which are found by random guesses in one iteration with very high probability, so we consider convergence only for the distinct region R. If any one offset in the distinct region R is within the neighborhood C of the correct offset, then we assume that after a small number of iterations, due to the density of random search in small local regions (mentioned previously), that all of R will be correct via propagation (for notational simplicity assume this is instantaneous). Now suppose R has not yet converged. Consider the random searches performed by our algorithm at the maximum scale w. The random search iterations at scale w independently sample the image B, and the probability p that any of these samples lands within the neighborhood C of the correct offset is

$$p = 1 - (1 - C/M)^m \tag{3.2}$$

Before doing any iterations, the probability of convergence is p. The probability that we did not converge on iterations $0, 1, ..., t-1$ and converge on iteration t is $p(1-p)^t$. The probabilities thus form a geometric distribution, and the expected time of convergence is $\langle t \rangle = 1/p - 1$. To simplify, let the relative feature size be $\gamma = m/M$, then take the limit as resolution M becomes large:

$$\langle t \rangle = [1 - (1 - C/M)^{\gamma M}]^{-1} - 1 \tag{3.3}$$

$$\lim_{M\to\infty} \langle t \rangle = [1 - \exp(-C\gamma)]^{-1} - 1 \tag{3.4}$$

By Taylor expansion for small γ, $\langle t \rangle = (C\gamma)^{-1} - \frac{1}{2} = M/(Cm) - \frac{1}{2}$. That is, our expected number of iterations to convergence remains constant for large image resolutions and a small feature size m relative to image resolution M.

We performed simulations for images of resolution M from 0.1 to 2 Megapixels that confirm this model. For example, we find that for a $m = 20^2$ region the algorithm converges with very high probability after 5 iterations for a $M = 2000^2$ image.

The above test case is hard but not the worst one for exact matching. The worst case for exact matching is when image B consists of a highly repetitive texture with many distractors similar to the distinct feature in A. The offset might then get "trapped" by one of the distractors, and the effective neighborhood region size C might be decreased to 1 (i.e., only the exact match can pull the solution out of the distractor during random search). However in practice, for many image analysis and synthesis applications such as the ones we show in this thesis, finding an approximate match (in terms of patch similarity) will not cause any noticeable difference. The chances of finding a successful approximate match are actually higher when many similar distractors are present, since each distractor is itself an approximate match. If we assume there are Q distractors in image B that are similar to the exact match up to some small threshold, where each distractor has approximately the same neighborhood region C, then following the above analysis the expected number of iterations for convergence is reduced to $M/(QCm) - 0.5$.

3.8.2 Analysis for Real-World Images

Here we analyze the approximations made by our algorithm on real-world images. We investigate performance of the translation only algorithm finding a single nearest neighbor. For performance plots of the general algorithm, see Figure 3.4 and Figure 3.5. To assess how our algorithm addresses different degrees of visual similarity between the input and output images, we performed error analysis on datasets consisting of pairs of images spanning a broad range of visual similarities.

	Time [s]		Memory [MB]	
Megapixels	Ours	kd-tree	Ours	kd-tree
0.1	0.68	15.2	1.7	33.9
0.2	1.54	37.2	3.4	68.9
0.35	2.65	87.7	5.6	118.3

Table 3.1: Running time and memory comparison for the input shown in Figure 3.3. We compare our algorithm against a method commonly used for patch-based search: kd-tree with approximate nearest neighbor matching. Our algorithm uses $n = 5$ iterations. The parameters for kd-tree have been adjusted to provide equal mean error to our algorithm.

These included inputs and outputs of our image editing operations (very similar), stereo pairs[2] and consecutive video frames (somewhat similar), images from the same class in the Caltech-256 dataset[3] (less similar) and pairs of unrelated images. Some of these were also analyzed at multiple resolutions (0.1 to 0.35 MP) and patch sizes (4x4 to 14x14). Our algorithm and ANN+PCA kd-tree were both run on each pair, and compared to ground truth (computed by exact NN). Note that because precomputation time is significant for our applications, we use a single PCA projection to reduce the dimensionality of the input data, unlike Kumar et al. [63], who compute eigenvectors for different PCA projections at each node of the kd-tree. Because each algorithm has tunable parameters, we also varied these parameters to obtain a range of approximation errors.

We quantify the error for each dataset as the mean and 95th percentile of the per-patch difference between the algorithm's RMS patch distance and the ground truth RMS patch distance. For 5 iterations of our algorithm, we find that mean errors are between 0.2 and 0.5 gray levels for similar images, and between 0.6 and 1.5 gray levels for dissimilar images (out of 256 possible gray levels). At the 95th percentile, errors are from 0.5 to 2.5 gray levels for similar images, and 0.9 to 6.0 gray levels for dissimilar images.

Our algorithm is both substantially faster than kd-tree and uses substantially less memory over a wide range of parameter settings. For the 7x7 patch sizes used for most results in the thesis, we find our algorithm is typically 20x to 100x faster, and uses about 20x less memory than kd-tree, regardless of resolution. Table 3.1 shows a comparison of average time and memory use for our algorithm vs. ANN kd-trees for a typical input: the pairs shown in Figure 3.3. The rest of our datasets give similar results. To fairly compare running time, we adjusted ANN kd-tree parameters to obtain a mean approximation error equal to our algorithm after 5 iterations.

The errors and speedups obtained are a function of the patch size and image resolution. For smaller patches, we obtain smaller speedups (7x to 35x for 4x4 patches), and our algorithm has

[2]`http://vision.middlebury.edu/stereo/data/scenes2006/`
[3]`http://www.vision.caltech.edu/Image_Datasets/Caltech256/`

higher error values. Conversely, larger patches give higher speedups (300 times or more for 14x14 patches) and lower error values. Speedups are lower at smaller resolutions, but level off at higher resolutions.

Recent work [63] indicates that vp-trees are more efficient than kd-trees for exact nearest-neighbor searches. We found this to be the case on our datasets. However, exact matching is far too slow for our applications. When doing approximate matching with PCA, we found that vp-trees are slower than kd-trees for equivalent error values, so we omitted them from our comparison above.

3.9 Discussion

The nature of our algorithm bears some superficial similarity to LBP and Graph Cuts algorithms often used to solve Markov Random Fields on an image grid [107]. However, there are fundamental differences: Our algorithm is designed to optimize an energy function without any neighborhood term. MRFs often use such a neighborhood term to regularize optimizations with noisy or missing data, based on coherency in an underlying generative model. In contrast, our algorithm has no explicit generative model, but uses coherency in the *data* to prune the search for likely solutions to a simpler parallel search problem. Because our search algorithm finds these coherent regions in early iterations, our matches err toward coherence. Thus, even though coherency is not explicitly enforced, our approach is sufficient for many practical synthesis applications, as shown in Chapters 5 and 6. Moreover, our algorithm avoids the expensive computations of the joint patch compatibility term and inference/optimization algorithms.

To summarize, this chapter has presented the core matching algorithm in the context of matching densely sampled descriptors. We have analyzed convergence both empirically and theoretically, and found that our algorithm performs at least an order of magnitude faster than previous work, across a wide range of parameter settings. We next discuss implementation details for parallel architectures and large collections of images.

Chapter 4

Parallelism and Search Strategies

In this chapter, we discuss details of implementing our algorithm on parallel architectures such as GPU and multicore CPU, and "PatchWeb" which is an algorithm similar to PatchMatch that addresses the problems of search and scalability in large image collections. We also introduce two new search operations: *enrichment* and *binning*, which help constrain the search process, particularly for large image datasets.

4.1 Parallelism

We have implemented parallel variants of our matching algorithm on both the GPU and CPU. The random search operation is trivially parallelizable, so the main challenge is to parallelize the propagation operation. Here we discuss the different parallel algorithms.

We implemented a fully parallel GPU algorithm in the languages CUDA and GLSL. To keep the operations more coherent, we alternate between iterations of random search and propagation, where each stage addresses the entire offset field in parallel. Although propagation is inherently a serial operation, we adapt the jump flood scheme of Rong and Tan [91] to perform propagation over several iterations. Whereas our CPU version is capable of propagating information all the way across a scanline, we find that in practice long propagations are not needed, and a maximum jump distance of 8 suffices. We also use only 4 neighbors at each jump distance, rather than the 8 neighbors proposed by Rong and Tan. Both GPU implementations showed similar performance. With similar approximation accuracy, the GPU algorithm is roughly 7x faster than the single core CPU algorithm, on a GeForce 8800 GTS card.

We implemented two different versions of the parallel algorithm on the CPU. First, we

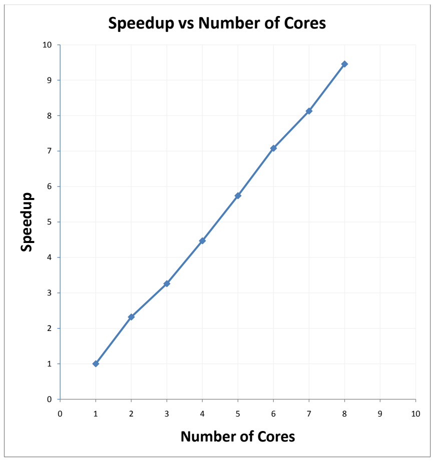


Figure 4.1: Approximately linear speedup of the tiled parallel algorithm with increasing number of cores. We attribute deviations from linearity to cache effects.

implemented the jump flood algorithm described above on the CPU. Its errors are comparable with the original CPU algorithm, and although it is roughly 2.5x slower than the CPU algorithm on a single core, the run-time scales linearly with the number of cores.

We therefore tried a second approach and parallelized the ordinary CPU algorithm by dividing the NNF into horizontal tiles, and handling each tile on a separate core. Because the tiles are handled in parallel, information can propagate vertically the entire length of a tile in a single iteration. To ensure information has a chance to propagate all the way up and down the image, we synchronize using a critical section after each iteration. To prevent resource conflicts due to propagation between abutting tiles, we write back the nearest neighbors in the last row of the tile only after synchronization. (Alternately, to avoid this synchronization issue, on an n core machine, one can split the input into $2n$ tiles vertically, and process even tiles in parallel, followed by odd tiles in parallel.)

For the tiled algorithm, we observe an approximately linear speed-up, on our 8 core test machine, as shown in Figure 4.1.

For future work, we believe more efficient GPU implementations can be made, by finding alternatives to the jump flood propagation. We suggest several possible improvements. First, in the original serial algorithm, each diagonal of the image is serially dependent only on previous diagonals, so it may be possible to serially walk through the diagonals, processing each in parallel, to do propagation. (Because propagation only needs to work for a limited distance, we could process every nth diagonal at the same time, e.g. $n = 8$). Alternatively, this could be done on entire columns or rows of the image, for better load balancing, but using different propagation patterns (e.g. when processing columns from left to right, and propagating information right, propagate by using one's neighbors to the left, above-left and below-left). Finally, the tiled multicore algorithm could be adapted to the GPU.

4.2 Search Operations: Enrichment and Binning

Here we introduce two new search operations that can be used generally, but are particularly important when searching large image collections. This is because they help constrain the purely random exploration done by the "random search" operation.

4.2.1 Enrichment

The first search operation we call *enrichment*. The propagation step of PatchMatch propagates good matches across the spatial dimensions of the image. However, in special cases we can also consider propagating matches across the space of patches themselves: For example, when matching an image A to itself – as in non-local-means denoising (Section 7.3) – many of a patch's k nearest neighbors will have the original patch and some of the other $k-1$ patches in their own k-NN list.

We define enrichment as the propagation of good matches from a patch to its k-NN, or vice versa. We call this operation enrichment because it takes a nearest neighbor field and improves it by considering a "richer" set of potentially good candidate matches than propagation or random search alone. From a graph-theoretic viewpoint, we can view ordinary propagation as moving good matches along a rectangular lattice whose nodes are patch centers (pixels), whereas enrichment moves good matches along a graph where every node is connected to its k-NN. We introduce two types of enrichment, for the special case of matching patches in A to other patches in A:

Forward enrichment uses compositions of the function $\mathbf{f}$ with itself to produce candidates for improving the nearest neighbor field. The canonical case of forward enrichment is $\mathbf{f}^2$. That is, if $\mathbf{f}$ is a NNF with k neighbors, we construct the NNF $\mathbf{f}^2$ by looking at all of our nearest neighbor's nearest neighbors: there are k^2 of these. The candidates in $\mathbf{f}$ and $\mathbf{f}^2$ are compared and the best k overall are used as an improved NNF $\mathbf{f}'$. If min() denotes taking the top k matches, then we have: $\mathbf{f}' = \min(\mathbf{f}, \mathbf{f}^2)$. Other variants of forward enrichment can be considered, such as $\mathbf{f}^3$, or $\mathbf{f}^4$, and so forth. Also, to improve asymptotic running time from $O(k^2)$ to $O(k)$, we could randomly sample k elements from $\mathbf{f}^2$, or take only the top $\sqrt{k}$ elements of $\mathbf{f}$ before computing $\mathbf{f}^2$. For moderate values of k, such as $k = 16$, the canonical $\mathbf{f}^2$ algorithm converges faster than these alternatives.

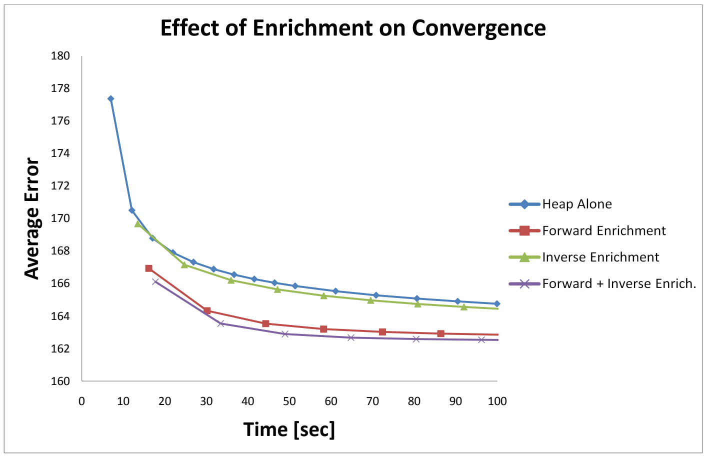


Figure 4.2: Comparison of the heap algorithm with and without enrichment. The slowest algorithm is the heap, and the fastest algorithm uses both forward and inverse enrichment. The plot here represents an average over a dataset of 30 images at 0.2 megapixels and $k = 16$ neighbors.

Similarly, **inverse enrichment** walks the nearest-neighbor pointers backwards to produce candidates for improving the NNF. The canonical algorithm here is $\mathbf{f}^{-1}$. That is, compute the multi-valued inverse $\mathbf{f}^{-1}$ of function $\mathbf{f}$. Note that $\mathbf{f}^{-1}(a)$ may have zero values if no patches point to patch a, or more than k values if many patches point to a. We store $\mathbf{f}^{-1}$ by using a list of varying length at each position. Again, to improve the current NNF, we rank our current k best neighbors and all neighbors in $\mathbf{f}^{-1}$, producing an improved NNF $\mathbf{f}''$: $\mathbf{f}'' = \min(\mathbf{f}, \mathbf{f}^{-1})$. We tried also other inverse powers ($\mathbf{f}^{-2}$, $\mathbf{f}^{-3}$, and so forth) but found the canonical $\mathbf{f}^{-1}$ algorithm to be the fastest. Note that in most cases the distance function is symmetric, so patch distances do not need to be computed for $\mathbf{f}^{-1}$. Finally we can concatenate inverse and forward enrichment, and we found that $\mathbf{f}^{-1}$ followed by $\mathbf{f}^{2}$ is fastest overall. The performance of these algorithms is compared in Figure 4.2.

In the case of matching different images A and B, inverse enrichment can be trivially done. Forward enrichment can be applied by computing nearest neighbor mappings in both directions; we leave this investigation for future work.

4.2.2 Binning

The second search operation we call *binning*. This operation can be thought of as being analogous to inserting and querying the patch appearance in a very low dimensional kd-tree. We do not aim to distinguish patch appearance accurately, but instead capture a few of the first color and gradient components grossly.

In this operation, we first determine for each patch, which "bin" does it fall into? This is done as a precomputation. We determine the bin by doing a PCA dimension reduction of patch appearance to a very low number of dimensions k. Then each of these k dimensions is quantized by partitioning it into p intervals each of equal size. Thus there are a total of p^k bins, forming a uniform, non-adaptive multidimensional grid. We typically choose small p and k so the total number of bins is in on the order of ten thousand, such as $p = 9, k = 4$. Also the PCA need be approximated only very roughly, so the patches can be sparsely sampled when computing PCA. For each bin, we list in a vector all of the patch coordinates that fall into that bin. (In some applications it is important to reduce memory consumption, so every nth patch that falls into the bin can be stored instead).

After this precomputation, we run the PatchMatch algorithm with the bin search operation added. This operation proposes a new candidate nearest neighbor with potentially lower patch distance. It does this by simply selecting a random patch from the bin the current query patch falls into.

In the context of searching large image collections, binning accelerates convergence by giving a higher chance of randomly hitting a good target. Random search alone would sample randomly from the entire collection, and would have a low chance of hitting a "good target", whereas binning will propose patches of similar gross appearance. For the 100 image input in Figure 4.4, binning accelerates the algorithm by a factor of 2-4 for equal error values. The next section describes the full algorithm for matching in large image collections, which incorporates binning and enrichment, as well as memory management and scalable algorithms for large numbers of images.

4.3 PatchWeb: Matching in Large Image Collections

This section presents PatchWeb, an algorithm which adapts PatchMatch for the special constraints of search in large image collections.

4.3.1 Introduction

Recently, vast collections of digital photographs have been published online and many users have built up sizable personal photo collections. Consequently, an increasing number of research projects have focused on using image collections as an input. Results have been demonstrated for removing unwanted objects in photos [50], enchancing poorly-taken photos [31], increasing image resolution [46], among many other works in graphics.

Many of these methods have in common that it is necessary to find correspondences between a query photograph and a large existing photo collection. In previous works, to obtain practical running times, features computed densely at every single pixel have been avoided. Instead, algorithms have tended to reduce the search space, by reducing the amount of input data to a small tractable amount. This has been done with sparse features [40, 82], global features that describe an entire scene [50], or other techniques for reducing dimensionality. However, these sparse features can reduce the performance of the algorithm.

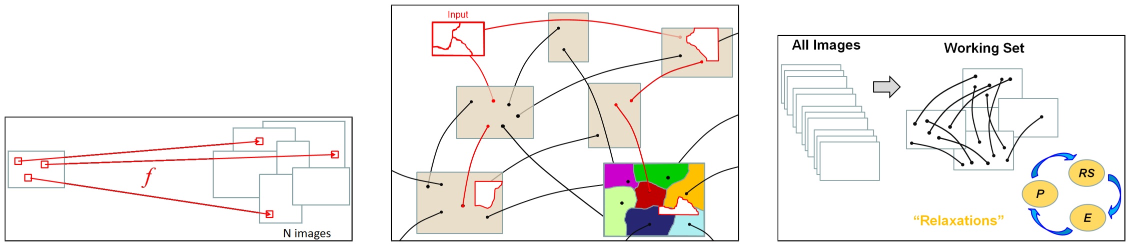
Several further complications result from the large amounts of data input to these algorithms. First, in some cases such as removing objects from photos [50], millions of images must be used to obtain high-quality results, so any search technique must be able to scale into millions of images. Second, for algorithms such as increasing image resolution [43], highest quality results may be obtained by considering all possible dense correspondences. For a megapixel query image and a million image database, a brute force algorithm to find the best correspondence at each pixel must already examine 10^{18} correspondences, thus requiring an intractable amount of processing time. Because of this, previous work artifically restricted the set of possible matching images, reducing the quality of results. Finally, many existing methods have poor scaling properties, whereby finding correspondences between a query image and a collection of n photos takes $O(n)$ time. As the dataset grows larger, this linear scaling tends to inhibit other desirable properties such as user interactivity.

