VDOI:10.1145/2018396.2018421

# The PatchMatch Randomized Matching Algorithm for Image Manipulation

By Connelly Barnes, Dan B Goldman, Eli Shechtman, and Adam Finkelstein

**Abstract**

**This paper presents a new randomized algorithm for quickly finding approximate nearest neighbor matches between image patches. Our algorithm offers substantial performance improvements over the previous state of the art (20–100×), enabling its use in new interactive image editing tools, computer vision, and video applications. Previously, the cost of computing such matches for an entire image had eluded efforts to provide interactive performance. The key insight driving our algorithm is that the elements of our search domain—patches of image pixels—are correlated, and thus the search strategy takes advantage of these statistics. Our algorithm uses two principles: first, that good patch matches can be found via random sampling, and second, that natural coherence in the imagery allows us to propagate such matches quickly to surrounding areas. Our simple algorithm allows finding a single nearest neighbor match across translations only, whereas our general algorithm additionally allows matching of *k*-nearest neighbors, across all rotations and scales, and matching arbitrary descriptors. This one simple algorithm forms the basis for a variety of applications including image retargeting, completion, reshuffling, object detection, digital forgery detection, and video summarization.**

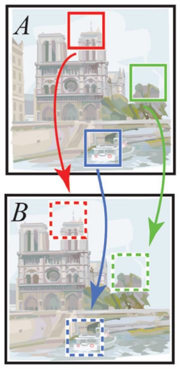
## 1. INTRODUCTION

As digital and computational photography have matured, researchers have developed sophisticated methods for analyzing and editing digital photographs and video. Many of the most powerful of these methods are *patch-based*: they divide the image into many small, overlapping rectangles of fixed size (e.g., $7 \times 7$ squares, one defined around every pixel), called patches, and then manipulate or analyze the image based on its patches. For example, patch-based techniques can be used for *image retargeting*, in which an image is resized to a new aspect ratio—the computer automatically produces a good likeness of the original image but with new dimensions. These techniques can also be used for *image completion*, in which a user simply erases an unwanted portion of an image, and the computer automatically synthesizes replacement pixels that plausibly match the rest of the image.

However, because these algorithms must search and manipulate millions of patches, performance in many cases had previously been far from interactive: operations such as image completion could previously take minutes.[24]

In this paper we describe an algorithm that accelerates many patch-based methods by at least an order of magnitude. This makes it possible to apply many powerful techniques for image editing for the first time in an interactive interface, as shown in Figure 1. We also offer intuitive controls for our image editing interface. Further, our algorithm is not limited to image editing, and can be applied to many techniques that use image patches. We report our experiences using our algorithm in object detection, digital forgery detection, and video summarization. Because our algorithm is a fairly general mathematical tool, we believe similar techniques could be used for other application domains in vision, graphics, or other fields where dense matchings are desired.

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*[15] 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 on the order of $O(M)$ or higher with relatively large constants, limiting their application for high resolution imagery.



To design an efficient search algorithm we look at the statistics of natural images—photographs of real objects—and design a good search strategy by taking

The original version of this paper is entitled “PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing” and was published in *ACM Transactions of Graphics (Proc. SIGGRAPH)*, August 2009.

Figure 1. Manipulating images using our interactive tools. 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.

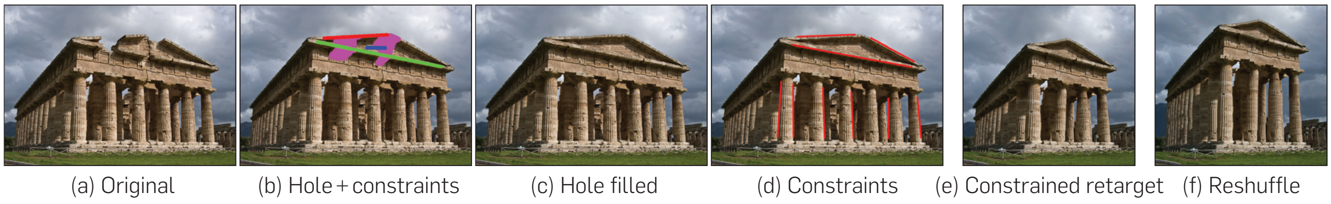


Figure 2. Given an approximate match between patches with patch distance *D*, these 2D histograms show peaked distributions of the (*x*, *y*) coordinates where better matches are located. A better initial match with lower distance *D* causes more peaking. This is averaged over a dataset of more than 100 similar and dissimilar natural image pairs. Note the center pixel is black, but this is not visible at print resolution.