We propose an algorithm that attempts to overcome a number of these shortcomings in existing search methods. Our algorithm finds approximate nearest neighbor matches between a query image and a large image collection. It is the first algorithm to our knowledge to attempt to solve the *dense matching problem*: that is, try to find the best correspondence(s) at every single pixel, which independently match the entire dataset. It is *scalable* to large numbers of images, meaning it can be run on a single computer or a cluster using bounded time or memory resources.

Our algorithm is inspired by the fast matching algorithm, PatchMatch, for finding correspondences between $n = 2$ input images. We call our algorithm "PatchWeb" because it builds a graph structure or "web" between the images in the collection. After a preprocess that takes time linear in the size n of the image collection, query operations can be run efficiently, returning dense nearest neighbor matches between the query image and the existing photo collection. Our algorithm is depicted in Figure 4.3.

Specifically, our algorithm solves the problem of finding for each patch of query image or region A, given an image collection $B_1, \ldots, B_n$, what is the closest patch in the entire image collection, under a patch distance metric such as L^p norm.

Our algorithm works by storing for each image B_i in the collection a *Nearest-Neighbor Field*



(a) Query image collection (b) Precompute web data structure (c) How web is built

Figure 4.3: Illustration of PatchWeb algorithm. (a) Our goal is to quickly find dense nearest-neighbor matches between a query image and an image collection. (b) To do this we do a precomputation on the image collection, and construct a graph that gives nearest neighbor matches among the collection. This graph is the web: it stores a nearest neighbor field giving each patch a neighbor elsewhere in the web. (c) The algorithm for building the web. From the entire image collection, we randomly select a working set, then we run relaxations to build nearest neighbor links among the working set, using operators such as random sampling, propagation, and enrichment.

(NNF) mapping every patch in B_i to the most similar patch in the rest of the image collection. We use similar techniques as the PatchMatch algorithm: starting with a poor initial guess for the field, we improve the field by proposing better candidate correspondences. These better candidates are proposed by a sequence of operators. The correspondences among the image collection B_i are improved until convergence criteria are met.

In most cases the image collection B cannot fit in memory. Therefore, we introduce a *working set*, that is, the images which can currently be read, which is sampled from the set of all images in the image collection. On a cluster, each processor will have its own working set. Each processor proposes improved correspondences for as long as desired.

We call the overall algorithm relax(), because given a collection of images, it improves the nearest neighbor matches between them using relaxation. The query algorithm is done by simply adding the query image A to the working set and running relax() with the NNFs for the existing image collection in read-only mode, so that the existing web will not have links pointing to the query image.

Because the dense matching problem was considered difficult and there is no previous work known to us in this area, we compare our PatchWeb algorithm to a few competing algorithms such as PatchMatch on all pairs of images, and find it exhibits better scaling properties. We do not claim that our algorithm is optimal in time or memory usage, only that it is the first to solve the dense matching problem in a way that is scalable.

The contributions of this section include an algorithm for finding dense correspondences between millions of images; and useful properties such as efficient query and scalability to cluster computing, as well as working in bounded time and memory constraints.

4.3.2 Related Work

We first will review distributed data structures, and low-level descriptors commonly used when analyzing collections of images. Next we will discuss previous work in various potential application domains, such as object/scene recognition, image-based rendering, image and resolution enhancement, and compression, and how these relate to our proposed algorithm.

Distributed data structures. Our dense matching problem will require a very large amount of storage – for the case of one million one megapixel images, already a trillion patches are involved, between which correspondences must be found, necessitating an even higher amount of computation. Therefore, distributed storage and computation are appropriate. Two distributed data structures are distributed *k*d-trees [81] and distributed LSH [48]. The distributed *k*d-tree works as a forest of *k*d-trees: each compute node handles one sub-part of the tree, so that all nodes represent a single larger *k*d-tree. There are scalability and engineering difficulties with using a *k*d-tree, because distributed kd-trees are hard to implement, and for a large enough number of images, especially if descriptors are not very compact, the trees may not fit in memory and parts must be serialized to disk. While in principle a distributed tree with the right systems choices might compete with our algorithm, our algorithm is simple to implement and scales easily from a single computer to a cluster without requiring architecture changes. Distributed LSH [48] was implemented in an unstructured peer-to-peer network, assigning hash buckets to peers in a Chord-style overlay network. A similar implementation of LSH could be run on a compute cluster, however the same concerns arise concerning difficulty of engineering and memory taken by hashes of e.g. trillions of patches into a number of different bases. In our algorithm, we use similar ideas as LSH by mapping patches into a low-dimensional set of bins, but avoid the memory problem by doing this only for the active working set.

Descriptors and visual words. Many papers which do recognition or image-based rendering need to identify similar-appearing scenes or objects. To describe scenes, a common descriptor is GIST [84], which distinguishes general classes of scenes (mountains, trees, streets, etc) by using a single compact spectral descriptor that distinguishes perceptual dimensions. Later matching of pyramids of histograms [66] was shown to improve on scene recognition, achieving then state-of-the-art recognition rates. To describe objects, often SIFT features [73] computed densely or at sparse interest points are used. For object and scene recognition, often visual words, and the bag of words model [40, 101] may be used, which computes quantized histograms of lower-level features such as SIFT descriptors or patches.

Object/scene recognition. Many works have demonstrated recognition in large image collections. These include scalable recognition of large numbers of images using quantized codebooks of visual words [102] or trees of visual words [82]. In the latter work, as in our work, compactness of the data structure is very important for efficiency, so they represent image patches compactly as one or two integers giving a path through a pre-computed tree. However our work uses densely computed features, so even more data must be stored in our data structure. Use of small 32x32 color thumbnails [110] with 80 million images and only naive global alignment algorithms has shown high recognition performance, comparable to the state of the art for class-specific detectors. Because high performance is obtained only with a very large number of images, this motivates us to allow our algorithm to also scale. Finally, motivating our algorithm, it has been shown that densely sampled SIFT descriptors and only a simple nearest-neighbor classification scheme can give then state-of-the-art performance for image classification [14].

Image completion, enhancement, editing. It was shown [50] that large undesired objects can be removed from a photograph by using millions of images and GIST descriptors, if there is a single, correctly aligned "scene level" match elsewhere in the dataset. Even using simple global descriptors, this takes an hour to remove an object on one CPU, or 5 minutes when parallelized across 15 CPUs, limiting interactivity. When an object to be removed is smaller or incorrectly aligned, more local matching as can be obtained by our algorithm might be appropriate. Restoration of incorrectly exposed photographs has been demonstrated by matching to a large image collection using visual words [31]. Similarly, computer graphics renderings can be enhanced in realism [55] by using a large corpus of real photographs. The Sketch2Photo system [22] allows a user to turn a sketch into a photorealistic scene by automatically downloading thousands of candidate images and using heuristics to stitch them together. This takes 20 or more minutes to generate the final result. As future work, we might try applying our algorithm to paint by numbers to gain faster user interaction time.

Image-based rendering of large image collections. Photo tourism [105] showed that a collection of images around a tourist landmark could be aligned automatically into a sparse 3D model. Later, similar matching and reconstruction [1, 88] has been done on a larger scale, up to a million photograph automatic reconstruction of Rome. The infinite images work [57] demonstrated that using 6 million images, seamless transitions of a virtual camera could be created, navigating left/right, forward, or rotating. We do not work on image-based rendering, but candidate correspondences could be gathered from our PatchWeb structure for such reconstructions.

Enhancing image resolution. Given a low resolution image, resolution can be increased by using a training set of low-resolution example photographs and their high-resolution counterparts, and finding similar low-resolution patches [43]. In the previous work this required that the training set be of the same class of images, and contain only low tens of photographs, because otherwise the dense nearest neighbor search becomes slow. We hope to do similar super-resolution using millions of photographs, and detect image class automatically if needed. More recently, it was shown that image super-resolution could be done within a single photograph, by matching across scales [44]. Likewise, hallucination of plausible high resolution detail can be done given an appropriate image collection [38, 46], allowing increases in resolution of say 10-100x where classical super-resolution algorithms would fail.

Finally, using a single image, or a collection of images, image compression can be done [116] by finding repeated elements that look similar, and replacing them all with a single replicated element. We hope that for these and other applications, our webs can help identify repeated regions even in very large image collections.

4.3.3 PatchWeb Algorithm

In this section we explain our matching algorithm (illustrated in Figure 4.3). Recall that our problem is given a query image A and an image collection $B_1, \ldots, B_n$, to quickly answer queries for each patch in A, which is the nearest neighbor patch in collection B. We do this by first building a web: we precompute a graph of nearest neighbors among the image collection B. Then we query the image A against the web, computing nearest neighbors that map image A into the collection B. For efficiency we do all matching approximately, and note that without loss of generality any densely sampled descriptor can be used in place of patches. We denote the generalized patch distance function as $D()$: the best correspondence for a given pixel is that which minimizes this distance. For example depending on the application, $D()$ might compare L^2 norm between patches, or L^2 norm between SIFT features.

Our algorithm uses similar ideas as the PatchMatch algorithm [8], which computes dense nearest neighbor matchings between two images. The PatchMatch algorithm works by first initializing with random correspondences, then improving correspondences by proposing new candidate correspondences, and taking the better of the current and new candidate correspondences, then iterating until convergence. We take a similar approach. We fit as many images as we can into memory, find good correspondences between the patches of these images, then store some of these correspondences to disk and load some new images into memory which do not yet have good correspondences. At a very high level, our algorithm can be thought of as automatically finding clusters of similar patches by an iterative random process.

We associate with each image B_i in the collection a *Nearest-Neighbor Field* (NNF) which stores for each pixel coordinate, the best matching patch in the rest of the image collection. Specifically, the NNF is defined as $f_i : \mathbb{R}^2 \mapsto \mathbb{R}^3$ so $f_i(x, y) = (x', y', j)$, where (x, y) is the coordinate in image B_i, and (x', y') is the nearest neighbor in some other image B_j. We also store the patch distance D_{patch} associated with this correspondence. Note that D_{patch} must be stored because in large image collections the target image B_j may not be in memory. By packing the target coordinates and patch distance into a 64-bit integer, we can compactly store the NNF as a single channel 64-bit image. We allocate 12 bits each to x and y coordinates, 16 bits to image index, and 24 bits to patch distance, giving a maximum image resolution of 4096 pixels and maximum dataset size of 64k images. In the future we may increase the number of bits to address more images. We store each NNF alongside its image B_i on disk, and also ZIP compress the NNF to minimize storage and IO requirements.

4.3.4 Building the Web via Relaxation

We first initialize the web B with every image containing a NNF mapping to $(-1, -1, -1)$, with patch distance ∞. This is a dummy correspondence indicating infinite patch distance, so it will be replaced by any proposed correspondence.

Next we improve the correspondences in the web via relaxation. We repeatedly sample a working set of images from the collection of all images B. We choose the working set size as large as possible so that working set still fits in main memory. All images in the working set are loaded from disk, as are the associated NNFs.

We loop through all coordinates (x, y) of every image B_i in the working set for a number of iterations m, alternating whether we loop in scanline or reverse-scanline order. At the current pixel position there is stored a correspondence $f_i(x, y)$. We propose a set of new candidate correspondences Z which is built up using five different operators. Specifically, this is a set, so we do not store duplicates, and we also store both the source position (x, y, i) and the terminus position (x', y', j). Through empirical observation, we found the following set of operators converges more efficiently than any subset: propagation, random search, enrichment (forward enrichment of two hops), binning, and uniform sampling. The first four operators we have presented in earlier sections. The last operator, uniform sampling, simply picks a patch uniformly at random from the working set, excepting the currrent image.

After running these operators, we will have a set of candidate improved correspondences, each of which has source at the current pixel and target in a different image. We compute the patch distance associated with each correspondence $(x, y, i) \to (x', y', j)$: if it is lower than the distance currently stored in the NNF at position (x, y), then we update the NNF with the better correspondence. In order to gain a richer set of correspondences, for each correspondence we also examine the complement correspondence $(x', y', j) \to (x, y, i)$, and if this is lower than the distance stored in the NNF at the target position (x', y', j) we update that entry as well. This operation is inverse enrichment. At last, after a fixed number of iterations on the current working set, we sample a new working set.

4.3.5 Working Set Selection

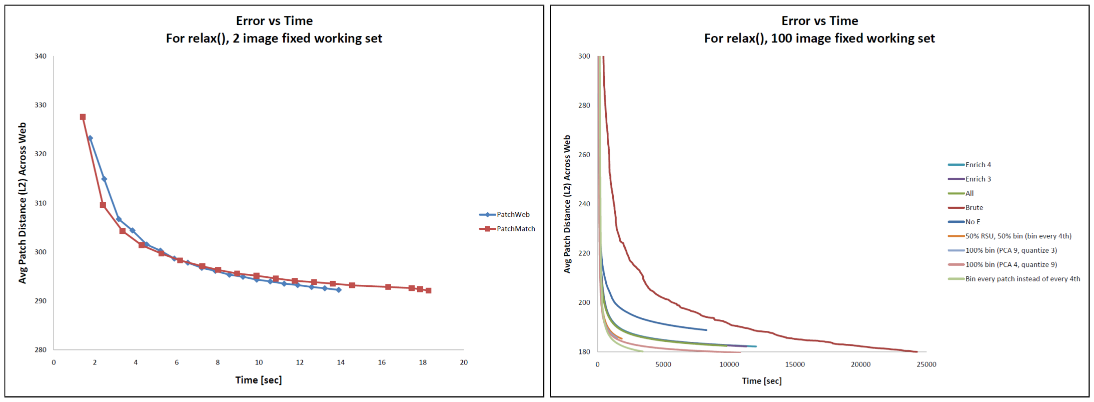
There are a few options for building a working set. The simplest strategy is to sample uniformly from the set of all images. A more advanced strategy is to retain a fixed fraction of the original working set, sample a new fixed fraction of new images uniformly, and sample the remaining images according to the enrichment operator applied to the first two sets of chosen images. Preliminary experiments indicate that the latter strategy converges more quickly. We hypothesize that this is because the images implictly cluster into groups having related patch appearance. We plan to explore this more thoroughly in our ongoing work.

4.3.6 Parallelism

The algorithm parallelizes trivially over multiple cores or nodes in a cluster. Multiple compute nodes can even be improving the same NNF, independently, within different working sets, at the same time. The only synchronization that needs to be done is when writing back an NNF, the copy on disk should be locked, and merged with the NNF in memory, choosing the nearest neighbor with lower distance.

4.3.7 Querying a Web via Relaxation

At this point we have built a web. We can use exactly the same relaxation algorithm to query the web. Suppose we have an image A that we wish to query against a web of images $B_1, \ldots, B_n$. We simply add image A to our set of images and build a web on the combination. We add a few constraints: first, for efficiency, image A should always be selected in every working set, even though on multiprocessor machines this means there will be multiple NNFs associated with image A. We



(a) $n = 2$ (b) $n = 100$

Figure 4.4: Performance for building a web. Comparison with PatchMatch on all pairs of images. Left is $n = 2$ images, our algorithm and PatchMatch have similar performance. Right is $n = 100$ images, the best version of our algorithm is the bottommost line, whereas the topmost line is PatchMatch on all pairs. For a given patch distance, we outperform PatchMatch significantly, by a factor of 8-20.

simply reduce these NNFs, taking the best nearest neighbor across all processors, after relaxations are done. Second, to avoid corrupting the existing web for image collection B, we load the NNFs for the B images in read-only mode. Third, again for efficiency, it does not help to find better links between B images, so instead of looping through the entire working set in scan and reverse-scan order, we simply repeatedly loop through image A until convergence.

4.3.8 Convergence Analysis

Here we discuss the number of iterations required to converge. In general, the number of iterations required to build the web and to query the web can be different, and application dependent.

For building the web, we compare in Figure 4.4 the performance of PatchWeb against a straw-man algorithm: run PatchMatch on all possible pairs of images, and take the best nearest neighbor found overall. This is a reasonable comparison, as PatchMatch gives state-of-the-art performance for finding correspondences between two images. Our algorithm's performance increases for increasing numbers of images n, because the all-pairs PatchMatch algorithm takes $O(n^2)$ time, whereas PatchWeb takes time that is closer to linear, though of course determined by empirical factors such as working set size. As a sanity check we compare our algorithm with PatchMatch for the case $n = 2$ and find the performance of our algorithm is comparable to PatchMatch. This is expected, since we use similar operations, and neither enrichment nor binning is expected to help much until there are many images.

The performance of our algorithm is empirically determined. This is good, because it allows us to have a flexible algorithm that scales well. But can we bound the performance of the algorithm? Consider a synthetic input where we have n images of which two are identical, we can call them images 1 and 2. How long will it take for them to find each other? Suppose the working set size is m. Because of binning, if the images are chosen in the same working set, with probability almost one they will find each other. Assume for simplicity that the working set policy is to sample randomly when choosing the working set. The probability that our two images are in the same working set is:

$$p = \frac{\#\,\text{subsets of containing 1, 2}}{\#\,\text{subsets}} = \frac{\binom{n-2}{m-2}}{\binom{n}{m}} = \frac{m(m-1)}{n(n-1)} \approx \frac{m^2}{n^2} \tag{4.1}$$

This is a geometric distribution so it will take on average n^2/m^2 choices of working sets before the images find each other. Each working set requires $O(m)$ time to process, so on a single processor this takes n^2/m time overall. So if we can increase the working set size m as n increases, because we have spare memory, our algorithm takes $O(n)$ time overall, but once we run out of memory our algorithm takes $O(n^2)$ time. However, in many applications it is not necessary to obtain an exact match, only a reasonable match under some error threshold. Also in the future, by exploring more carefully the issue of working set selection, we may be able to improve asymptotic performance.

4.3.9 Applications

We have not currently implemented any applications for PatchWeb, other than preliminary experiments with image completion that were only partially successful. The two challenges we ran into with image completion are that (1) synthesis algorithms tend to fall into many local minima, because almost every possible patch appearance is represented somewhere in the dataset, and (2) synthesized patches do not necessarily come from the correct semantic scene, or have the correct brightness properties. For applications, in future work we may explore super resolution [43, 46], denoising [68], image classification using dense descriptors [14, 56, 112], enhancement of personal photos, paint by numbers [22], or compression of images or video.

4.4 Summary

In summary, in this chapter we have presented parallel variants of the PatchMatch algorithm, as well as PatchWeb, which addresses scalability concerns in large image collections. We have introduced two additional search operators: enrichment and binning, which help the search process, particularly in large image collections. We have also suggested potential applications of PatchWeb.

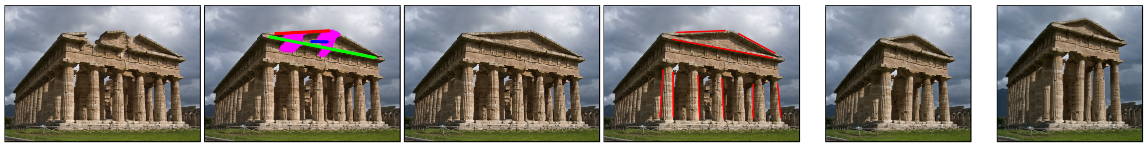
Chapter 5

Image Editing Applications

5.1 Introduction

As digital and computational photography have matured, researchers have developed methods for high-level editing of digital photographs and video to meet a set of desired goals. For example, recent algorithms for *image retargeting* allow images to be resized to a new aspect ratio – the computer automatically produces a good likeness of the contents of the original image but with new dimensions [94, 116]. Other algorithms for *image completion* let a user simply erase an unwanted portion of an image, and the computer automatically synthesizes a fill region that plausibly matches the remainder of the image [29, 60]. *Image reshuffling* algorithms make it possible to grab portions of the image and move them around – the computer automatically synthesizes the remainder of the image so as to resemble the original while respecting the moved regions [25, 98].

In each of these scenarios, user interaction is essential, for several reasons: First, these algorithms sometimes require user intervention to obtain the best results. Retargeting algorithms, for example, sometimes provide user controls to specify that one or more regions (e.g., faces) should be left



(a) original (b) hole+constraints (c) hole filled (d) constraints (e) constrained retarget(f) reshuffle

Figure 5.1: Structural image editing. Left to right: (a) the original image; (b) a hole is marked (magenta) and we use line constraints (red/green/blue) to improve the continuity of the roofline; (c) the hole is filled in; (d) user-supplied line constraints for retargeting; (e) retargeting using constraints eliminates two columns automatically; and (f) user translates the roof upward using reshuffling.

relatively unaltered. Likewise, the best completion algorithms offer tools to guide the result by providing hints for the computer [106]. These methods provide such controls because the user is attempting to optimize a set of goals that are known to him and not to the computer. Second, the user often cannot even articulate these goals *a priori.* The artistic process of creating the desired image demands the use of trial and error, as the user seeks to optimize the result with respect to personal criteria specific to the image under consideration.

The role of interactivity in the artistic process implies two properties for the ideal image editing framework: (1) the toolset must provide the flexibility to perform a wide variety of seamless editing operations for users to explore their ideas; and (2) the performance of these tools must be fast enough that the user quickly sees intermediate results in the process of trial and error. Most high-level editing approaches meet only one of these criteria. For example, one family of algorithms known loosely as *non-parametric patch sampling* has been shown to perform a range of editing tasks while meeting the first criterion – flexibility [52, 98, 119]. These methods are based on small (e.g. 7x7) densely sampled patches at multiple scales, and are able to synthesize both texture and complex image structures that qualitatively resemble the input imagery. Because of their ability to preserve structures, we call this class of techniques *structural image editing.* Unfortunately, until now these methods have failed the second criterion – they are far too slow for interactive use on all but the smallest images. However, due to our novel fast matching algorithm that accelerates such methods by at least an order of magnitude, it is now possible to apply these techniques in an interactive structural image editing framework.

In Section 5.3, we demonstrate the application of this algorithm in the context of a structural image editing program with three modes of interactive editing: image retargeting, image completion and image reshuffling. The system includes a set of tools that offer additional control over previous methods by allowing the user to constrain the synthesis process in an intuitive and interactive way (Figure 5.1). Because our image completion produced high quality results in times that are user interactive, it was included in Adobe Photoshop CS5 as a feature "content aware fill" (Figure 5.2).

5.2 Related Work

Patch-based sampling methods have become a popular tool for image and video synthesis and analysis. Applications include texture synthesis, image and video completion, summarization and retargeting, image recomposition and editing, image stitching and collages, new view synthesis, noise removal and more. We will next review some of these applications and discuss their degree of

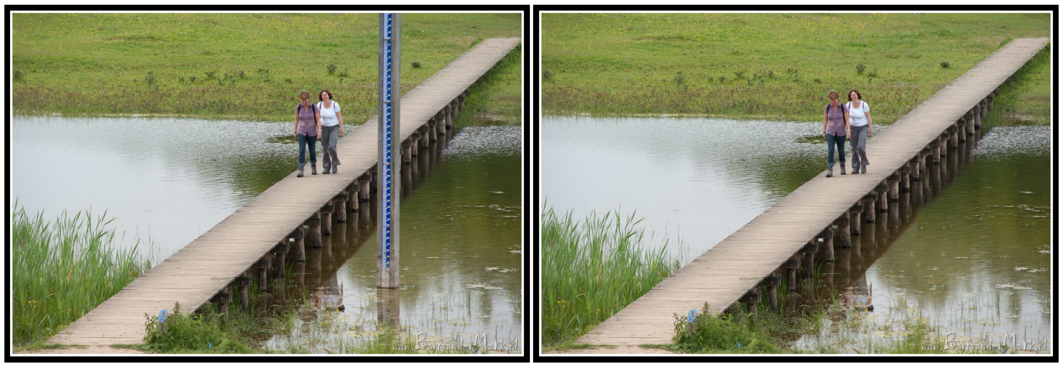

(a) input (b) result

Figure 5.2: Example of image completion by the Photoshop Content-Aware Fill interface. (a) the original image; (b) post is selected by the user and removed. A high quality result is produced because there are many repetitive patterns in the input image. Image courtesy of Bart van der Mark.

interactivity.

Texture synthesis and completion. Efros and Leung [37] introduced a simple non-parametric texture synthesis method that outperformed many previous model based methods by sampling patches from a texture example and pasting them in the synthesized image. Further improvements modify the search and sampling approaches for better structure preservation [3, 29, 35, 36, 65, 69, 117]. The greedy fill-in order of these algorithms sometimes introduces inconsistencies when completing large holes with complex structures, but Wexler et al. [119] formulated the completion problem as a global optimization, thus obtaining more globally consistent completions of large missing regions. This iterative multi-scale optimization algorithm repeatedly searches for nearest neighbor patches for all hole pixels in parallel. Although their original implementation was typically slow (a few minutes for images smaller than 1 MP), our algorithm makes this technique applicable to much larger images at interactive rates. Patch optimization based approaches have now become common practice in texture synthesis [61, 64, 118]. In that domain, Lefebvre and Hoppe [67] have used related parallel update schemes and even demonstrated real-time GPU based implementations. Komodakis and Tziritas [60] proposed another global optimization formulation for image completion using Loopy Belief Propagation with an adaptive priority messaging scheme. Although this method produces excellent results, it is still relatively slow and has only been demonstrated on small images.

Control and interactivity. One advantage of patch sampling schemes is that they offer a great deal of fine-scale control. For example, in texture synthesis, the method of Ashikhmin [3] gives the user control over the process by initializing the output pixels with desired colors. The image analogies framework of Hertzmann et al. [52] uses auxiliary images as "guiding layers," enabling a variety of effects including super-resolution, texture transfer, artistic filters, and texture-by-numbers. In the field of image completion, impressive guided filling results were shown by annotating structures that cross both inside and outside the missing region [106]. Lines are filled first using Belief Propagation, and then texture synthesis is applied for the other regions, but the overall run-time is on the order of minutes for a half MP image. Our system provides similar user annotations, for lines and other region constraints, but treats all regions in a unified iterative process at interactive rates. Fang and Hart [39] demonstrated a tool to deform image feature curves while preserving textures that allows finer adjustments than our editing tools, but not at interactive rates. Pavic et al. [87] presented an interactive completion system based on large fragments that allows the user to define the local 3D perspective to properly warp the fragments before correlating and pasting them. Although their system interactively pastes each individual fragment, the user must still manually click on each completion region, so the overall process can still be tedious.

Image retargeting. Many methods of image retargeting have applied warping or cropping, using some metric of saliency to avoid deforming important image regions [72, 95, 116, 120]. Seam carving [5, 94] uses a simple greedy approach to prioritize seams in an image that can safely be removed in retargeting. Although seam carving is fast, it does not preserve structures well, and offers only limited control over the results. Simakov et al. [98] proposed framing the problem of image and video retargeting as a maximization of bidirectional similarity between small patches in the original and output images, and a similar objective function and optimization algorithm was independently proposed by Wei et al. [118] as a method to create texture summaries for faster synthesis. Unfortunately, the approach of Simakov et al. is extremely slow compared to seam carving. Our constrained retargeting and image reshuffling applications employ the same objective function and iterative algorithm as Simakov et al., using our new nearest-neighbor algorithm to obtain interactive speeds. In each of these previous methods, the principal method of user control is the ability to define and protect important regions from distortion. In contrast, our system integrates specific user-directable constraints in the retargeting process to explicitly protect lines from bending or breaking, restrict user-defined regions to specific transformations such as uniform or non-uniform scaling, and fix lines or objects to specific output locations.

Image "reshuffling" is the rearrangement of content within an image, according to user input, without precise mattes. Reshuffling was demonstrated simultaneously by Simakov et al. [98] and by Cho et al. [25], who used larger image patches and Belief Propagation in an MRF formulation. Reshuffling requires the minimization of a global error function, as objects may move significant distances, and greedy algorithms will introduce large artifacts. In contrast to all previous work, our reshuffling method is fully interactive. As this task might be particularly hard and badly constrained, these algorithms do not always produce the expected result. Therefore interactivity is essential, as it allows the user to preserve some semantically important structures from being reshuffled, and to quickly choose the best result among alternatives.

5.3 Editing Tools

In this section, we discuss some of the novel interactive editing tools enabled by our algorithm. First, however, we must revisit the bidirectional similarity synthesis approach [98]. This method is based on a bidirectional distance measure between pairs of images - the source (input) image S and a target (output) image T. The measure consists of two terms: (1) The *completeness* term ensures that the output image contains as much visual information from the input as possible and therefore is a good summary. (2) The *coherence* term ensures that the output is coherent w.r.t. the input and that new visual structures (artifacts) are penalized. Formally, the distance measure is defined simply as the sum of the average distance of all patches in S to their most similar (nearest-neighbor) patches in T and vice versa:

$$d_{BDS}(S,T) = \overbrace{\frac{1}{N_S}\sum_{s\subset S}\min_{t\subset T} D(s,t)}^{d_{complete}(S,T)} + \overbrace{\frac{1}{N_T}\sum_{t\subset T}\min_{s\subset S} D(t,s)}^{d_{cohere}(S,T)} \tag{5.1}$$

where the distance D is the SSD (sum of squared differences) of patch pixel values in L*a*b* color space. For image retargeting, we wish to solve for the image T that minimizes d_{BDS} under the constraints of the desired output dimensions. Given an initial guess for the output image, T_0, this distance is iteratively minimized by an EM-like algorithm. In the E step of each iteration i, the NN-fields are computed from S and T_i and "patch-voting" is performed to accumulate the pixel colors of each overlapping neighbor patch. In the M step, all the color "votes" are averaged to generate a new image T_{i+1}. To avoid getting stuck in bad local minima, a multi-scale "gradual scaling" process is employed: T is initialized to a low-resolution copy of S and is gradually resized by a small factor, followed by a few EM iterations after each scaling, until the final dimensions are obtained. Then, both T and S are gradually upsampled to finer resolutions, followed by more EM iterations, until the final fine resolution is obtained.

In addition to image retargeting, Simakov et al. [98] showed that this method can be used for other synthesis tasks such as image collages, reshuffling, automatic cropping and the analogies of these in video. Furthermore, if we define a missing region (a “hole”) in an image as T and the rest of the image as S, and omit the *completeness term*, we end up with exactly the image and video completion algorithm of Wexler et al. [119]. Importance weight maps can be used both in the input (e.g., for emphasizing an important region), and in the output (e.g., for guiding the completion process from the hole boundaries inwards).

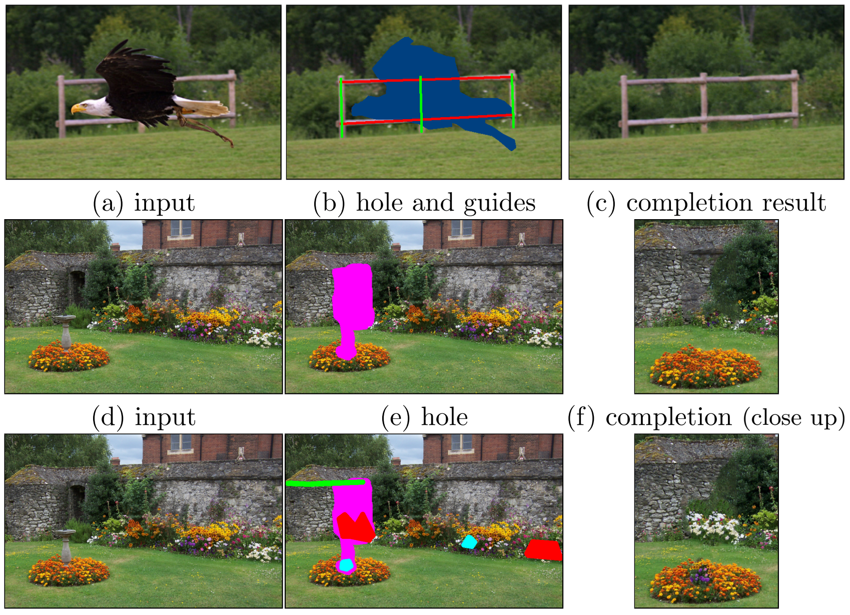
The randomized NNF algorithm given in Section 3.4 places no explicit constraints on the offsets other than minimizing patch distances. However, by modifying the search in various ways, we can introduce local constraints on those offsets to provide the user with more control over the synthesis process. For image retargeting, we can easily implement *importance masks* to specify regions that should not deform at all [5, 94, 98, 120] or scale uniformly [116], as in previous work. However, we can also explicitly define constraints not well supported by previous methods, such as straight lines that must remain straight, or objects and lines that should move to some other location while the image is retargeted. We can clone objects (“copy and paste”) or define something else that should replace the resulting hole. Objects can be scaled uniformly or non-uniformly (e.g, for “growing” trees or buildings vertically) in the context of structural image editing. All these are done naturally by marking polygons and lines in the image. A simple bounding box for objects often suffices.

Some of these image editing tools are new and others have been used before in a limited context. However, we believe that the collection of these tools – in conjunction with the interactive feedback provided by our system – creates new powerful image editing capabilities and a unique user interaction experience.

5.3.1 Search Space Constraints

Image completion of large missing regions is a challenging task. Even the most advanced global optimization based methods [60, 119] can still produce inconsistencies where structured content is inpainted (e.g., a straight line that crosses the missing region). Furthermore, in many cases the boundaries of the missing region provide few or no constraints for a plausible completion. Sun et al. [106] proposed a guided structure propagation approach, in which the user draws curves on top of edges that start outside the hole and defines where they should pass through it. This method propagates structure nicely along the curves, but the overall process is still slow (often in the order of a few minutes for a 0.5 megapixel image) and sometimes requires further manual intervention.

In our work, we have adopted the same user interaction approach, allowing the user to draw curves across the missing region. The curves can have different labels (represented in our interface


(a) input (b) hole and guides (c) completion result

(d) input (e) hole (f) completion (close up)

(g) same input (h) hole and guides (i) guided (close up)

Figure 5.3: Two examples of guided image completion. The bird is removed from input (a). The user marks the completion region and labels constraints on the search in (b), producing the output (c) in a few seconds. The flowers are removed from input (d), with a user-provided mask (e), resulting in output (f). Starting with the same input (g), the user marks constraints on the flowers and roofline (h), producing an output (i) with modified flower color and roofline.

using different colors) to indicate propagation of different structures. However unlike Sun et al. [106], which utilizes separate completion processes for curves and textured regions, our system synthesizes both simultaneously in the same unified framework. This is accomplished by limiting the search space for labeled pixels inside the hole to the regions outside the hole with the same label. (Paradoxically, adding these extra constraints accelerates the convergence properties by limiting the search space.) Figure 5.1 illustrates the effect of these search space constraints for image completion. In addition to curve and edge structures, the same tool can be used to specify particular contents for some portions of the hole. This type of interaction is similar to the "texture by numbers" approach [52] when applied to image completion. We show examples of these cases in Figures 5.1 and 5.3.

5.3.2 Deformation Constraints

Many recent retargeting methods allow the user to mark *semantically important* regions to be used with other automatically-detected cues (e.g., edges, faces, saliency regions) [5, 98, 116]. One important cue that has been overlooked by previous methods are the lines and objects with straight edges that are so common in images of both man-made scenes (indoor photos, buildings, roads) and in natural scenes (tree trunks, horizon lines). Keeping such lines straight is important to produce plausible outputs. However, marking a line as an important *region* in existing techniques usually forces the line to appear in its entirety in the output image, but does not guarantee that the line will not bend or break (see Figures 5.10). Moreover often we do not care if the line gets shorter or longer in the output, as long as it remains straight. We show additional examples of straight-line constraints in Figures 5.4 and 5.5.

Model constraints. To accomplish this, we extend the BDS formulation from a free optimization to a *constrained* optimization problem, by constraining the domain of possible nearest neighbor locations in the output for certain subsets of input points. Following the notation of equation (5.1), let $\{p_i\} \in L_k$ refer to all the pixels belonging to the region (or line) L_k in the input image S, and let $\{q_j\} \in T | NN(q_j) \in L_k$ refer to all the points q_j in the output T whose nearest neighbors lie in the region L_k (see notations in Figure 5.6). The objective from equation (5.1) then becomes:

$$\arg\min d_{BDS} \quad \text{s.t.} \quad \mathcal{M}_k(p_i, NN(p_i), q_j, NN(q_j)) = 0 \tag{5.2}$$

where we have K models $\mathcal{M}_k$ ($k \in 1 \ldots K$) to satisfy. The meaning of $\mathcal{M}_k() = 0$ is as follows: in the case of lines, satisfying a model means that the dot product of the line 3-vector (l, in homogeneous coordinates) and all the points in the output image should be zero, i.e., $NN(p_i)^T l = 0$. In the case of regions, we limit ourselves here to 2D homography transformations (H) and therefore satisfying the model means that the distance of all projected points and the corresponding NN points in the output image should be zero, i.e., $H_k p_i - NN(p_i) = 0$.