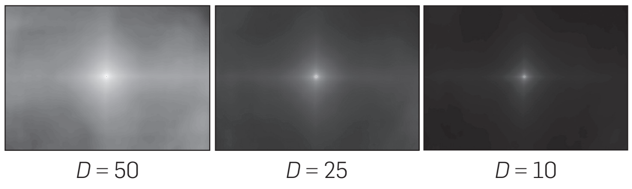
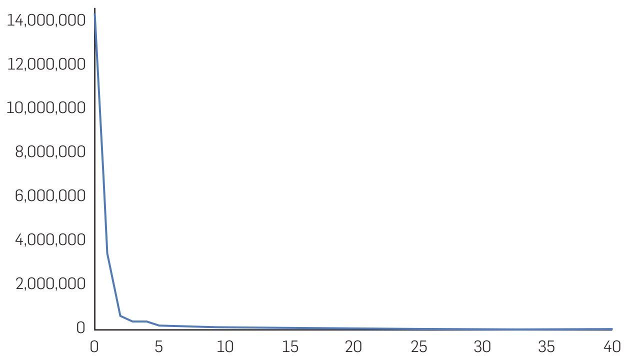


Figure 3. Histogram showing a correlation between adjacent patches' nearest neighbors. On the horizontal axis, we measure how far apart are nearest neighbors of adjacent patches, in Euclidean 2D distance. On the vertical axis, we count the number of patches with a given distance. Most patches have zero Euclidean distance, indicating perfect coherence.



advantage of these statistics. Specifically, we look at the correlation between adjacent patches, and find that they are highly correlated. For example, as shown in Figure 2, given an approximate match between patches with patch distance *D*, the locations of matches with patch distance less than *D* are not uniformly distributed throughout the image, but instead follow a peaked distribution. We can also ask in the ground truth nearest neighbor matches, for two patches that are horizontally or vertically adjacent, how far apart spatially are their matches? This is visualized as a histogram in Figure 3 which again shows a peaked distribution.

Such biased distributions allow us to devise an efficient iterative search strategy for natural images—PatchMatch—that focuses computational effort on regions most likely to produce good matches. It converges for all images in the limit, but converges extremely quickly for natural images that follow its prior assumptions. We make several 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), and dimensionality reduction methods (e.g., PCA). In contrast, our algorithm searches in the 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[1]). 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). 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 alternated at each patch: *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. Theoretical and empirical tests show the algorithm has good convergence properties for tested imagery up to 2 MP, and our CPU implementation shows speedups of 20–100 times versus *kd*-trees with PCA. Tree methods search in time $O(c_m mM)$, and incur the "curse of dimensionality:" $c_m$ is exponential in the patch dimension $m$ [15]. In contrast, our algorithm takes time $O(NmM)$, where $N$ is the number of iterations (typically 5 when searching translations and 20 for rotations and scales). In addition, unlike *kd*-trees, our algorithm uses substantially less auxiliary memory.

## 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, morphing, noise removal, super-resolution and more. We will next review some of these applications and discuss the common search techniques that they use as well as their degree of interactivity.

**Nearest neighbor search methods**. 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 algorithms only use local or sparse correspondences. Local search can only identify small displacements, so multiresolution refinement is often used (e.g., in optical-flow[3]), but large motions of small objects can be missed. Sparse keypoint[14] 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.

**Dense patch-based methods**. Those methods that find both dense and global matches have often had high time cost in the matching stage. Moreover, whereas in texture synthesis the texture example is usually a small image, in other applications such as patch-based completion, retargeting and reshuffling, the input image is typically much larger, making the search problem even more critical. Various speedups for this search have been proposed, generally involving tree structures such as TSVQ[23], *kd*-trees,[10, 24] and VP-trees,[12] each of which supports both exact and approximate search (ANN). The FLANN method[16] automatically chooses which tree algorithm to use according to the data. In synthesis applications, approximate search is often used in conjunction with dimensionality reduction techniques such as PCA,[10] because ANN methods are much more time- and memory-efficient in low dimensions. Ashikhmin[1] 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. The propagation step of our algorithm is inspired by the same coherence assumption. The *k-coherence* technique[21] combines the propagation idea with a precomputation stage in which the *k* nearest neighbors of each patch are 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. It assumes that the initial offsets are close enough that only a small number of nearest neighbors need to be searched. 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 nonlinear distance metrics, allows searching over additional continuous domains such as rotation and scale, and can be easily modified with constraints in the image space to preserve structures and enable local interactions. Locality sensitive hashing is an alternative to tree structures that can be used to search image patches,[18] but it also requires substantial additional memory and a precomputation step, unlike our algorithm.