Now the question remains how best to impose these constraints on the solution. We observed that during the "gradual scaling" process lines and regions deform only gradually due to lack of space. This gives us an opportunity to correct these deformations using small adjustments after each EM

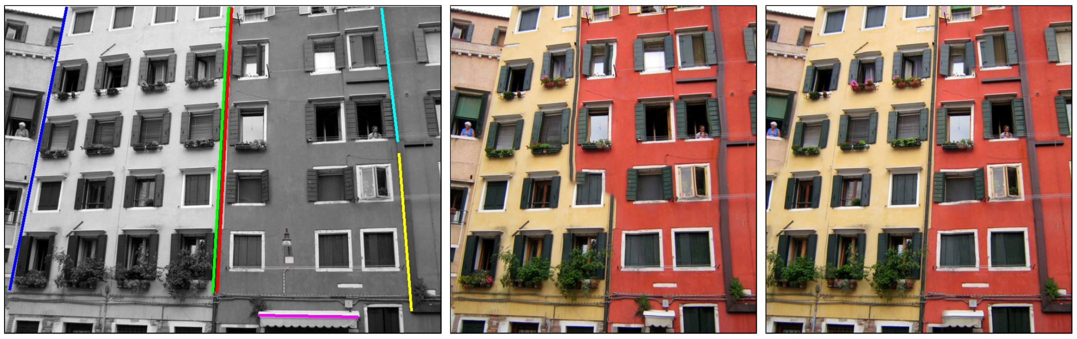

(a) original (b) retargeted (c) with constraints

Figure 5.4: Model constraints. The original image (a) is retargeted without constraints, producing broken boundaries in (b). When constraints shown as colored lines are added, the boundaries remain straight (c).

iteration. So, in order to satisfy the additional constraints, we apply an iterative correction scheme using the RANSAC [41] robust estimation method after each iteration. We assume that in each iteration most of the locations of $NN(p_i)$ as well as q_j *almost* satisfy the desired model, and we estimate this model robustly by discarding outliers. The estimated model is used to project the output points onto $\widehat{NN(p_i)}$ and $\widehat{q_j}$, and to correct the NN fields accordingly. For regions, outlier points are corrected but we obtained better results for lines by excluding outliers from the voting process.

We found the following models to be useful for constrained retargeting and reshuffling applications: *Free lines*, with output points constrained to lie on a straight line with unconstrained translation and slope (see Figure 5.5); *Fixed-slope lines*, with slope identical to the input, but free translation (see Figure 5.4); *Fixed-location lines*, with fixed slope and user-defined translation, for which there is no model to estimate, but the points are still projected onto the line allowing its length to change (see Figure 5.11(n) where the water-line was dragged down with a line constraint); *Translating regions*, with free translation but fixed scale (see Figure 5.10(right) where the car and bridge were marked as a translating region); and *Scaled regions*, with user-defined uniform scale and free translation (see Figure 5.5).

Hard constraints (a.k.a. "reshuffling"). The model constraints described in the previous section usually succeed in preserving lines and regions. However, in difficult cases, with large scale factors – or when there are contradictions between the various constraints – they cannot all be well satisfied automatically. In other cases, such as in image *reshuffling* [25, 98], the user may want to strictly define the location of a region in the output as a hard constraint on the optimization. This can be easily done in our framework, by fixing the NN fields of the relevant region points to the desired offsets according to these hard constraints. After each EM iteration we simply correct

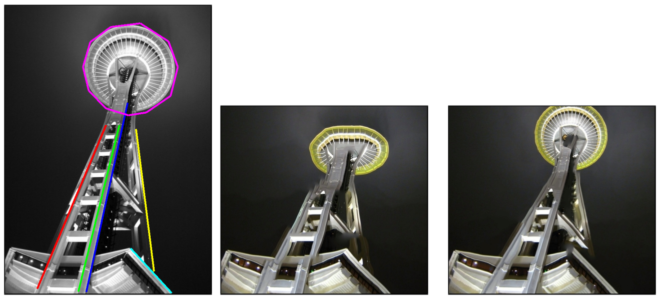

(a) original (b) retargeted (c) with constraints

Figure 5.5: Free lines and uniform scale constraints. The original image (a) is retargeted without constraints (b). Constraints indicated by colored lines produce straight lines and the circle is scaled down to fit the limited space (c).

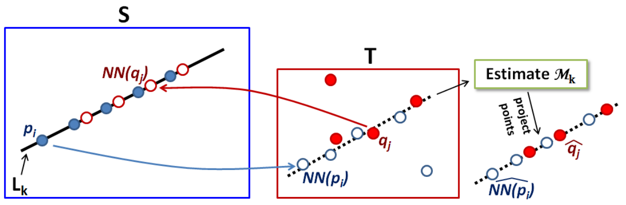
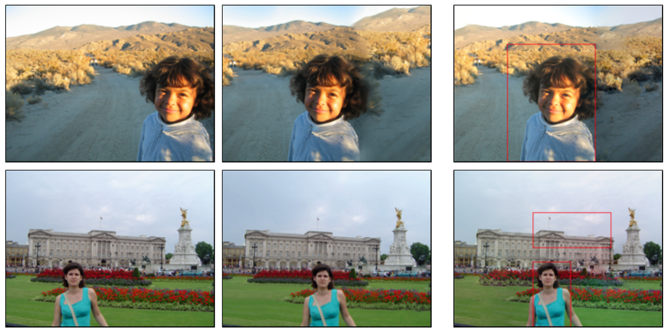


Figure 5.6: Model constraints. L_k is a straight line in the source image S. For point p_i on L_k, the nearest neighbor in T is $NN(p_i)$. A point q_i in T has nearest neighbor $NN(q_i)$ that lies on L_k. We collect all such points $NN(p_i)$ and q_i and robustly compute the best line M_k in T, then project the points to the estimated line.

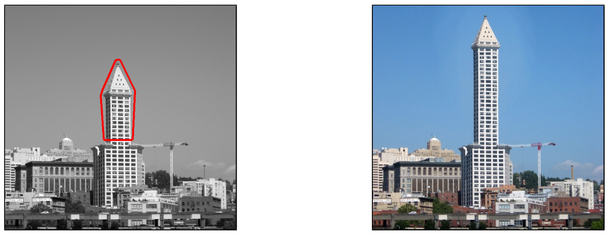
the offsets to the output position, so that the other regions around the objects gradually rearrange themselves to align with these constrained regions. For an object that moves substantially from its original output location, we give three intuitive options to the user to determine the initialization of the contents of the hole before the optimization starts: *swap*, in which the system simply swaps the pixels between the old and new locations; *interpolate*, in which the system smoothly interpolates the hole from the boundaries (as in Wexler et al. [119]); and *clone*, in which the system simply clones the object in its original location. For small translations these all generate similar outputs, but for large objects and large motions these options lead to entirely different results (see Figure 5.11).

Local structural scaling. A tool shown in Figure 5.8 allows the user to mark an object and rescale it while preserving its texture and structure (unlike regular scaling). We do this by gradually scaling the object and running a few EM iterations after each scale at the coarse resolution, just as in the global retargeting process.



(a) input (b) our reshuffling (c) Patch Transform

Figure 5.7: Examples of reshuffling. Input images are shown in the first and fourth column. The user annotates a rough region to move, and the algorithm completes the background in a globally consistent way. “Patch Transform” results are shown from Cho et al. [2008], which takes minutes to run, and our algorithm, which runs interactively.


(a) building marked by user (b) scaled up, preserving texture

(c) bush marked by user (d) scaled up, preserving texture.

Figure 5.8: Examples using local scale tool. In both examples, the user marks a source polygon, and then applies a nonuniform scale to the polygon, while preserving texture.

5.3.3 Implementation Details

Small changes of orientation and scale in some of the deformation models (e.g., *free lines* and *scaled regions*) can be accomplished by simply rearranging the locations of existing patches. For larger angle and scale changes that may be required for extreme retargeting factors, one may have to rotate/scale the patches as well. In each of the above cases we use a weighted version of equation (5.1) (see [98]) and we increase the weight of patches in the important regions and lines by 20%. We also note that finer levels of the pyramid have better initial guesses, and therefore the search problem is easier and fewer EM iterations are needed. We thus use a high value (typically 20-30) of EM iterations at the coarsest level, and at finer levels we use a number of EM iterations that decreases linearly with the pyramid level. For the last few levels of the pyramid, global matching is less necessary, so we find that reducing the random search radius to $w = 1$ does not significantly affect quality.

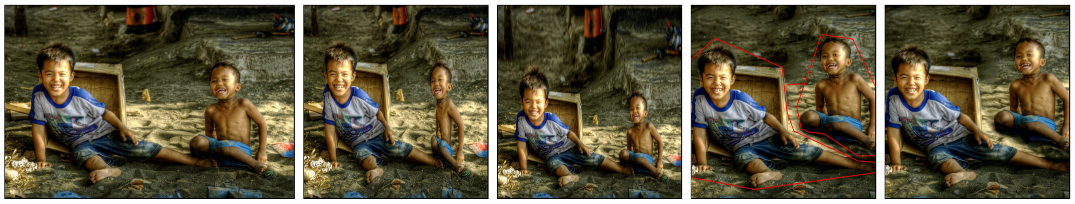

(a) (b) (c) (d) (e)

Figure 5.9: Retargeting. From left: (a) Input image, (b) [Rubinstein et al. 2008], (c) [Wang et al. 2008], (d) Our constraints, (e) Our result.

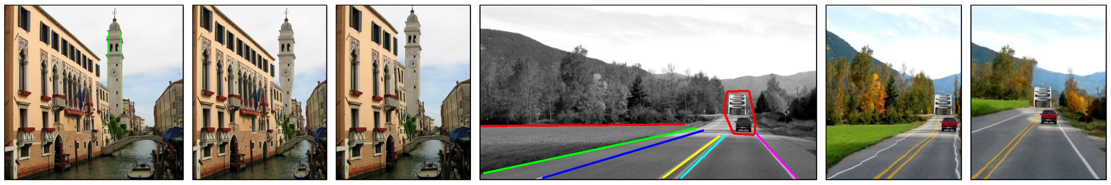

(a) (b) (c) (a) (b) (c)

Figure 5.10: Retargeting: (a) Input image, annotated with constraints, (b) [Rubinstein et al. 2008], (c) Our output.

5.4 Results, Discussion, and Future Work

As we have discussed, our nearest-neighbor framework can be used for retargeting, completing holes, and reshuffling content in images. We have compared performance and quality with several other competing methods in these domains.

Demonstrations of the interactive use of these tools can be seen in the videos accompanying our paper [8]. Figures 5.9 and 5.10 illustrate differences between the results produced by our method and those of Rubinstein et al. [94] and Wang et al. [116]. In Figure 5.9, both existing retargeting methods deform one of the two children in the photograph, whereas our system allows us to simply reshuffle one of the children, thus gaining more space for retargeting, and the background is automatically reconstructed in a plausible manner. In Figure 5.10, we see that "seam carving" introduces unavoidable geometric distortions in straight lines and compresses repeating elements. In contrast, our method allows us to explicitly preserve perspective lines and elide repetitive elements.

By marking out mask regions, users can interactively fill nontrivial holes. For example, in Figure 5.1, we use our system to perform some long-overdue maintenance on an ancient Greek temple.

Our reshuffling tools may be used to quickly modify architectural dimensions and layouts, as shown in Figure 5.11. Visually plausible buildings – albeit occasionally fantastical! – can be constructed easily by our algorithm, because architecture often contains many repeating patterns. Reshuffling can also be used to move human or organic objects on repeating backgrounds, as previously illustrated in Figure 5.7. However, in some cases line constraints are needed to fix any linear elements of the background that intersect the foreground object, as demonstrated in Figure 5.9, in which a line constraint is used to constrain the shadow in the sand.

As with all image synthesis approaches, our algorithm does have some failure cases. Most notably, for pathological inputs, such as the synthetic test case of Section 3.8.1, we have poor convergence properties. In addition, extreme edits to an image can sometimes produce "ghosting" or "feathering"

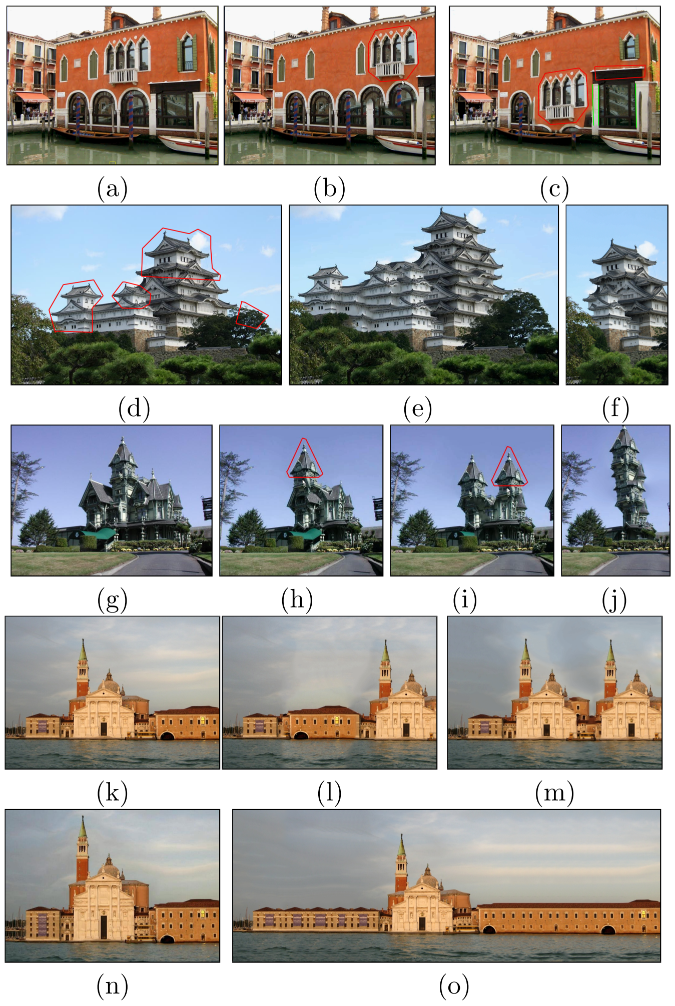


Figure 5.11: Modifying architecture with reshuffling. (a-c) The window position for (a) an input image is (b) moved to the right or (c) to the lower story of the building. (d-f) Hard constraints on a building are used to make various renovations. (g) Another house and (h-j) combinations of retargeting and reshuffling produce more extreme remodeling variants. (k-n) The buildings in (k) the input image are (l) horizontally swapped (m) cloned, (n) vertically stretched, (o) and widened.

artifacts where the algorithm simply cannot escape a large local minimum basin. However, we point out that the speed of our algorithm makes it feasible to either introduce additional constraints or simply rerun the algorithm with a new random seed to obtain a different solution. Although such repeated trials can be a burden with a slower algorithm, in our experiments we occasionally enjoyed such explorations of the space of creative image manipulations!

Among the many exciting avenues for future work in this domain, we highlight several important ones:

Extending the matching algorithm. One might use the *k*-coherence strategy [109] with our algorithm. In general, we find that the optimal random sampling pattern is a function of the inputs, as discussed in Chapter 2. By exploring these tradeoffs and further investigating GPU implementations, additional speed gains may be realized, opening up new applications in real-time vision and video processing.

Other applications. Although we have focused on creative image manipulation in this chapter, we also hope to implement completion, retargeting, and reshuffling for video. Fitzgibbon et al. [42] obtained state-of-the-art results for new view synthesis by using a patch-sampling objective term and a geometry term. Dense nearest neighbor patch search was also shown to be useful for learning based superresolution [43]. These are all excellent candidates for acceleration using our algorithm. We also hypothesize that by operating in the appropriate domain, it may be possible to apply our techniques to perform retargeting, hole filling, and reshuffling on 3D geometry, or 4D animation or volumetric simulation sequences. Finally, although the optimization we perform has no explicit neighborhood term penalizing discontinuous offsets (as required in stereo and optical flow), we believe its speed may be of use as a component in vision systems requiring fast, dense estimates of image correspondence, such as object tracking.

In summary, we have applied our new fast matching algorithm to a wide variety of patch-based optimization approaches, so that image synthesis can now be applied in a real-time interactive interface. Furthermore, the simplicity of our algorithm makes it possible to introduce a variety of high-level semantic controls with which the user can intuitively guide the optimization by constraining the nearest-neighbor search process. We believe we have only scratched the surface of the kinds of interactive controls that are possible using this technique.

Chapter 6

Video Tapestries

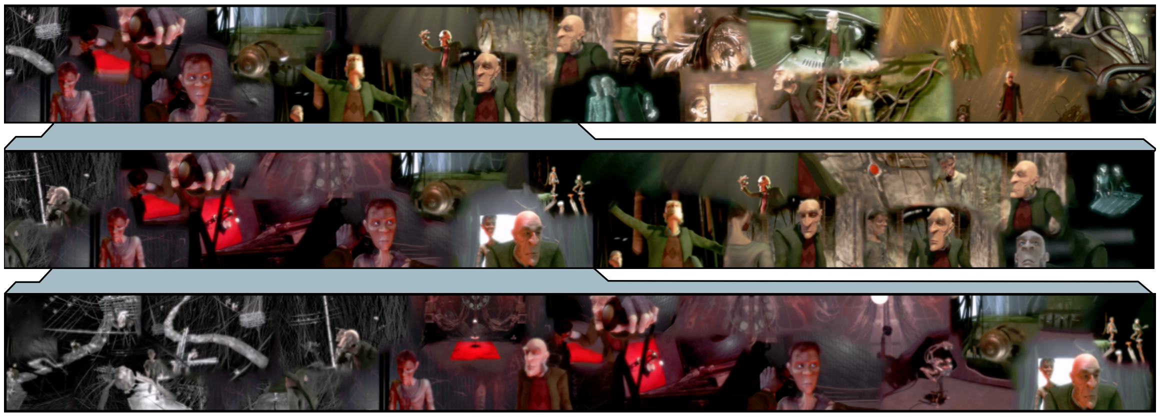


Figure 6.1: A multiscale tapestry represents an input video as a seamless and zoomable summary image which can be used to navigate through the video. This visualization eliminates hard borders between frames, providing spatial continuity and also continuous zooms to finer temporal resolutions. This figure depicts three discrete scale levels for the film *Elephants Dream* (Courtesy of the Blender Foundation). The lines between each scale level indicate the corresponding domains between scales. See the video online at `www.cs.princeton.edu/gfx/pubs/Barnes_2010_VTW/` to view the continuous zoom animation between the scales.

6.1 Introduction

In recent years, the number of videos available in digital form has increased dramatically. However, due to the large volumes of data involved, it is difficult for a user to locate a scene in even a single video, or quickly comprehend the content of a video by looking at an overview. For example, a DVD film typically contains 200,000 frames, and there may even be 20 times more raw footage before it is edited into its final form [13]. To help people efficiently browse, comprehend, and navigate video, both commercial software and research systems have explored how to best summarize video in the form of static images.

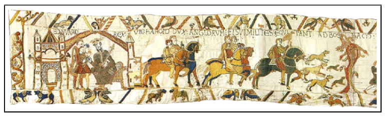
In the commercial domain, video navigation tools include simple scene selection interfaces, such as are found in DVD menus, and timelines that contain a sequence of images, as found in video editing software such as Adobe Premiere. In the case of DVD menus, a selection of thumbnails representing pre-selected scenes are shown to the user. In the case of timelines, one thumbnail may be shown for every shot of the video, or a sequence of thumbnails may be shown at spatial intervals corresponding linearly to the times of each scene. Each of these representations incurs a tradeoff: DVD menus are easily comprehensible, but they may be incomplete, omitting important temporal details. Video editing timelines are more complete because they contain more of the details – every shot is represented as a separate block – and they support zooming in to view more temporal detail, but they usually lack coherent transitions between zoom levels, and can be visually confusing when viewed at coarser zoom levels.

In this chapter, we present a novel approach to visualize timelines and browse through scenes of a video that attempts to provide the comprehensibility of a DVD menu with the completeness and ease of navigation of a video editing timeline. Previous work in video summarization has mostly focused on selecting important frames – called "keyframes" – and arranging them in layouts with hard borders. However, such systems generally ignore one or more critical aspects of the user experience in browsing through video. First, the use of hard borders between rectangular frames is an artifact of how film has historically been represented, and is not necessarily an optimal visual representation of video. The hard borders between frames introduce additional distracting and dominant image content, making discrete representations harder to visually parse. In contrast, our approach is inspired by the techniques that artists have historically used to depict action in a continuous manner without hard borders, in art forms such as medieval tapestries, architectural friezes, and color scripts for cinematography (Figure 6.2). Second, for applications like video editing, it should be possible to present a summary online, in real-time. Third, we believe users should be able to zoom in to a summary to expose fine-scale temporal details. As we have argued that abrupt discontinuities in the spatial layout are objectionable, we also believe that abrupt discontinuities in temporal zooming are also undesirable, because they can be disorienting to the viewer. Animating the transitions between scales helps the user assimilate the context and orientation of the transition [27]. While some previous systems satisfy some of our goals, no system satisfies all of them.

We present a unified framework for creating seamless summaries of video that allow the user to continuously zoom in to expose more temporal detail. We call these summaries *multiscale tapestries*. An example tapestry is shown in Figure 6.1.

Our contributions are: A set of criteria for evaluating the qualities of an optimal summary;

(a) Grail tapestry


(b) Bayeux tapestry

(c) Trajan's column

(d) color script by L. Bove

Figure 6.2: Our method is inspired by artistic techniques depicting action without hard borders, such as medieval tapestries (a-b), Roman friezes (c), and color scripts for cinematography (d).

summaries seamlessly blended across their width with implicit region-of-interest preservation; a multiscale representation with continuous zoom across temporal scales, and; rendering of tapestries in real-time with limited precomputation.

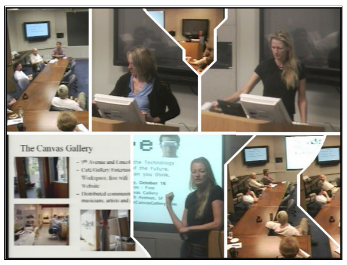
6.1.1 Criteria

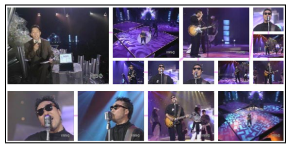
To construct our multiscale video tapestries, we propose four important qualities that are desirable in a visual summary of video, in rough priority order:

First, it should be *coherent*, depicting only visual entities that appear in the source video. This criterion motivates our desire to eliminate frame boundaries, since such structures do not exist in the source contents.

Second, it should be roughly *chronological*, presenting events in a spatial order that corresponds to their temporal order in the film. In our work as in most other such interfaces, we use a left-to-right ordering corresponding to Roman language reading order, but top-to-bottom and right-to-left orderings are trivially feasible. Although our horizontal layout does not fit all window shapes, it streamlines navigation by naturally mapping "left" and "right" to "before" and "after."

Third, it should be *continuous* in scale space, with visually smooth transitions from coarse to


(a) Chiu et al. [24]


(b) Uchihashi et al. [113]

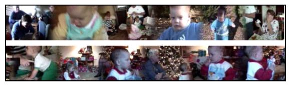

(c) Wang et al. [115]

Figure 6.3: Example arrangements in previous work: (a) Voronoi layout of keyframes, (b) rectangular layout, (c) soft-border collage.

fine temporal scales.

Fourth, it should be *complete*, containing as much of the unique visual content in the source video as possible given the available pixel budget. Since our tapestries have vastly fewer pixels than the source video, we prioritize the most unique content over repeated content.

We observe that two of these desiderata for tapestries, *coherence* and *completeness*, have been previously formalized using the notion of *bidirectional similarity* (BDS), and successfully applied to the related problems of image collage and video skims [98]. Furthermore, recent methods for approximating bidirectional similarity have made such image synthesis algorithms practical for high-quality interactive use, as demonstrated in Chapter 5. Thus, our system builds upon the synthesis algorithm of Simakov et al. and the PatchMatch algorithm described in Chapter 3 as its foundations, applying additional constraints and energy terms to impose the remaining criteria of chronological ordering and scale-space continuity.

6.1.2 Overview

After discussing related work (Section 6.2), we first review the concept of bidirectional similarity in the context of a single-scale tapestry (Section 6.3). Our multi-scale approach is then divided into three stages, for tractability. In the first stage (Section 6.4.1), our system selects a set of representative keyframes at each zoom scale. In the second stage (Section 6.4.2), static tapestries are constructed at each scale. In the third and final stage (Section 6.4.3), our system interpolates zoom animations between the discrete scales. We then present implementation details and results (Section 6.5), and the findings of our user study (Section 6.6). Finally, we will revisit how our criteria are resolved in each stage, limitations, and future work (Section 6.7).

6.2 Related Work

There is a large body of work on the problems of video summarization and navigation. A recent survey paper [111] comprehensively reviews these techniques, which can be broadly divided into two classes: *video skims*, and *image summaries*. Video skims [58, 74, 103, 104] summarize a longer video with a shorter summary video. These skims can be useful for navigation, but they potentially require that the user observe the video for a long time, and do not naturally exploit wide screen real estate as do the timelines or scene selection interfaces that we envision. Therefore, we restrict further discussion to methods that create static image summaries.

An important early work on visualizing video is [33]. This work used hierarchies of summarization and annotation icons to represent collections of video, and like our work was inspired by comic art and paintings. The work also introduced new video representations, such as the *videogram*, a summary image showing only the center vertical line from each frame, and time line icons: 3D stacks visualizing the spacetime volume.

Other early work on summarizing video as images focused on methods for selecting the important keyframes to be shown in the summary. For example, Dementhon et al. [34] used curve splitting in a feature space to determine keyframes, and allowed users to subdivide finer to expose more temporal detail. Subsequent work detected keyframes by greedy and global methods that consider how much change has occurred in various feature spaces, and used clustering and coverage methods that try to ensure that the summary represents the same set of content as the original video [111]. Work on selection of keyframes also often relied on detecting shot boundaries, which is now considered a mature research area [62]. In the context of skeletal animation, Assa et al. [4] utilized motion capture data to produce effective visual summaries of human actions. Our approach does use keyframes to limit computation, but employs a consistent energy function for keyframe selection and static tapestry composition. Previous methods such as [97, 115] have explored using multiple scales of temporal detail, however, our method is the first to offer continuous zooming between scales.

Other research has emphasized the importance of the composition of a static summary, examples of which are seen in Figure 6.3. For panoramic videos, Taniguchi et al. [108] automatically detected panoramas and key frames, and used an adaptive method to pack the resulting images to make effective use of space. Comic-book style layouts [17, 113] employed variation in sizes and tiling patterns of rectangular frames to create more appealing summaries. For news video, collages have been used to create more interesting layouts [26]. By detecting important regions and using Voronoi diagrams, Chiu et al. [24] created non-rectangular, arbitrary layouts which effectively pack the salient information into a limited space. For browsing into large numbers of video clips, [59] explored a mosaic representation, where each video is visualized as a small tile.

In each of these layout approaches, discrete moments in time have been separated by hard borders. Wang et al. [115] found that users prefer video summaries with soft blending between adjacent frames over hard boundaries. In later work, the soft blending was also extended to use arbitrary shapes for the layout of the summary [76, 121]. However, these methods still construct the summary by a sequence of independently motivated steps: selecting regions of interest within keyframes based on saliency maps, shot detection, and camera motion detection, extracting and resizing these keyframes, computing optimal seams, and applying blending. In contrast, we formulate the optimal summary image using just three stages with tightly related optimizations. Moreover, our method supports coherent and continuous zoom and can be computed quickly enough for interactive synthesis.

A number of tools are available for making collages and creating soft blendings for summaries. Previous work has demonstrated collage construction using graph cut and gradient domain techniques [2, 92, 93]. Optimal boundaries between collage regions can be detected by graph cuts [65], minimum cuts, or simply pasted in the gradient domain without optimization. More recently, Sivic et al. [100] showed that by using a large database of images, photorealistic transitions can be made between abutting images.

Concurrently, [28] have developed a technique for interactively creating seamless summaries of video. They extract a panoramic background plate, and compose matted foreground objects on the background. Unlike our system, their work focuses more on videos that convey specific actions, and they focus on interactive authoring and animation facilities. Our work in contrast focuses on summarizing long films with a fully automatic method, and we allow for continuous zoom.

We draw inspiration for our tapestries not only from ancient art but also from the "color scripts" used in animation production. These sketches, intended to depict the color palette of a film from beginning to end, are sometimes composed as semi-continuous strips representing multiple scenes [49]. See Figure 6.2 for one example.

Our multiscale tapestries are based on the bidirectional similarity summarization approach [98], as it provides us with a theoretical framework for how to optimally compute a tapestry of any input video. When employed for image summarization, this method can merge repetitive or redundant structures and implicitly defines saliency via uniqueness.

6.3 Single-Scale Tapestries

For simplicity, we begin by considering an objective for tapestries at a single zoom scale. The objective function for the optimal single-scale tapestry is inspired by the work of Simakov et al. [98], which defined an objective function to be minimized for image retargeting. This objective contains two complementary error terms, $d_{complete}$ and d_{cohere}, which ensure that every patch in the source image is found in the target image and vice versa. The complete objective is

$$d_{BDS}(S,T) = \overbrace{\frac{1}{N_S}\sum_{s\subset S}\min_{t\subset T} D(s,t)}^{d_{complete}(S,T)} + \overbrace{\frac{1}{N_T}\sum_{t\subset T}\min_{s\subset S} D(s,t)}^{d_{cohere}(S,T)} \tag{6.1}$$

This is the same as equation (5.1) presented in the context of image editing earlier. Here S is the source, or original image, T is the target image, small rectangular image patches s and t are sampled from the source and target images, and the number of source and target patches are N_S and N_T, respectively. The patches are of fixed size: we use 7x7 patches in our implementation. The distance $D(s,t)$ is the distance in color space between these square patches: we use L^2 distance in RGB space. When retargeting, the source image S will have different dimensions than the retargeted image T. Simakov et al. [98] also used the same energy function to *shorten* videos, by letting S and T represent video volumes of different dimensions, and s and t represent small space-time patches (boxes).

Our single-scale tapestry approach uses the same definition of optimality given by Equation 6.1, with the following changes to the defined terms: We start with an input video S with dimensionality three, to produce a summary image (tapestry) T with dimensionality two. We take image patches s and t, so the sum in d_{cohere} is taken over all $p \times p$ image patches in the target image T, and the sum in $d_{complete}$ is taken over all $p \times p$ image patches in *every frame* of the source video S. We wish to enforce at least a loose time ordering in the final summary, with time advancing approximately from left to right. This is effected by introducing an additional term to our distance function $D(s,t)$ between two patches s and t that maps the time dimension of the video S to the x-axis of the summary T:

$$D(s,t) = D_{color}(s,t) + \alpha(\tau_s - \beta x_t)^2 \tag{6.2}$$

where τ_s denotes time in the input video, x_t denotes horizontal position in the output tapestry, α is a user parameter that controls how strictly time must increase linearly from left to right, and β is a space-time proportionality factor chosen so the right side of the tapestry T coincides with the last frame of the input video S. Note that D_{color} is the color space distances between the patches as described above.

In principle, given an input video S, the best summary T of a user-specified resolution could be found by optimizing Equation (6.1). However, this is in general an NP-hard problem, so efficient approximations must be introduced to solve it effectively. Simakov et al. [98] proposed an iterative, guided algorithm that starts with an initial guess and iteratively refines T to produce a target image with minimal error. This iterative algorithm can be accelerated by the PatchMatch algorithm described in Chapter 3 to perform image synthesis at interactive rates. Our implementation relies on these approaches, but naïve application alone would still be intractable. It would be inefficient to perform the optimization on the entire input video S, because in most cases this video cannot even fit in core memory. Even with sufficient memory, the optimization would be far too slow even for a preprocessing step, much less a real-time interface.

For this reason, we split the optimization into two stages, starting at the level of entire frames and then proceeding to the finer level of image patches within frames. First, we find a subset of keyframes from the input video that optimizes a frame-level approximation of the objective (6.1) over the set of all input frames. This can also be viewed as clustering over the set of frames. Second, using only these keyframes as a restricted domain S we optimize Equation 6.1 at the level of small image patches. Since the clustering stage is the bottleneck computation, it can optionally be replaced by a simple constant-rate sampling of the input frames, or even using manual frame selection.

6.4 Multi-Scale Tapestries

Unfortunately, the formulation of single-scale tapestries described in the previous section does not incorporate our criterion of continuity between scales. Indeed, at two different scales, the subsets of keyframes chosen in the first stage may not even intersect. And even if they do, we are still faced with the challenge of interpolating smoothly between two images of different size and significantly differing content.

Thus, we must modify our objectives to handle multi-scale tapestries, and add a third stage for synthesizing intermediate tapestries between scales. In the subsections that follow we describe in detail each stage of multi-scale tapestry generation. Single-scale tapestry generation is essentially identical to the first two stages, but without the subset constraint ensuring that keyframes from each zoom level are present in the finer zoom levels.

6.4.1 Keyframe Clustering

In our automatic clustering process, we wish to find a fixed number of key frames n_1 for the coarsest level (we use $n_1 = 24$). For finer levels $i = 2, 3, \ldots, d$, where d is the number of levels to reach a maximum temporal sampling rate (e.g., 2 fps), we increase the number of keyframes n_i geometrically ($n_i = n_{i-1} * \rho$, where $\rho = 2$ in our implementation).

Given the objective of Equation (6.1), we perform our clustering as an offline precomputation. Suppose we wish to choose the set of keyframes K at zoom level 1 that minimizes the objective function. To perform this clustering efficiently we need to be able to evaluate Equation (6.1) quickly. Let $f_1, \ldots, f_n$ represent discrete frames from the input video. We precompute an $n \times n$ asymmetric affinity matrix $\mathbf{A}$, where A_{ij} is one direction in our objective function (6.1):

$$A_{ij} = \frac{1}{N_{f_i}} \left(\sum_{s \subset f_i} \min_{t \subset f_j} D_{color}(s,t) + \alpha(\tau_{f_i} - \tau_{f_j})^2 \right) \tag{6.3}$$

D_{color} represents the color distance between s and t, and τ_{f_i} and τ_{f_j} denote the times associated with the frames i and j. Note that the key frames in K have known time values, so we use their associated times directly instead of the spatial time mapping βx_t in Equation (6.2). Moreover, where $(\tau_{f_i} - \tau_{f_j})^2$ exceeds some threshold it will always dominate the patch color differences and therefore we can speed up the preprocessing step by omitting the computation of the affinity matrix outside a central diagonal band.

Given the affinity matrix $\mathbf{A}$ and a set of key frames $K = \{K_1, \ldots, K_m\}$, we approximate Equation (6.1) as follows:

$$d_{BDS}(K) = \overbrace{\frac{1}{n} \sum_{i=1}^{n} \min_{j=1 \ldots m} A_{i,K_j}}^{d_{complete}(K)} + \overbrace{\frac{1}{m} \sum_{i=1}^{m} \min_{j=1 \ldots n} A_{K_i,j}}^{d_{cohere}(K)} \tag{6.4}$$

The second term $d_{cohere}(K)$ is zero, as the diagonal elements of $\mathbf{A}$ are zero: each of the chosen key frames also exists in the set of input frames. Note that this is only an upper bound on the true objective, as for $d_{complete}$ the min operator in Equation 6.3 is taken only over the domain of a single keyframe rather than the collection of keyframes.

We employ the k-medoids clustering algorithm [12] to minimize this objective. The key frames at each level are found by taking the known key frames at the previous level, and solving for the best new key frames to add to the layout, while existing frames are not removed or changed. Thus the key frames at each level are a subset of the frames at the next level.

This automated keyframe selection process is usually effective at highlighting salient information, but it can optionally be replaced with simple sub-sampling in time (e.g. 2 fps), or the user can manually choose key frames to tell the story better. Uniform sub-sampling offers the advantage that the final tapestry proceeds approximately linearly with time, which may be desirable for video editing applications where the duration of events is important. Moreover, it requires no special processing. Manual selection has the benefit of human higher-level comprehension of characters, plot, and story arc. We have found the some combination of these works best: In our highest-quality results, we perform automatic clustering, but then a user manually adjusts the key frames slightly to create a tapestry with improved composition and more semantically meaningful contents.

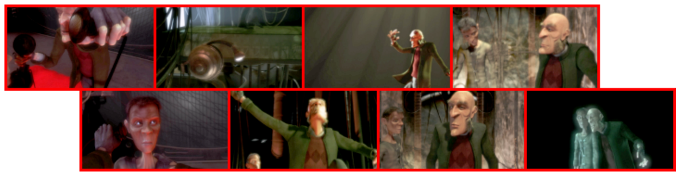


Figure 6.4: The brickwork layout used as an initial guess for the optimization. Courtesy of the Blender Foundation.

Figure 6.8 compares the results from these three approaches.

6.4.2 Discrete Tapestries

Given the key frames, we solve for the optimized tapestry independently at each scale by minimizing the objective in equation (6.1) using the method of Simakov et al. [98], which takes downhill steps in the objective function (this time at the patch level, as opposed to the frame level as in the previous section). We use a brickwork layout as our initial guess: the key frames are arranged in two rows, in increasing temporal order, as shown in Figure 6.4. Then we retarget this input image by a factor of 75% in the y direction, encouraging repetitive structures to merge and seamless blending to occur. This process was found through experimentation; we tried scaling in x and even more in y, and found that this process makes the best compromise between saving space and preserving important structures. Note that although we use a particular layout as our initial guess, only patches from the keyframes themselves are used in the S term, so the regions of the initial guess which overlap multiple keyframes will be blended or compressed away by the retargeting process.

This method successfully condenses the tapestry, eliminating redundant image features, condensing repeating elements, and blending image regions seamlessly. However, it can sometimes destroy high-level semantic information such as faces, so we introduce a weighting factor for each patch [98] and increase the weight of patches that overlap faces, as detected by the method of Bourdev and Brandt [18].

For efficiency, we compute tapestries in discrete tiles of about 500 pixels wide. Although this could in principle affect the output, we find it has little impact in practice, because the time component of the distance function in Equation 6.2 limits interaction between regions of the tapestry that are very far apart. We avoid seams between tiles by overlapping the output tiles, and in the retargeting process we introduce a hard constraint that the colors in the overlap region must match the previously computed tiles. The implementation of these hard constraints is similar to those of Chapter 5, except that here after each iteration of the retargeting algorithm, we composite feathered copies of the known tiles using alpha-blending to avoid abrupt visual discontinuities.