**Matching across rotations and scales**. When search across a large range of scales and rotations is required, a dense search was previously considered impractical due to the high dimensionality of the search space. The common way to deal with this case is via keypoint detectors.[14] These detectors either find a local scale and orientation for each keypoint or do an affine normalization. These approaches are not always reliable due to image structure ambiguities and noise. Our generalized matching algorithm can operate on any common image descriptors (e.g., SIFT) and unlike many of the above tree structures, supports any distance function. Even while the algorithm naturally supports dense global matching, it may also be constrained to only accept matches in a local window if desired.

**Texture synthesis and completion**. Efros and Leung[9] introduced a simple nonparametric 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.[1, 23] The greedy fill-in order of these algorithms sometimes introduces inconsistencies when completing large holes with complex structures, but Wexler et al.[24] formulated a related problem of image completion as a global optimization, thus obtaining more globally consistent synthesis of large missing regions. This iterative multiscale 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 megapixel), 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.[22]

**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[1] gives the user control over the process by initializing the output pixels with desired colors. The Image Analogies framework of Hertzmann et al.[10] 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.[20] 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 0.5 megapixel 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.

**Image retargeting**. Many methods of image retargeting have applied warping or cropping, using some metric of saliency to avoid deforming important image regions.[25] Seam carving[2] 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.[19] 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.[22] 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.

**Image "reshuffling"** is the rearrangement of content in an image, according to user input, without precise mattes. Reshuffling was demonstrated simultaneously by Simakov et al.[19] and Cho et al.,[8] 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.

**Object detection, digital forgeries, and collages**. In addition to image editing, our algorithm can be applied to other problems in image analysis and video. For object detection, we take an approach similar to deformable template models.[11] Unlike previous approaches, we do not need to restrict our matching to sparse interest points or detect principle scales or orientations. We can instead use our matching algorithm to match all patches across all scales and orientations. For digital forgery detection, Popescu and Farid[17] previously demonstrated a method that can find regions of an image duplicated by a clone tool, by sorting image blocks after discarding JPEG compression artifacts. Our method works similarly. Our method is not robust to JPEG artifacts, but uses our more general matching algorithm, so it could potentially be generalized to find different types of forgeries such as those produced by our automatic hole filling. Finally, images can be stitched into collages[8, 19] using patch-based methods. We use this approach for our video summarization application.

## 3. MATCHING ALGORITHM

In this section we describe our core matching algorithm, which accelerates the problem of finding nearest neighbor patches by 20–100x over previous work. Our algorithm is a randomized approximation algorithm: it does not always return the exact nearest neighbor, but returns a good approximate nearest neighbor quickly, and improves the estimate with each iteration.

For brevity's sake, in this paper we present a simplified version of the algorithm that searches for only one nearest neighbor per-patch, across only two translation dimensions. There will be more discussion of extensions later.

### 3.1. High level motivation

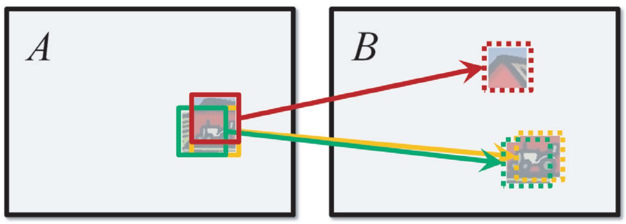
The high level intuition behind our algorithm is shown in Figure 4. We have two images $A$ and $B$ with patches visualized as colored rectangles. We wish to find for each patch in $A$ the most similar patch in $B$. We do this by taking advantage of spatial locality properties: when we have a good match, we can *propagate* it to adjacent points on the image, and if we have a reasonable match, we can try to improve it by *randomly searching* for better matches around the target position.