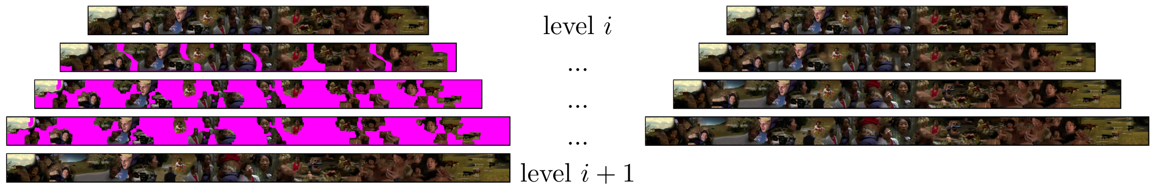


Figure 6.5: Continuous zooming. **Left:** "Island" constraints on the animated zoom sequence between a discrete tapestry at a smaller zoom value (top) and a larger zoom value (bottom). Regions present in both images are constrained to move with constant, linear velocity. Magenta regions are unconstrained. The interpolation starts with the bottom image as initial guess, and retargets the image to smaller sizes, holding the linearly-animated islands as hard constraints. Islands expand as the smaller image is reached, enforcing convergence. **Right:** Results of the retargeting where the unconstrained (magenta) regions have been synthesized. The sequence zoom exhibits temporal coherence.

When rendering tapestries in high quality mode, all tiles for all zoom levels are precomputed. When rendering tapestries as a lower resolution interactive process, we calculate tiles on-demand as the user visits each part of the zoom hierarchy. Even when running in the lower resolution interactive mode, the retargeting process takes about 0.5 second to run on each tile, so we cache the tiles, and to avoid pauses in the user interface, we use a background thread to pre-cache tiles in a small neighborhood of the user's current position in the x direction and in scale space.

6.4.3 Continuous Zoom

At this point we have computed discrete tapestries at a fixed set of zoom levels. To satisfy our desideratum of continuity in the animation between discrete levels, we now wish to fill in the space-time volume between each pair of the discrete tapestries in a temporally coherent way. Let us label a small tapestry at zoom level i as A and a larger tapestry at level $i+1$ as B: Our goal is to complete the space-time region between A and B.

The algorithm for filling this region proceeds as follows: First, our system identifies corresponding regions between the two discrete tapestries A and B. These regions are animated linearly across the gap between A and B. Finally, remaining regions are filled in using bidirectional similarity synthesis [8]. The paragraphs that follow provide additional details.

Finding correspondences. In Section 6.4.1 we guaranteed that the frames in A are a subset of the frames in B. Thus we can find correspondences between A and B, again using the fast correspondence algorithm of Chapter 3. This establishes dense correspondences between small square patches of size $p \times p$.

Identifying coherent corresponding regions. Next we wish to find corresponding regions between the two tapestries that are above some threshold size. We create an edge graph E and choose large connected components of at least size R. The edge graph E has vertices consisting of the coordinates in A, and the horizontal and vertical edges are set only if the correspondence field is smooth, i.e. the field vector has changed less than a threshold. This gives us large corresponding regions between the two images, which we call "islands." In the space-time region we are filling in between A and B, we use the corresponding "island" regions as hard constraints, and they are constrained to animate with constant linear velocity. Note that it may be objectionable if a keyframe changes from the top to the bottom row (or vice versa) during zooming. Therefore, we optionally run a second pass of keyframe clustering requiring contiguous sets of keyframes to be even numbers.

Retargeting. Next, given the constraints, we fill in the unconstrained parts of the space-time volume, shown in magenta in Figure 6.5-left. We begin at the larger image B, and iteratively retarget down to smaller resolutions until we reach the size of A. For each resolution, we repeatedly run the summarization algorithm [98] to retarget down the current image, with the source image S in Equation (6.1) as the larger tapestry B. We omit the $d_{complete}$ term from the objective, because the input domain is changing as we transition to the next discrete scale. We also accelerate the process by retargeting only at the finest scale.

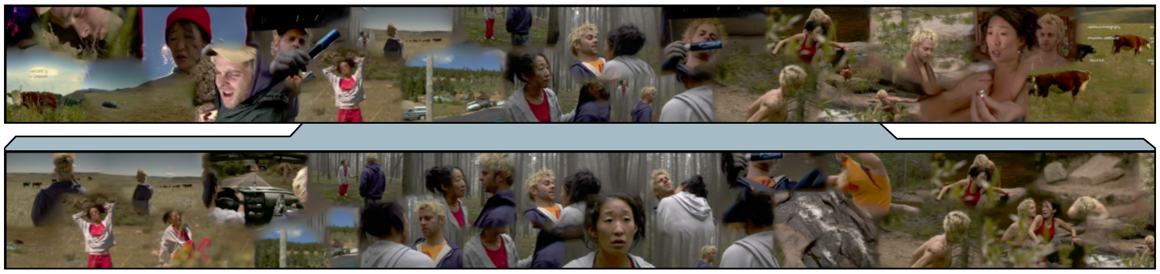
Boundary conditions. Because our initial guess begins at B, the boundary condition at the bottom of the space-time volume is implicitly enforced. However, we also need to ensure that the animation sequence ends at the smaller tapestry A. To accomplish this we use as our input domain S a weighted combination of patches from A and B, linearly interpolating the weighting factor so the animation converges to A at the last frame. We also grow the islands by inflating them as we approach A in the space-time volume.

An example of a temporally coherent space-time interpolation is shown in Figure 6.5-right. As in the previous section, we compute this online by using overlapping tiles.

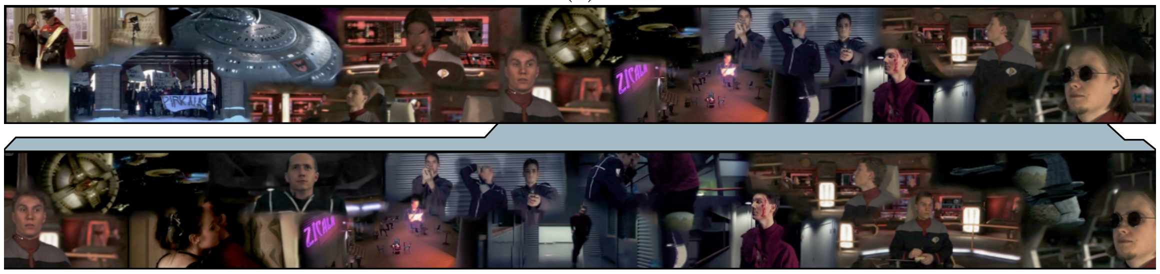
6.5 Implementation and Results

Results showing discrete tapestry scales are shown in Figures 6.1 and 6.6. The continuous zoom operation is shown in the video accompanying our paper [9].

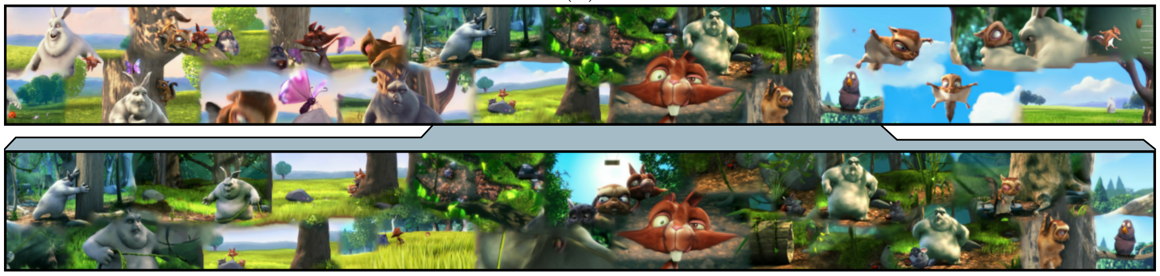
The running time and space used by our algorithm varies according to the level of approximations made. The clustering preprocess for a short film (10 mins, sampled at 2fps) takes roughly 75 minutes: 15 minutes for the actual clustering and 60 minutes to compute the square affinity matrix. For clustering frames in full-length films, the computation of the affinity matrix dominates the running time. As an approximation, the entire affinity matrix need not be computed, and only a band



(a)

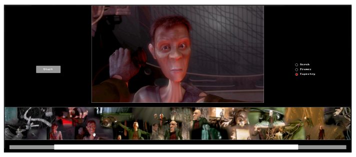


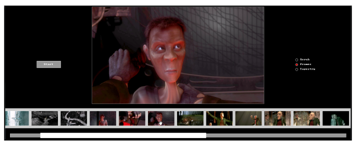
(b)



(c)

Figure 6.6: Discrete tapestries for various films: (a) *Kind of a Blur* (© Skunk Creek Productions), (b) *Star Wreck* (Energia Productions, Creative Commons license) and (c) *Big Buck Bunny* (Blender Foundation, Creative Commons license). The lines between each zoom level indicate the extent of the corresponding regions at each scale.


(a) Tapestries


(b) Keyframe filmstrip

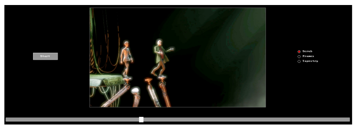

(c) Scrollbar

Figure 6.7: Three interfaces presented to users in a study. Footage from *Elephants Dream*, courtesy of The Blender Foundation

within a threshold distance of the main diagonal need be computed, as argued in Section 6.4.1. Nevertheless, even with this acceleration, precomputation of the affinity matrix requires 6 hours for a 75 minute film. Note that the clustering process is optional and can be omitted, or could be replaced with a more efficient process.

Once the keyframes are chosen, high resolution high quality tapestries can be computed offline. Each discrete tile (500 pixels wide) takes about 10 seconds to compute. The continuous zoom interpolation uses tiles of the same size, and it takes roughly another 10 seconds to compute the zoom animation between scales. Alternatively, the interface can be run at interactive rates at a somewhat lower resolution and quality. Our interactive implementation computes 500 pixel-wide tiles of the discrete tapestry in about 0.5 seconds each, and each zoom tile can be computed in about 2 seconds. As described previously, to avoid pauses in the user interface, we pre-fetch tiles in the neighborhood of the user's current scale space region. Therefore the user can move around the scale space interactively. We show in the video accompanying our paper [9] that due to the interactive rendering rates, the user can change keyframes and an updated tapestry can be recomputed interactively.

6.6 User Study

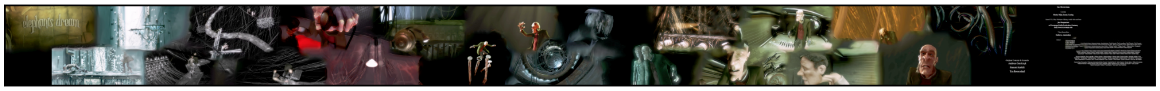
To evaluate the effectiveness of our approach we performed a user study focusing on the task of finding an event in a familiar film. We invited 22 individuals familiar with the films *Star Wars Episode IV: A New Hope* (1977) and *The Matrix* (1999) to find events with a visual and a temporal component, such as "the first appearance of Luke Skywalker" or "the last appearance of Obi-Wan Kenobi." Each participant was trained to use the three different interfaces shown in Figure 6.7: Our tapestries, a keyframe filmstrip, and a simple scrollbar. In all three cases, a preview window above the interface showed the frame corresponding to the current x location of the mouse in the interface. The training and practice session used a third film, *The Lord of the Rings: The Fellowship of the Ring* (2001). We first asked users to find events in *Star Wars* using each of the three interfaces. We next tested the importance of the zoom animation by switching it off for some tasks, replacing it with an instantaneous transition between discrete tapestries. In the final section of the study, users freely chose their favorite of the three interfaces when finding events in *The Matrix*. Users were quizzed on their familiarity with the events in *Star Wars* both before and after the study, and were given a post-study questionnaire to evaluate different interfaces.

During the study we timed the performance on each task, but found no statistically relevant difference in task performance: The average task time was about 30 seconds, but varied dramatically between individuals, probably because of differing familiarity with the subject material and comfort with video browsing interfaces in general. (In a previous pilot study we included a fourth interface consisting of DVD chapter menus and fast-forward/rewind controls, but task times for this interface were consistently 3 to 4 times longer than the others, so we omitted it from the final study to limit user frustration.)

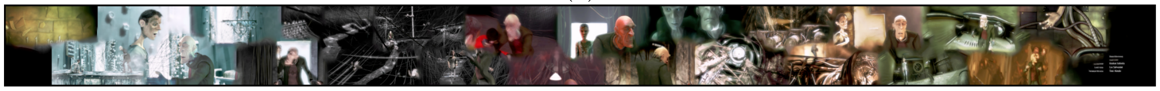
However, users strongly preferred tapestries over other interfaces, both by their own evaluation and by their preferences when given free choice of interface:

- 95% of participants (21 of 22) ranked tapestries as the most aesthetically pleasing of the three interfaces, while one participant ranked the keyframe interface first.
- 68% of participants (15 of 22) found tapestries the easiest to use for the event-finding task, 23% (5 of 22) ranked keyframes the easiest, and 9% (2 of 22) ranked tapestries and other interfaces tied for ease-of-use.
- 73% of participants (16 of 22) preferred using tapestries with the continuous zoom animation instead of discrete jumps.
- 73% of participants (16 of 22) used tapestries more than other interfaces when given their choice of interface.
- 73% of participants (16 of 22) preferred tapestries as their overall favorite interface of the three. 18% (4) preferred keyframes, and 9% (2) ranked frames and tapestries as tied.

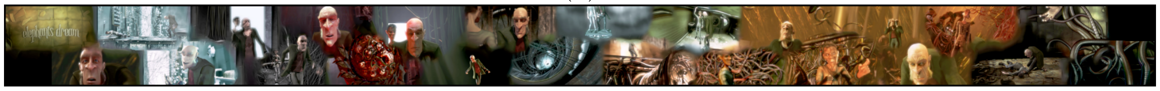
Users also gave valuable feedback for improving our interface for zooming through scale space. Some users who were most familiar with the subject material and with video browsing felt that the zoom animation was too lengthy, slowing down their search time. The zoom transition time used in our study was 1.2 seconds, but several works suggest that shorter zoom transitions between 0.3 and 1.0 seconds are more appropriate. Thus, the results shown in the video use shorter zoom transitions of .6 seconds.



(a)



(b)



(c)

Figure 6.8: Comparison of key frame selection methods for the film Elephants Dream: (a) Uniform sampling. Note that many frames are dark or otherwise contain little information. (b) Clustering. The visual information is denser and less redundant. For example, the end credits have been reduced to a single cluster, while the bright gray part on the left was expanded to include the two dominant characters (missed by the uniform sampling). (c) Manual selection. The information density is yet higher, and the characters' facial expressions are highlighted.

6.7 Discussion and Future Work

In each of the stages of our algorithm, we attempt to preserve all of our four criteria described in Section 6.1.1. In keyframe selection, BDS is optimized under the constraints that entire frames are accepted or rejected, while chronological ordering and continuity between scales are maintained using subset constraints. In static tapestry generation, our system again optimizes for bidirectional similarity, applying an additional temporal distance term to loosely enforce chronological ordering. Here scale-space continuity is not explicitly enforced, but rather inherited implicitly from the subset constraints of keyframe selection. Finally, when constructing zoom animations, the continuity constraint is enforced by constructing and interpolating "islands," the coherence term from BDS is optimized to fill in the remaining space, and chronological ordering is inherited implicitly from the static tapestries.

It may seem to be a deficiency that our framework has no explicit cut detection preprocess as in many other common video analysis techniques. However, in professionally edited film, most cuts are intended to be invisible [80], so we place no special significance on the location of a cut: only the similarity between frames matters. For example, if a scene contains a rapid series of shots alternating between two characters, it is plausible to summarize it using as few as one keyframe for each character, rather than a keyframe for each shot.

Keyframe selection. Our keyframe selection algorithm offers a simple theoretical solution that optimizes the same energy function as discrete tapestry synthesis. However, our system does not depend on using this keyframe selection mechanism, and indeed other keyframe selection methods may be advantageous for specific applications. For example, uniform keyframe selection offers the twin advantages of speed – no precomputation is necessary – and a roughly linear mapping from the x coordinate of the tapestry to time. Manual keyframe selection offers the advantages of human cognition and aesthetics to choose visually appealing and informative keyframes. Figure 6.8 shows a comparison of three tapestries generated from same source content but with three different keyframe selection mechanisms.

Failure cases. Although the bidirectional similarity synthesis algorithm eliminates and blends redundant visual content, it often happens that two adjacent keyframes in an initial layout have significantly different visual contents. In such cases, the algorithm fails gracefully by creating a blurred and feathered stitch along their boundary. Our method uses face detection as a preprocess: When this fails we can see some faces squashed in the final tapestry. But the cost of false positives is low, so we can use an extreme portion of the ROC curve to estimate face regions, and the only artifact is that some non-face regions may fail to be compacted. Nonetheless, the face detector may fail to fire on unusual faces (i.e. robots, bearded faces, cartoon faces). Face detection is not the only option for preserving structures: our method supports external identification of other important objects or regions within the video that could be supplied either manually or using other computer vision techniques (e.g. salient region detectors and object detectors).

Another extension would be to create animated tapestries, in which the region underneath the mouse begins "playing" a short sequence or multiple sequences within the tapestry in order to visualize the dynamic content of that scene. The work of [28] achieves a similar animation effect by animating the extracted foreground objects. Other improvements to the tapestry generation could be achieved by adding a boundary compatibility term for adjacent keyframes to encourage even more contiguous tapestries.

In this work we proposed an optimization that attempts to minimize an underlying energy function by introducing several stages of approximations. In the future we would like to explore the feasibility of constructing a tapestry by minimizing a single unified objective function directly, rather than multiple successive stages with slightly different approximations.

In summary, we have presented a new video summarization technique that constructs spatially continuous and zoomable visualizations of video. We presented an objective function for such summaries and optimize this objective using a series of stages, computing the final stages in real-time. We hope that this technique will prove useful for video navigation, video editing, and the presentation of video search results.

Chapter 7

Computer Vision Applications

7.1 Introduction

Computing correspondences between image regions is a core issue in many computer vision problems, from classical problems like template tracking and optical flow, to low-level image processing such as non-local means denoising and example-based super-resolution, to synthesis tasks such as texture synthesis and image inpainting, to high level image analysis tasks like object detection, image segmentation and classification. 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. 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.

For efficiency, many common algorithms only use local or sparse correspondences. Local search can only identify small displacements, so multi-resolution refinement is often used (e.g., in optical-flow [7]), but large motions of small objects can be missed. Sparse keypoint [73, 77]

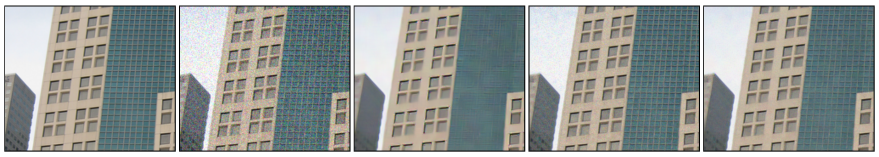


Figure 7.1: Denoising using Generalized PatchMatch. Ground truth (a) is corrupted by Gaussian noise (b). Buades et al. [21] (c) denoise by averaging similar patches in a small local window: PSNR 28.93. Our method (d) uses PatchMatch for nonlocal search, improving repetitive features, but uniform regions remain noisy, as we use only $k = 16$ nearest neighbors: PSNR 29.11. Weighting matches from both algorithms (e) gives the best overall result: PSNR 30.90.

correspondences are commonly used for alignment, 3D reconstruction, and object detection and recognition. These methods work best on textured scenes at high resolution, but are less effective in other cases. More advanced methods [20, 99] that start with sparse matches and then propagate them densely suffer from similar problems. Thus, such methods could benefit from relaxing the locality and sparseness assumptions. Moreover, many analysis applications [6, 15, 51, 89] and synthesis applications [29, 37, 43, 98] inherently require dense global correspondences for adequate performance.