We define a 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 $\mathbf{b}$.[a] We refer to the values of $\mathbf{f}$ as *nearest neighbors*, and they are stored in an array whose dimensions are those of $A$.

As a reminder, our key insights are to search in the space of possible coordinate offsets, to search over adjacent patches cooperatively, and that even random coordinate assignments are likely to be a good guess for many patches over a large image.

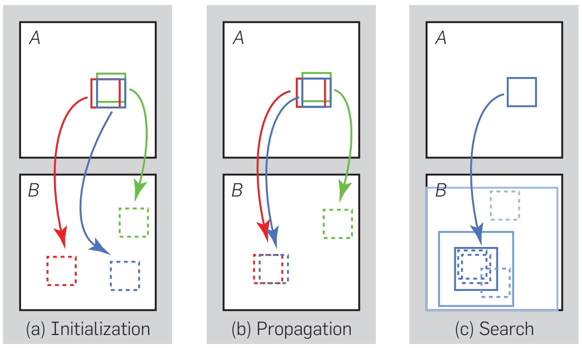
The algorithm has three main components, illustrated in Figure 5. Initially, the NNF is filled with either uniform random assignments or some prior information. Next, an iterative update process is applied to the NNF, in which good patch nearest neighbors are propagated to adjacent

**Figure 4. Illustration of the matching problem. For each patch in image *A*, we find the most similar patch in image *B*. Sometimes patches that overlap have identical or coherent matches (shown in the yellow and green matches), and sometimes the matching coordinates are nearby (the red match).**



[a] Our notation is in absolute coordinates, versus relative coordinates in Barnes et al.[6]

**Figure 5. 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.**



pixels, followed by random search in the neighborhood of the best nearest neighbor found so far. Sections 3.2 and 3.3 describe these steps in more detail.

## 3.2. Initialization

The NNF 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 $B$. The image editing applications described in Section 4 repeat the search process in a coarse-to-fine pyramid, interleaving search and reconstruction at each scale. So we have the option to use the previous solution—possibly upscaled from a coarser level of the pyramid—as an initial guess. 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 initial guess only at patches where $D$ is smaller, and then perform the remaining iterations. This gives a good tradeoff of retaining local minima found on previous iterations without getting stuck there.

## 3.3. Iteration

After initialization, we iteratively improve 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 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 offset by the same 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 $\mathbf{f}(\mathbf{z} - \Delta_p) + \Delta_p$, where $\Delta_p$ 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. If used in isolation, propagation converges very quickly, but 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 $\mathbf{v}_0$:

$$\mathbf{u}_i = \mathbf{v}_0 + w\alpha^i \mathbf{R}_i \qquad (1)$$

where $\mathbf{R}_i$ is a uniform random in $[-1, 1] \times [-1, 1]$, $w$ is a large maximum search "radius," and $\alpha$ is a fixed ratio between search window sizes. We examine patches for $i = 0, 1, 2, \ldots$ 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 $B$.

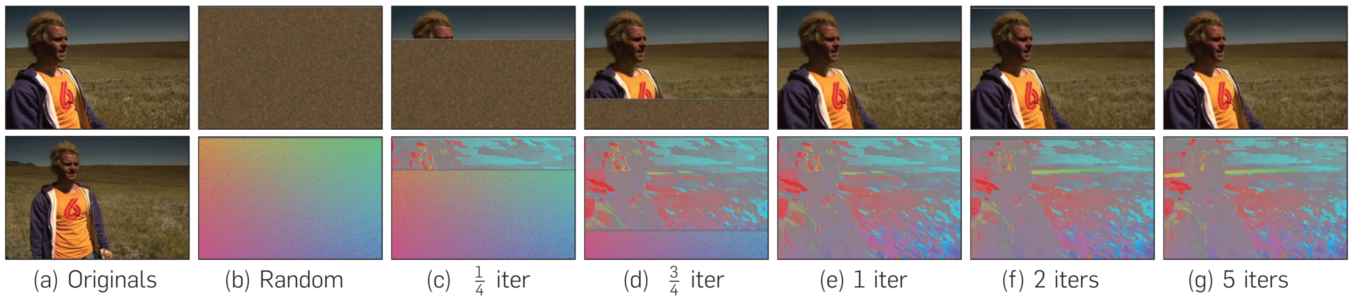
**Halting criteria**. Although different criteria for halting may be used depending on the application, in practice we have found it works well to iterate a fixed number of times. All the results shown here were computed with 4–5 iterations total, after which the NNF has almost always converged. Convergence is illustrated in Figure 6.

**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 $\mathbf{u}$, *early termination* can be used 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 $p$ 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 $D(\mathbf{f}(x, y))$.

## 3.4. Discussion

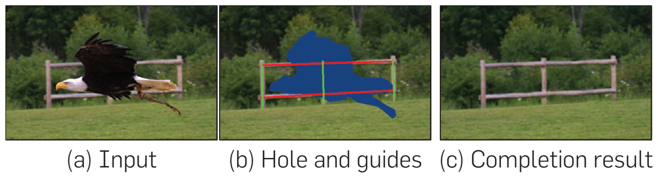
Our algorithm converges quickly to a good approximate solution in a small number of iterations. We compared our convergence with competing methods such as *kd*-trees with PCA, vp-trees with PCA, and theoretically analyzed the convergence properties of our algorithm.[6] We found that for equal matching error, and low numbers of iterations, our algorithm is 20–100x faster than the best competing algorithm, *kd*-tree with PCA, and uses substantially less memory.

**Figure 6. 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, and (g) after five iterations, almost all patches have stopped changing.**



Our algorithm bears some superficial similarity to Belief Propagation and Graph Cuts algorithms often used to solve Markov Random Fields on an image grid. However, unlike the MRF models, our algorithm does not have a neighborhood term that explicitly creates smooth or coherent matches. Because our search algorithm finds coherent regions in early iterations, our matches implicitly err towards coherence. Thus our approach is sufficient for many practical synthesis applications, while avoiding the computational expense of MRF approaches.

**Figure 7. Example 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.**



## 4. APPLICATIONS

We have presented a fast algorithm for finding good matches between patches in arbitrary images. Based on our algorithm, we have developed an interactive interface for editing images, using sophisticated patch-based synthesis techniques. Our synthesis uses the framework of Simakov et al.,[19] which was previously demonstrated on synthesis tasks such as image collages, reshuffling, retargeting, automatic cropping, and the analogues of these in video. This method works by iteratively improving a current output image, starting at a coarse resolution and proceeding to finer resolutions, in each iteration repeatedly making sure all patches in the source image are present in the current result, and vice versa. Thus at the core of this method is our correspondence algorithm, which is used to query patches in both query directions. If instead we define a region of undesired content (a "hole") to be removed from an image, the method of Simakov becomes very similar to the image completion algorithm of Wexler et al.[24] Thus, undesired objects can be removed from photographs using the same synthesis framework. However, Simakov et al. reported several minutes to generate output images. Because our object removal technique offers high quality results with low synthesis time, it is suitable for commercial deployment, and has been implemented as the new Content-Aware Fill feature in Adobe Photoshop CS5.

For many of these applications, our interactive interface allows the user to receive feedback in seconds. Many synthesis techniques are therefore interactive for the first time because of our algorithm. We also offer the user new interactive controls for guiding the output image with constraints.

We show a number of example user interactions in Figure 1. These include automatically replacing an undesired image region, as well as retargeting the image to change aspect ratio (using constraints to prevent pillars from breaking), and reshuffling, or moving up the roof of the building. In Figure 7 we show removal of an undesired object. In Figure 8 we show how some of our constraints can prevent a building from bending or breaking during the retargeting process. Because our synthesis framework works by iteratively modifying the output image to the desired resolution, features tend to bend or break slowly, so constraints can be applied during the iterative process to prevent breaks. In Figure 9 we show a local scale tool that allows a region to be scaled while preserving texture. In Figure 10 we show how many of these tools can be combined in a workflow of editing architecture.

We have also investigated additional applications such as object detection, label transfer, symmetry detection, denoising, detecting digital forgeries,[7] and video summarization.[5] We present some results for object detection, forgery detection, and video summarization here.