Our generalized PatchMatch matching algorithm finds dense, global correspondences an order of magnitude faster than previous approaches, such as dimensionality reduction (e.g. PCA) combined with tree structures like kd-trees, VP-trees, and TSVQ. Our algorithm finds an approximate nearest-neighbor in an image for every small (e.g. 7x7) rectangular patch in another image, using a randomized cooperative hill climbing strategy.

Vision problems involve challenging matching scenarios, so we will need the full power of the generalized matching algorithm. For example, for problems such as object detection, denoising, and symmetry detection, we will wish to detect multiple candidate matches for each query patch. Thus we require the k-nearest neighbors algorithm. Likewise, for problems such as super-resolution, object detection, image classification, and tracking (at re-initialization), the inputs may be at different scales and rotations, therefore we need the variant of our matching algorithm that searches across these dimensions. Third, for problems such as object recognition, patches are insufficiently robust to changes in appearance and geometry, so we may in these cases need to match other types of image descriptors.

In this chapter we present example applications of the new algorithm to the following computer vision problems: non-local means denoising, clone detection, symmetry detection, SIFT flow, and object detection. We believe this Generalized PatchMatch algorithm can be employed as a general component in a variety of existing and future computer vision methods, and we demonstrate its applicability for image denoising, finding forgeries in images, symmetry detection, and object detection.

7.2 Related Work

When a *dense*, *global* matching is desired, previous approaches have typically employed tree-based search techniques. In image synthesis (e.g., [52]), one popular technique for searching image patches is dimensionality reduction (using PCA) followed by a search using a kd-tree [78]. In Boiman et

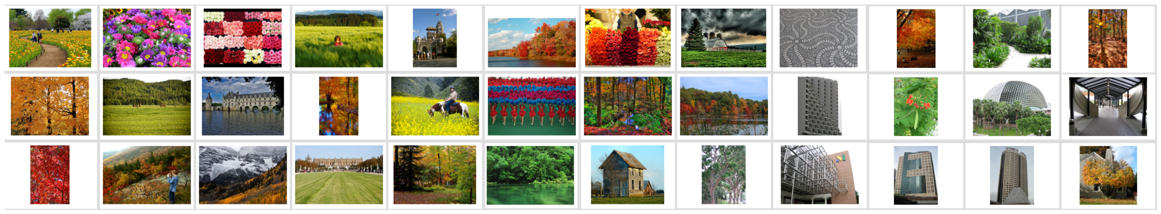


Figure 7.2: Dataset of 36 input images for denoising.

al [14], nearest-neighbor image classification is done by sampling descriptors on a dense grid into a kd-tree, and querying this tree. Other tree structures that have been employed for querying patches included TSVQ [83] and vp-trees [63]. Another popular tree structure is the k-means-tree that was successfully used for fast image retrieval [82]. The FLANN method [79] combines multiple different tree structures and automatically chooses which one to use according to the data. Locality-sensitive hashing [32] and other hashing methods can be used as well. Each of these algorithms can be run in either approximate or exact matching mode, and find multiple nearest neighbors. When search across a large range of scales and rotations is required, a dense search is considered impractical due to the high dimensionality of the search space. The common way to deal with this case is via keypoint detectors [77]. These detectors either find an optimal local scale and the principal local orientation for each keypoint or do an affine normalization. These approaches are not always reliable due to image structure ambiguities and noise. Our generalized PatchMatch algorithm can operate on any common image descriptors (e.g., SIFT) and unlike many of the above tree structures, supports any distance function, and can match across differing rotations and scales. It does this while remaining significantly faster than competing tree-based techniques. Even while the algorithm naturally supports dense global matching, it may also be constrained to only accept matches in a local window if desired.

In the remainder of this chapter, we investigate several applications in computer vision, and prior work related to those applications is mentioned therein.

7.3 Non-Local Means Denoising

For image denoising, Buades et al. [21] showed that high-quality results could be obtained by *non-local means denoising:* finding similar patches within an image and then averaging these. Subsequent work [30, 75] showed that this patch-based method could be extended to obtain state-of-the-art results by performing additional filtering steps. While Buades et al. [21] searched for similar patches only within a limited search window, Brox et al. [19] showed that a tree-based method could be used to obtain better quality for some inputs. However they do increase the distance to far away patches so searching is still limited to some local region.

Our kNN algorithm described in Chapter 3 can be used to find similar patches in an image, so it can be used as a component in these denoising algorithms. We implemented the simple method of Buades et al. [21] using our kNN algorithm. This method works by examining each source patch of an image, performing a local search over all patches within a fixed distance r of the source patch, computing a Gaussian-weighted L^2 distance d between the source and target patch, and computing a weighted mean for the center pixel color with some weight function $f(d)$.

To use our kNN algorithm in this denoising framework, we can simply choose a number of neighbors k, and for each source patch, use its k-NN in the entire image as the list of target patches. To evaluate this algorithm, we chose 36 images as our dataset (Figure 7.2). We corrupted these images by adding to each RGB channel noise from a Gaussian distribution with $\sigma = 20$ (out of 256 grey levels). If the dataset is denoised with Buades et al. (using an 11x11 search window) the average PSNR is 27.8. Using our kNN algorithm gives an average PSNR of 27.4, if the number of neighbors is small ($k = 16$). Counterintuitively, our algorithm gives worse PSNR values 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.

One solution would be to significantly increase our k. However, we found that Buades et al and our algorithm are complementary and both are efficient. Therefore, we simply run both algorithms, and list all target patches found by each, before averaging the patches under a weight function $f(d)$. We train the weight function on a single image and then evaluate on the dataset. This combined algorithm has an average PSNR of 28.4, showing that our kNN matching can improve denoising in the framework of Buades et al. The best results are obtained on images with repeating elements, as in Figure 7.1.

We also compared our results with the state-of-the-art BM3D algorithm [30]. For our dataset, BM3D produced an average PSNR of 29.9, significantly outperforming our results. However, we intentionally kept our denoising algorithm simple, and hypothesize that more advanced algorithms [30, 75] that are based on local search for speed, could do even better with our kNN algorithm.

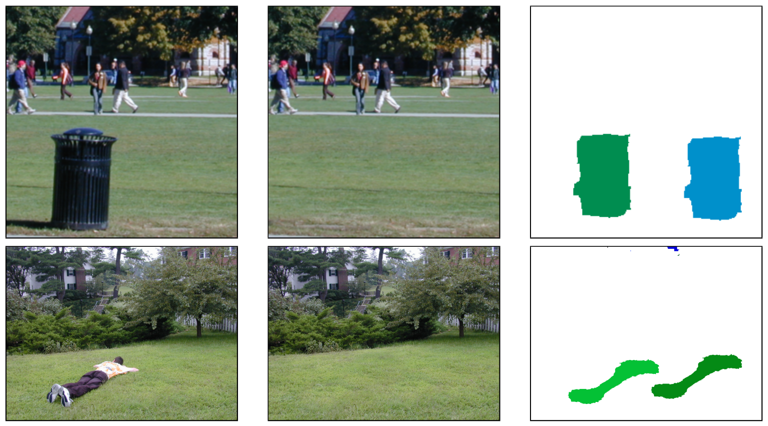

(a) original (b) forged (c) detected forgery

Figure 7.3: Detecting image regions forged using the clone brush. Shown are (a) the original, untampered image, (b) the forged image, (c) cloned regions detected by our kNN algorithm and connected components. Imagery from [89].

7.4 Clone Detection

One technique for digitally forging images is to remove one region of an image by cloning another region. For example, this can be done using Adobe Photoshop's clone brush. Such forgeries have been a concern in the popular press of late, as fake photos have been published in major newspapers, such as a missile launch photograph published in 2008 by *Agence France-Presse* where an extra missile was inserted.

Methods of detecting such forgeries have been proposed recently [11, 89]. These methods propose breaking the image into either square or irregularly shaped patches, applying PCA or DCT to discard minor variations in the image due to noise or compression, and sorting the resulting blocks to detect duplicates.

We can apply our kNN algorithm for the purposes of detecting cloned regions. Rather than sorting all blocks into a single ordered list, we can consider for each patch, its k-NN as potentially cloned candidates. We identify cloned regions by detecting connected "islands" of patches that all have similar nearest neighbors.

Specifically, we construct a graph and extract connected components from the graph to identify cloned regions. The vertices of the graph are the set of all (x, y) pixel coordinates in the image. For each (x, y) coordinate, we create a horizontal or vertical edge in the graph if its kNN are similar to the neighbors at $(x + 1, y)$ or $(x, y + 1)$, respectively. We call two lists A and B of kNN similar if for any pair of nearest neighbors $(ax, ay) \in A$ and $(bx, by) \in B$, the nearest neighbors are within a threshold distance T of each other, and both have a patch distance less than a maximum distance

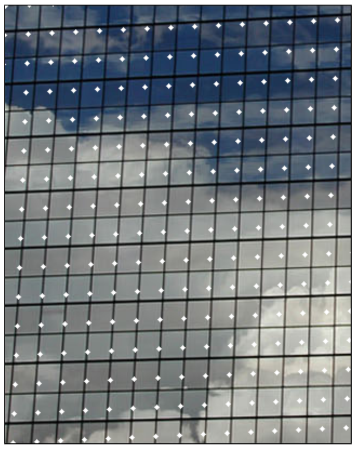


Figure 7.4: Results of symmetry detection using a regular lattice (superimposed white dots).

threshold. Finally, we detect connected components in the graph, and consider any component with an area above a minimum cloned region size C (we use $C = 50$) to be a cloned region.

Examples of our clone detection implementation are shown in Figure 7.3. Note that cloned areas are correctly identified. However, the area of the clone is not exactly that of the removed objects because our prototype is not robust to noise, compression artifacts, or feathering. Nevertheless, we believe it would be easy to adapt the algorithm to better recover the complete mask.

7.5 Symmetry Detection

Detecting symmetric features in images has been of interest recently. A survey of techniques for finding rotational and reflective symmetries is given by Park et al. [86]. Methods have also been developed for finding translational symmetries in the form of regular lattices [51].

Because our kNN algorithm matches repeated features non-locally, it can be used as a component in symmetry detection algorithms. Symmetries have been detected using sparse interest points, such as corner detectors or SIFT or edge interest points [86]. In contrast to sparse methods, our algorithm can match densely sampled descriptors such as patches or SIFT descriptors, and symmetries can be found by examining the produced dense correspondence field. This suggests that our algorithm may be able to find symmetric components even in the case where there are no sparse interest points present.

To illustrate how our method can be used for symmetry detection, we propose a simple algorithm for finding translational symmetries in the form of repeated elements on a non-deformed lattice. First we run our kNN algorithm. The descriptor for our algorithm is 7x7 patches. We calculate patch

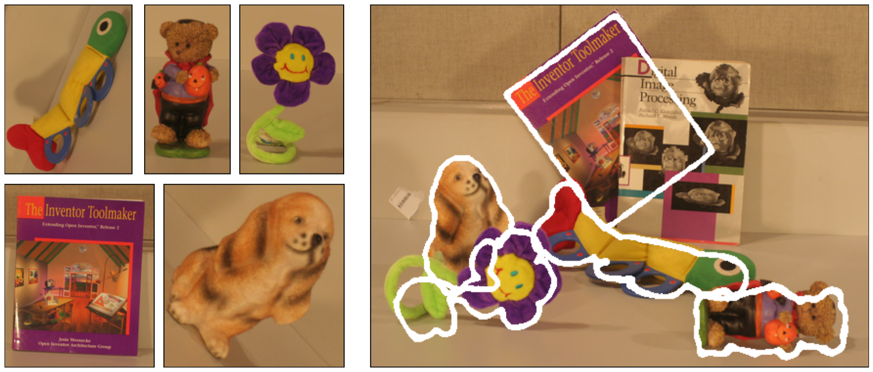


Figure 7.5: Detecting objects. Templates, left, are matched to the image, right. Square patches are matched, searching over all rotations and scales, as described in Section 3.5. A similarity transform is fit to the resulting correspondences using RANSAC.

distance using L^2 between corresponding pixels after correcting for limited changes in lighting by normalizing the mean and standard deviation in luminance to be equal. We find $k = 16$ nearest neighbors, and then use RANSAC [41] to find the basis vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ that form the lattice. We classify as inliers the coordinates where the distance between the lattice and all of the kNN is small. A result of our symmetry detection is shown in Figure 7.4.

7.6 Object Detection

Methods for object detection include deformable templates [54], boosted cascades [114], matching of sparse features such as SIFT [73], and others. Our algorithm can match densely sampled features, including upright patches, rotating or scaled patches, or descriptors such as SIFT. These matches are global, so that correspondences can be found even when an object moves across an image, or rotates or scales significantly. Provided that the descriptor is invariant to the change in object appearance, the correct correspondence will be found.

In Figure 7.5 we show an example of object detection. Similar to the method of Guo and Dyer [45], we break the template into small overlapping patches. We query these patches against the target image, searching over all rotations, and a range of scales, as per Section 3.5. A similarity transform is fit from the template to the target using RANSAC. We calculate patch distance using L^2, after correcting for lighting as we did in symmetry detection. The result is that we can find objects under partial occlusions and at different rotations and scales.

For greater invariance to lighting and appearance changes, a more complex local appearance model is needed. However it is straightforward to incorporate more complex models into our algorithm! For example, suppose we have photographs of two similar objects with different

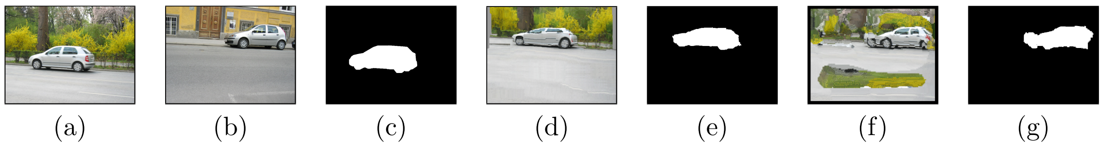


Figure 7.6: Label transfer using our method with SIFT descriptors. (a) car A; (b) car B; (c) labeled A; (d) A warped to match B using SIFT Flow [71] as well as the transferred label mask in (e); (f) A warped to B using our method and the transferred label mask in (g). Our flow is globally less smooth but can handle arbitrarily large motions.

appearance. We might wish to propagate labels from one image to the other for all similar objects and background. The SIFT Flow work [71] shows that this can be done using SIFT features correspondence on a dense grid combined with an optical-flow like smoothness term. The resulting field is solved using a coarse-to-fine approach and global optimization (belief propagation). Like most optical flow methods, SIFT Flow assumes locality and smoothness of the flow and thus can fail to align objects under large displacements. As shown in Figure 7.6, we can correctly transfer labels even when objects move a large amount. We do this by densely sampling SIFT descriptors and then matching these as described in Section 3.6.

7.7 Discussion and Future Work

This chapter applies the generalized PatchMatch algorithm to a broad range of core computer vision problems. We demonstrate several prototype examples, but many more are possible with additional machinery. For example, example-based super-resolution can use PatchMatch, using a single [44] or multiple [43] images. Section 7.6 shows an example of transferring labels using correspondences without a term penalizing discontinuity, but in other settings a neighborhood term is necessary for accurate optical flow [7, 20]. Finally, although we demonstrate object detection, our speed is not competitive with the best sparse tracking methods. It is possible that some variations of this approach using fewer iterations and downsampled images could be used to provide real-time tracking.

Chapter 8

Discussion

This thesis discusses advances we have made in matching of image and video, and related applications. We have made several contributions:

Matching algorithms motivated by statistics of natural images. We have made a statistical argument that motivates the heuristics our matching algorithm uses. We have introduced a novel matching algorithm, PatchMatch, and its generalizations, which match across arbitrary translations, rotations, scales, descriptors, and find k-nearest neighbors. Depending on which variant of our algorithm is benchmarked, it is up to an order or two of magnitude more efficient than previous techniques. We have also introduced parallel variants and an new extension, PatchWeb, to matching in large image collections.

Image editing. We have shown that our fast matching algorithm can permit new interactive applications in image completion, retargeting, and reshuffling. We have introduced novel constraints for helping the user guide the synthesis process.

Video tapestries. We have introduced a novel method of summarizing video, as "tapestries." These tapestries are continuous both spatially, with no hard borders between frames, and temporally, in that they permit a smooth zoom animation to finer detail levels of the film. Our user study shows that users prefer the tapestry representation over common alternatives, such as keyframes and scrollbar interfaces. Our optimization for tapestries uses our fast matching algorithm, permitting real-time rendering of the tapestries.

Vision applications. Using our generalized matching algorithm, we have explored a number of vision applications, including image denoising, finding forgeries in images, symmetry detection, SIFT flow, and object detection.

There are many possible areas for future work. Throughout the thesis, we have listed possible future improvements. We summarize them here for completeness. In Chapter 2, we mentioned that our statistical argument could be used to make more efficient future matching algorithms, particularly for new domains such as video where optimal search patterns differ, or by conditioning on variables such as edge responses to vary search patterns. In Section 4.1, we mentioned that more efficient GPU algorithms may be possible by finding alternatives to jump flooding. In Section 4.2.1, we suggested that enrichment could be used in cases where an image is not being matched to itself. In Section 5.4, we mentioned that different parameters or convergence criteria could be used, possibly with better GPU algorithms, to open up applications in real-time vision or video processing. Finally, in Section 6.7, for the application of video tapestries, we suggested that the failure cases could be improved, or tapestries could be animated.

We now suggest a number of higher level possible avenues for future work.

First, the algorithm itself could be broadened. This can be done trivially, by applying the algorithm as-is to 1D signals such as audio, gene sequences, or boundary matching in images, or 3D signals such as volumetric data, video, or mesh surface representations. However, it would be interesting to show that in a specific problem domain, the algorithm should be tailored to use novel search techniques based on priors for that domain. Also, because many computer vision descriptors are sampled based on sparse interest points, one could make variant of our algorithm that works on sparse instead of dense descriptors. Or even more generally, it may be possible to make a general ANN algorithm, that assumes no particular spatial embedding, using the ideas of enrichment and binning presented in Section 4.2.

As a second avenue for future work, the matching algorithm could be further optimized. We have already made low level optimizations such as SSE accelerated patch distance computation, or skipping patches with zero distance. However, at a higher level, we note that there are at least four possible matching operators: propagation, random search, enrichment, and binning. Perhaps other operators could be developed as well. In addition, the matching could be done in multi-resolution, or adaptively: regions that have converged could be identified and no longer iterated. It would be interesting to determine which combination of these techniques is most efficient, and under which parameter settings. This varies depending on the application, and even the particular input used! Furthermore, combination of these techniques could be done creatively, and with various precomputations. For example, k-coherence [109] can be thought of as combining the ideas of enrichment and k-nearest neighbors: a precomputation finds k-fold similar patches for "jumps" that will be used later in an enrichment process. The research challenge, then, is to find a sweet spot of particular effectiveness out of these various possibilities.