For many of these applications, more general variants of our matching algorithm are needed. We have developed a generalization of our matching algorithm[7] that searches over all rotations and scales (useful for object detection), finds *k* nearest neighbors instead of a single nearest neighbor[b] (useful for detecting digital forgeries), and matches

**Figure 8. 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).**

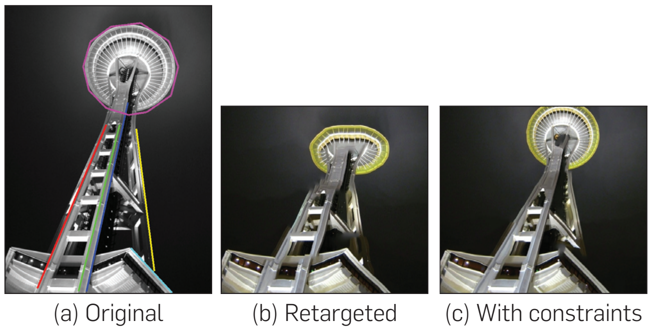
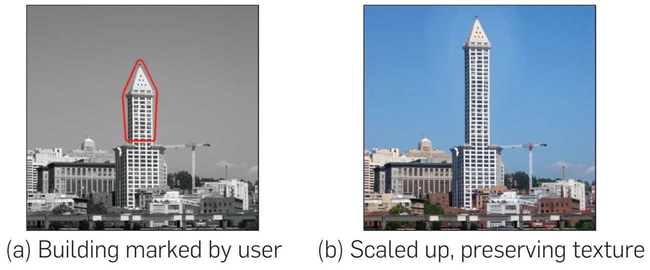


**Figure 9. Example using local scale tool. The user marks a source polygon (a), and then applies a nonuniform scale (b) to the polygon, while preserving texture.**


(a) Building marked by user
(b) Scaled up, preserving texture

arbitrary descriptors, that is, vectors computed at each pixel (useful for matching features, e.g., SIFT features that are robust to camera and lighting changes). These generalizations are simple and natural extensions of our original algorithm.

In Figure 11 we show an example of object detection. The algorithm accepts a template image to be found within a large target image. The template is then located by breaking both images into small square patches, then running our matching algorithm across all rotations and scales, with a patch descriptor that compensates for changes in lighting. Then for each template object, the resulting NNF is used to estimate the pose, after rejecting outliers due to occlusions and poor matches.

We show a second application of detecting digital forgeries made by the "clone brush" in Figure 12. When a user forges an image in this way, he or she removes an object by manually replacing it with a different region of the same image. Therefore, in the forged image some patches are duplicated in large coherent regions. We detect these by using our matching algorithm to find, for each patch, its *k*-nearest neighbors within the same image. Then we detect cloned regions by locating large regions in the NNF that are roughly coherent.

We finally present our video summarization system,[5] shown in Figure 13. This system automatically selects and collages

[b] Note that Liu and Freeman[13] also investigated *k*-NN search based on our algorithm.

**Figure 10. Modifying architecture with reshuffling. The images contain many repetitions, so the algorithm can often produce plausible output even when subject to extreme constraints.**

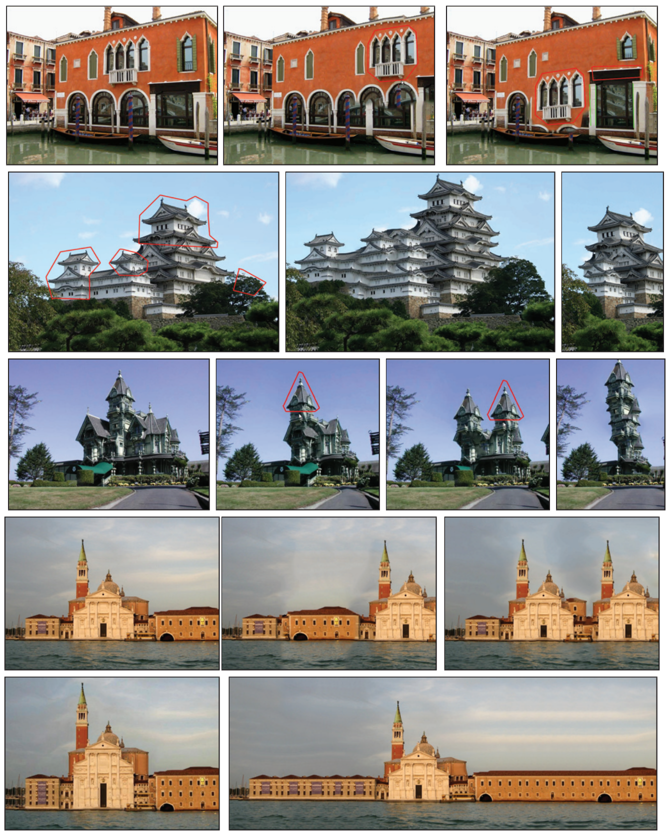
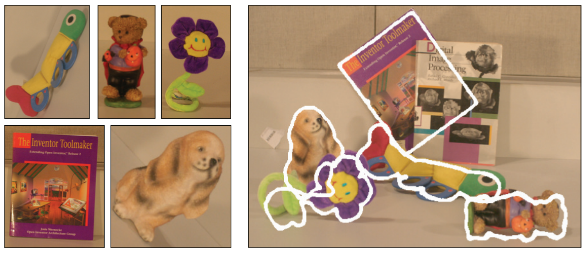


**Figure 11. Detecting objects. Templates, left, are matched to the image, right. Square patches are matched, searching over all rotations and scales.**



video frames to produce a seamlessly zoomable visual timeline. This timeline can be used as an alternative to the simple scrollbars typically used in video players. The user can select a desired scene to move to with the mouse. Or, to see more details from a given part of the film, the user can smoothly zoom in to expose more details from that part of the film. We produce our collages using patch-based synthesis, and because of the speed of our algorithm, we can produce timelines interactively.

## 5. FUTURE WORK

We believe our algorithm can be extended to different search domains such as 1D (e.g., audio) and 3D geometry, and allow

Figure 12. Detecting image regions forged using the clone brush: (a) the original, untampered image, (b) the forged image, and (c) the cloned regions detected by our algorithm. (Imagery courtesy of Popescu and Farid.[17])

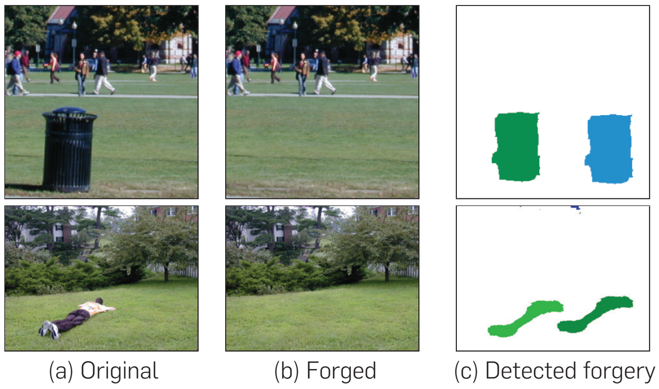


Figure 13. A multiscale tapestry represents an input video as a seamless and zoomable summary image that can be used to navigate through the video. This visualization eliminates hard borders between frames, providing spatial continuity and also continuous zooms to finer temporal resolutions. This figure depicts three discrete scale levels for the film *Elephants Dream* (Courtesy of the Blender Foundation). The lines between the scale levels indicate the corresponding domains between scales.

for other new applications such as synthesis of 3D geometry or stereo depth maps. For extensive detail on the matching algorithms, image statistics, applications, and directions for future work, consult Barnes.[4]

Our research has led us to an important conclusion about the design of image manipulation algorithms: by understanding the natural statistics of a problem domain, one can often customize a solution strategy around those statistics. In our case, by understanding the correlations between different nodes (pixels of an image), we designed the search strategy to take advantage of these statistics. We are excited about the potential of our techniques to accelerate search in many different domains, as well as the future work it has opened up.

## Acknowledgments

We would like to thank the following Flickr users for Creative Commons imagery: *Sevenbrane* (Greek temple), *Wili* (boys), *Moi of Ra* (flowers), and *Celie* (pagoda). This work was sponsored in part by Adobe Systems and the NSF grant IIS-0511965.