Finally, we suggest a number of possible future applications that could potentially use patch-based techniques: image collages, 3D geometry synthesis, video hole filling, video texture synthesis, image and video classification, real-time tracking, super resolution, paint by numbers, image analogies, and image and video compression. We could also improve our existing applications. Many of our algorithms — such as reshuffling — generate both good and bad outputs depending on user constraints. A general idea then is to generate many possible output images and automatically select the good images from this set. Alternatively, such constraints might be provided algorithmically, at least as a first guess.

In this thesis, we have presented a new algorithm, PatchMatch, and its generalizations. These permit efficient matching of dense image descriptors, potentially across various scales and rotations. The speed is up to an order or two of magnitude more efficient than previous techniques. We also presented PatchWeb, a similar algorithm that addresses scalability concerns when matching in large image collections.

Because of the wide variety of extensions and applications we have found for our algorithm, we believe it should be seen as a general purpose mathematical tool. We also believe the algorithm can be generalized and extended to many more domains. For example, we hypothesize that it could be applied to 3D geometry, or 1D signals such as found in audio, gene matching, or other areas entirely that we have not foreseen. We are excited about the future work this technique has opened up.

Bibliography

[1] S. Agarwal, N. Snavely, I. Simon, S.M. Seitz, and R. Szeliski. Building rome in a day. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 72–79. IEEE, 2010.

[2] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive digital photomontage. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 23(3):294–302, 2004.

[3] Michael Ashikhmin. Synthesizing natural textures. In *I3D Proceedings*, pages 217–226. ACM, 2001.

[4] J. Assa, Y. Caspi, and D. Cohen-Or. Action synopsis: pose selection and illustration. In *ACM Intl. Conference on Computer Graphics and Interactive Techniques*, pages 667–676, 2005.

[5] Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 26(3):10, 2007.

[6] S. Bagon, O. Boiman, and M. Irani. What is a good image segment? A unified approach to segment extraction. In *Proc. ECCV*, page IV: 44. Springer, 2008.

[7] S. Baker, D. Scharstein, JP Lewis, S. Roth, M.J. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *Proc. ICCV*, volume 5, 2007.

[8] C. Barnes, E. Shechtman, A. Finkelstein, and D.B. Goldman. PatchMatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3):24, 2009.

[9] Connelly Barnes, Dan B Goldman, Eli Shechtman, and Adam Finkelstein. Video tapestries with continuous temporal zoom. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 29(3), August 2010.

[10] Connelly Barnes, Eli Shechtman, Dan B Goldman, and Adam Finkelstein. The generalized patchmatch correspondence algorithm. In *European Conference on Computer Vision (ECCV)*, September 2010.

[11] S. Bayram, H.T. Sencar, and N. Memon. A Survey of Copy-Move Forgery Detection Techniques. *IEEE Western New York Image Processing Workshop*, 2008.

[12] P. Berkhin. *Grouping Multidimensional Data: A survey of clustering data mining techniques.* Springer, 2002.

[13] Steven Bernstein. *Film Production, Second Edition.* Focal Press, 1994.

[14] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, page 6, 2008.

[15] Oren Boiman and Michal Irani. Detecting irregularities in images and in video. In *IEEE International Conference on Computer Vision (ICCV)*, volume 1, pages 462–469, 2005.

[16] S. Boltz and F. Nielsen. Randomized motion estimation. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 781–784. IEEE.

[17] J. Boreczky, A. Girgensohn, G. Golovchinsky, and S. Uchihashi. An interactive comic book presentation for exploring video. In *Proceedings of SIGCHI*, pages 185–192. ACM, 2000.

[18] L. Bourdev and J. Brandt. Robust object detection via soft cascade. In *IEEE CVPR 2005*, volume 2, 2005.

[19] T. Brox, O. Kleinschmidt, and D. Cremers. Efficient nonlocal means for denoising of textural patterns. *IEEE Transactions on Image Processing*, 17(7):1083–1092, 2008.

[20] T. Brox and J. Malik. Large displacement optical flow. CVPR, 2009.

[21] A. Buades, B. Coll, and J. Morel. A non-local algorithm for image denoising. In *Proc. CVPR*, page II: 60, 2005.

[22] T. Chen, M.M. Cheng, P. Tan, A. Shamir, and S.M. Hu. Sketch2Photo: internet image montage. *ACM Transactions on Graphics (TOG)*, 28(5):1–10, 2009.

[23] M.M. Cheng, F.L. Zhang, N.J. Mitra, X. Huang, and S.M. Hu. RepFinder: finding approximately repeated scene elements for image editing. *ACM Transactions on Graphics (TOG)*, 29(4):1–8, 2010.

[24] P. Chiu, A. Girgensohn, and Q. Liu. Stained-glass visualization for highly condensed video summaries. In *IEEE ICME 2004*, 2004.

[25] Taeg Sang Cho, Moshe Butman, Shai Avidan, and William Freeman. The patch transform and its applications to image editing. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[26] M.G. Christel, A.G. Hauptmann, H.D. Wactlar, and T.D. Ng. Collages as dynamic summaries for news video. In *ACM Multimedia*, pages 561–569, 2002.

[27] Andy Cockburn, Amy Karlson, and Benjamin B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Comput. Surv.*, 41(1):1–31, 2008.

[28] Carlos D. Correa and Kwan-Liu Ma. Dynamic video narratives. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 29(3), 2010.

[29] A. Criminisi, P. Pérez, and K. Toyama. Object removal by exemplar-based inpainting. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2:721, 2003.

[30] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-D transform-domain collaborative filtering. *IEEE Trans. Image Processing*, 16(8), 2007.

[31] Kevin Dale, Micah K. Johnson, Kalyan Sunkavalli, Wojciech Matusik, and Hanspeter Pfister. Image restoration using online photo collections. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2009.

[32] Datar, Immorlica, Indyk, and Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *COMPGEOM: Annual ACM Symposium on Computational Geometry*, 2004.

[33] M.E. Davis. *Media streams: representing video for retrieval and repurposing.* PhD thesis, Wesleyan University, 1995.

[34] Daniel Dementhon, Vikrant Kobla, and David Doermann. Video summarization by curve simplifiation. In *ACM Multimedia*, pages 211–218, 1998.

[35] Iddo Drori, Daniel Cohen-or, and Hezy Yeshurun. Fragment-based image completion. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 22:303–312, 2003.

[36] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *ACM Transactions on Graphics (Proc. SIGGRAPH)*, pages 341–346. ACM, 2001.

[37] Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. *IEEE International Conference on Computer Vision (ICCV)*, 2:1033, 1999.

[38] M. Eisemann, E. Eisemann, H.P. Seidel, and M. Magnor. Photo zoom: high resolution from unordered image collections. In *Proceedings of Graphics Interface 2010 on Proceedings of Graphics Interface 2010*, pages 71–78. Canadian Information Processing Society, 2010.

[39] Hui Fang and John C. Hart. Detail preserving shape deformation in image editing. *ACM Transactions on Graphics*, 26(3):12, 2007.

[40] L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 524–531. Ieee, 2005.

[41] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[42] Andrew Fitzgibbon, Yonatan Wexler, and Andrew Zisserman. Image-based rendering using image-based priors. In *IEEE International Conference on Computer Vision (ICCV)*, page 1176, 2003.

[43] W.T. Freeman, T.R. Jones, and E.C. Pasztor. Example-based super-resolution. *Computer Graphics and Applications, IEEE*, 22(2):56–65, 2002.

[44] D. Glasner, S. Bagon, and M. Irani. Super-Resolution from a Single Image. In *Proc. of ICCV*, 2009.

[45] G. Guo and C.R. Dyer. Patch-based image correlation with rapid filtering. In *The 2nd Beyond Patches Workshop, in conj. with IEEE CVPR'07*, 2007.

[46] Y. HaCohen, R. Fattal, and D. Lischinski. Image upsampling via texture hallucination. In *Computational Photography (ICCP), 2010 IEEE International Conference on*, pages 1–8. IEEE, 2010.

[47] Yoav HaCohen, Eli Shechtman, Dan B Goldman, and Dani Lischinski. Non Rigid Dense Correspondence with Applications for Image Enhancement. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 30(4), 2011.

[48] P. Haghani, S. Michel, P. Cudré-Mauroux, and K. Aberer. LSH At Large–Distributed KNN Search in High Dimensions. In *International Workshop on Web and Databases (WebDB)*. Citeseer, 2008.

[49] Tim Hauser. *The Art of Wall-E.* Chronicle Books LLC, 2008.

[50] J. Hays and A.A. Efros. Scene completion using millions of photographs. *Communications of the ACM*, 51(10):87–94, 2008.

[51] J. Hays, M. Leordeanu, A.A. Efros, and Y. Liu. Discovering texture regularity as a higher-order correspondence problem. *Lec. Notes in Comp. Sci.*, 3952:522, 2006.

[52] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David Salesin. Image analogies. In *ACM Transactions on Graphics (Proc. SIGGRAPH)*, pages 327–340, 2001.

[53] D.E. Jacobs, D.B. Goldman, and E. Shechtman. Cosaliency: where people look when comparing images. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 219–228. ACM, 2010.

[54] A.K. Jain, Y. Zhong, and M.P. Dubuisson-Jolly. Deformable template models: A review. *Signal Processing*, 71(2):109–129, 1998.

[55] Micah K. Johnson, Kevin Dale, Shai Avidan, Hanspeter Pfister, William T. Freeman, and Wojciech Matusik. CG2Real: Improving the realism of computer generated images using a large collection of photographs. *IEEE Transactions on Visualization and Computer Graphics*, 2010.

[56] F. Jurie and B. Triggs. Creating efficient codebooks for visual recognition. *IEEE International Conference on Computer Vision (ICCV)*, 2005.

[57] B. Kaneva, J. Sivic, A. Torralba, S. Avidan, and W.T. Freeman. Infinite Images: Creating and Exploring a Large Photorealistic Virtual Space. *Proceedings of the IEEE*, 98(8):1391–1407, 2010.

[58] H.W. Kang, Y. Matsushita, X. Tang, X. Chen, PR Hefei, and PR Beijing. Space-time video montage. In *CVPR06*, pages 1331–1338, 2006.

[59] Kihwan Kim, Irfan Essa, and Gregory D. Abowd. Interactive mosaic generation for video navigation. In *ACM Multimedia*, pages 655–658, 2006.

[60] Nikos Komodakis and Georgios Tziritas. Image completion using efficient belief propagation via priority scheduling and dynamic pruning. *IEEE Transactions on Image Processing*, 16(11):2649–2661, 2007.

[61] Johannes Kopf, Chi-Wing Fu, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski, and Tien-Tsin Wong. Solid texture synthesis from 2d exemplars. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 26(3):2:1–2:9, 2007.

[62] W. Kraaij, A.F. Smeaton, P. Over, and J. Arlandis. Trecvid 2004-an overview. In *TRECVID video retrieval online proceedings*, 2004.

[63] N. Kumar, L. Zhang, and S. K. Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *European Conference on Computer Vision (ECCV)*, pages II: 364–378, 2008.

[64] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 24, 2005.

[65] Vivek Kwatra, Arno Schödl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 22(3):277–286, July 2003.

[66] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006.

[67] Sylvain Lefebvre and Hugues Hoppe. Parallel controllable texture synthesis. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 24(3):777–786, 2005.

[68] Anat Levin and Boaz Nadler. Natural Image Denoising: Optimality and Inherent Bounds. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[69] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 20(3):127–150, 2001.

[70] C. Liu and W. Freeman. A high-quality video denoising algorithm based on reliable motion estimation. *European Conference on Computer Vision (ECCV)*, pages 706–719, 2010.

[71] C. Liu, J. Yuen, A. Torralba, J. Sivic, and W.T.F. MIT. SIFT flow: dense correspondence across different scenes. In *Proc. ECCV*, page III: 28. Springer, 2008.

[72] Feng Liu and Michael Gleicher. Automatic image retargeting with fisheye-view warping. In *UIST*, pages 153–162. ACM, 2005.

[73] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[74] Y.F. Ma and H.J. Zhang. A model of motion attention for video skimming. In *Proc. Image Processing, Int'l Conf.*, volume 1, pages I–129–I–132, 2002.

[75] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman. Non-local sparse models for image restoration. In *Proc. of ICCV*, 2009.

[76] T. Mei, B. Yang, S.Q. Yang, and X.S. Hua. Video collage: presenting a video sequence using a single image. *The Visual Computer*, 25(1):39–51, 2009.

[77] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1615–1630, 2005.

[78] David M. Mount and Sunil Arya. ANN: A library for approximate nearest neighbor searching, October 28 1997.

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

[80] Walter Murch. *In the Blink of an Eye: A Perspective on Film Editing.* Silman-James Press, Los Angeles, 1995.

[81] E. Nardelli. Distributed k-d trees. In *Proceedings 16th Conference of Chilean Computer Science Society (SCCC96)*, pages 142–154. Citeseer, 1996.

[82] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Proc. CVPR*, volume 5, 2006.

[83] S. Niyogi and W.T. Freeman. Example-based head tracking. In *Proc. of Conf. on Automatic Face and Gesture Recognition (FG'96)*, page 374, 1996.

[84] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[85] Pau Panareda Busto, Christian Eisenacher, Sylvain Lefebvre, and Marc Stamminger. Instant Texture Synthesis by Numbers. *Eurographics Workshop on Vision, Modeling & Visualization 2010*, pages 81–85, 2010.

[86] M. Park, S. Leey, P.C. Cheny, S. Kashyap, A.A. Butty, and Y. Liuy. Performance evaluation of state-of-the-art discrete symmetry detection. CVPR, 2008.

[87] Darko Pavic, Volker Schonefeld, and Leif Kobbelt. Interactive image completion with perspective correction. *The Visual Computer*, 22:671–681(11), September 2006.

[88] J. Philbin and A. Zisserman. Object mining using a matching graph on very large image collections. In *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on*, pages 738–745. IEEE, 2009.

[89] A.C. Popescu and H. Farid. Exposing digital forgeries by detecting duplicated image regions. *Department of Computer Science, Dartmouth College*, 2004.

[90] E. Risser, C. Han, R. Dahyot, and E. Grinspun. Synthesizing structured image hybrids. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 29(4):1–6, 2010.

[91] Guodong Rong and Tiow-Seng Tan. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *I3D Proceedings*, pages 109–116, 2006.

[92] C. Rother, S. Kumar, V. Kolmogorov, and A. Blake. Digital tapestry. In *IEEE CVPR*, pages I: 589–596, 2005.

[93] Carsten Rother, Lucas Bordeaux, Youssef Hamadi, and Andrew Blake. Autocollage. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 25(3):847–852, 2006.

[94] Michael Rubinstein, Ariel Shamir, and Shai Avidan. Improved seam carving for video retargeting. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 27(3), 2008.

[95] Vidya Setlur, Saeko Takagi, Ramesh Raskar, Michael Gleicher, and Bruce Gooch. Automatic image retargeting. In Mark Billinghurst, editor, *MUM*, volume 154 of *ACM International Conference Proc. Series*, pages 59–68. ACM, 2005.

[96] E. Shechtman, A. Rav-Acha, M. Irani, and S. Seitz. Regenerative morphing. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[97] Frank Shipman, Andreas Girgensohn, and Lynn Wilcox. Generation of interactive multi-level video summaries. In *ACM Multimedia*, pages 392–401, 2003.

[98] Denis Simakov, Yaron Caspi, Eli Shechtman, and Michal Irani. Summarizing visual data using bidirectional similarity. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Anchorage, AK, USA, 2008.

[99] I. Simon and S.M. Seitz. A probabilistic model for object recognition, segmentation, and non-rigid correspondence. In *Proc. CVPR*, pages 1–7, 2007.

[100] J. Sivic, B. Kaneva, A. Torralba, S. Avidan, and W.T. Freeman. Creating and exploring a large photorealistic virtual space. In *IEEE CVPR Workshops, 2008.*, pages 1–8, 2008.

[101] J. Sivic, B.C. Russell, A.A. Efros, A. Zisserman, and W.T. Freeman. Discovering object categories in image collections. *MIT-CSAIL-TR-2005-012*, 2005.

[102] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. 2003.

[103] M.A. Smith and T. Kanade. *Video skimming for quick browsing based on audio and image characterization.* Technical Report CMU-CS-95-186, School of Computer Science, Carnegie Mellon University, 1995.

[104] MA Smith and T. Kanade. Video skimming and characterization through the combination of image and language understanding techniques. In *1997 IEEE CVPR*, pages 775–781, 1997.

[105] N. Snavely, S.M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. In *ACM SIGGRAPH 2006 Papers*, pages 835–846. ACM, 2006.

[106] Jian Sun, Lu Yuan, Jiaya Jia, and Heung-Yeung Shum. Image completion with structure propagation. In *ACM Transactions on Graphics (Proc. SIGGRAPH)*, pages 861–868, 2005.

[107] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE PAMI*, 30(6):1068–1080, 2008.

[108] Y. Taniguchi, A. Akutsu, and Y. Tonomura. PanoramaExcerpts: Extracting and packing panoramas for video browsing. In *ACM Multimedia*, pages 427–436, November 1997.

[109] Xin Tong, Jingdan Zhang, Ligang Liu, Xi Wang, Baining Guo, and Heung-Yeung Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 21(3):665–672, July 2002.

[110] A. Torralba, R. Fergus, and W.T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1958–1970, 2008.

[111] Ba Tu Truong and Svetha Venkatesh. Video abstraction: A systematic review and classification. *ACM Trans. Multimedia Comput. Commun. Appl.*, 3(1):3, 2007.

[112] T. Tuytelaars and C. Schmid. Vector quantizing feature space with a regular lattice. *IEEE International Conference on Computer Vision (ICCV)*, 2007.

[113] Shingo Uchihashi, Jonathan Foote, Andreas Girgensohn, and John Boreczky. Video manga: generating semantically meaningful video summaries. In *ACM Multimedia*, pages 383–392. ACM, 1999.

[114] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. CVPR*, 2001.

[115] Tang Wang, Tao Mei, Xian-Sheng Hua, Xueliang Liu, and He-Qin Zhou. Video collage: A novel presentation of video sequence. In *ICME*, pages 1479–1482. IEEE, 2007.

[116] Yu-Shuen Wang, Chiew-Lan Tai, Olga Sorkine, and Tong-Yee Lee. Optimized scale-and-stretch for image resizing. In *ACM SIGGRAPH Asia*, pages 1–8, New York, NY, USA, 2008. ACM.

[117] L. Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *ACM Transactions on Graphics (Proc. SIGGRAPH)*, pages 479–488, 2000.

[118] Li-Yi Wei, Jianwei Han, Kun Zhou, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Inverse texture synthesis. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 27(3), 2008.

[119] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-time completion of video. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(3):463–476, March 2007.

[120] Lior Wolf, Moshe Guttmann, and Daniel Cohen-Or. Non-homogeneous content-driven video-retargeting. In *IEEE International Conference on Computer Vision (ICCV)*, 2007.

[121] Bo Yang, Tao Mei, Li-Feng Sun, Shi-Qiang Yang, and Xian-Sheng Hua. Free-shaped video collage. *Multi-Media Modeling (MMM)*, pages 175–185, 2008.