## References

1. Ashikhmin, M. Synthesizing natural textures. In *I3D Proceedings* (2001). ACM, 217–226.
2. Avidan, S., Shamir, A. Seam carving for content-aware image resizing. *ACM Trans. Gr. (Proc. SIGGRAPH) 26*, 3 (2007), 10.
3. Baker, S., Scharstein, D., Lewis, J., Roth, S., Black, M., Szeliski, R. A database and evaluation methodology for optical flow. In *Proceedings of ICCV*, volume 5, 2007.
4. Barnes, C. PatchMatch: A Fast Randomized Matching Algorithm with Application to Image and Video. Ph.D. thesis. Princeton University, Princeton, NJ. May 2011.
5. Barnes, C., Goldman, D.B., Shechtman, E., Finkelstein, A. Video tapestries with continuous temporal zoom. *ACM Trans. Gr. (Proc. SIGGRAPH) 29*, 3 (Aug. 2010).
6. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Gr. (Proc. SIGGRAPH) 28*, 3 (2009), 24.
7. Barnes, C., Shechtman, E., Goldman, D.B., Finkelstein, A. The generalized PatchMatch correspondence algorithm. In *ECCV*, Sept. 2010.
8. Cho, T.S., Butman, M., Avidan, S., Freeman, W. The patch transform and its applications to image editing. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.
9. Efros, A.A., Leung, T.K. Texture synthesis by non-parametric sampling. *IEEE ICCV 2* (1999), 1033.
10. Hertzmann, A., Jacobs, C.E., Oliver, N., Curless, B., Salesin, D. Image analogies. In *ACM Transactions on Graphics (Proc. SIGGRAPH)* (2001), 327–340.
11. Jain, A., Zhong, Y., Dubuisson-Jolly, M. Deformable template models: A review. *Signal Process. 71*, 2 (1998), 109–129.
12. Kumar, N., Zhang, L., Nayar, S.K. What is a good nearest neighbors algorithm for finding similar patches in images? In *ECCV* (2008), II, 364–378.
13. Liu, C., Freeman, W. A high-quality video denoising algorithm based on reliable motion estimation. In *Proceedings of the ECCV*, 2010.
14. Mikolajczyk, K., Schmid, C. A performance evaluation of local descriptors. *IEEE Pattern Anal. Mach. Intell. (PAMI) 27*, 10 (2005), 1615–1630.
15. Mount, D.M., Arya, S. ANN: A library for approximate nearest neighbor searching. Oct. 28, 1997.
16. Muja, M., Lowe, D. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications* (*VISAPP*), 2009.
17. Popescu, A., Farid, H. *Exposing Digital Forgeries by Detecting Duplicated Image Regions*. Technical Report. Department of Computer Science, Dartmouth College, 2004.
18. Shapira, L., Avidan, S., Shamir, A. Mode-detection via median-shift. In *IEEE Computer Vision and Pattern Recognition (CVPR)* (2009), IEEE, 1909–1916.
19. Simakov, D., Caspi, Y., Shechtman, E., Irani, M. Summarizing visual data using bidirectional similarity. In *IEEE Computer Vision and Pattern Recognition* (*CVPR*), Anchorage, AK, 2008.
20. Sun, J., Yuan, L., Jia, J., Shum, H.-Y. Image completion with structure propagation. In *ACM Transactions on Graphics (Proc. SIGGRAPH)* (2005), 861–868.
21. Tong, X., Zhang, J., Liu, L., Wang, X., Guo, B., Shum, H.-Y. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Trans. Gr.* (*Proc. SIGGRAPH*) *21*, 3 (July 2002), 665–672.
22. Wei, L.-Y., Han, J., Zhou, K., Bao, H., Guo, B., Shum, H.-Y. Inverse texture synthesis. *ACM Trans. Gr. (Proc. SIGGRAPH) 27*, 3 (2008).
23. Wei, L.Y., Levoy, M. Fast texture synthesis using tree-structured vector quantization. In *ACM Transactions on Graphics (Proc. SIGGRAPH)* (2000), 479–488.
24. Wexler, Y., Shechtman, E., Irani, M. Space-time completion of video. *IEEE Pattern Anal. Mach. Intell. (PAMI) 29*, 3 (2007), 463–476.
25. Wolf, L., Guttmann, M., Cohen-Or, D. Non-homogeneous content-driven video-retargeting. In *IEEE ICCV*, 2007.

**Connelly Barnes**, Princeton University, Princeton, NJ.

**Dan B Goldman**, Adobe Systems, Seattle, WA.

**Eli Shechtman**, Adobe Systems, Seattle, WA.

**Adam Finkelstein**, Princeton University, Princeton, NJ.