Image Retargeting using "Patch-Based Bidirectional Similarity"

PHANITTA CHOMSINSAP, University of California, Santa Barbara

Patch-based Bidirectional Similarity (BDS) measure is an approach to summarization of visual data based on two important properties: completeness and coherence [1]. This approach addresses various problems of previous methods of image retargeting such as image cropping and resizing, and seam carving. This report discusses the followings: (1) the algorithms used and their implementation; (2) the comparison of two search methods, naïve approach and PatchMatch algorithm [2], and their computation time; (3) the comparison of retargeting results between Patch-Based Bidirectional Similarity and Seam Carving method.

1 ALGORITHM

The steps for the entire algorithm are (1) Retargeting at the coarsest scale; (2) Gradual resolution refinement; (3) Final scale refinement. I will also explore two different methods of patch-search, naïve approach and PatchMatch algorithm, and compare their results.

1.1 Retargeting Steps

The algorithm is implemented in the FinalProj (using the given search and vote implementation) and FinalProj2 (using my search and vote implementation) functions. The image used for the implementation is SimakovFarmer.png. The goal of this algorithm is to retarget the original image of size 194 by 259 to a target size of 194 by 169.

1.1.1 Retargeting at the Coarsest Scale. First, the source image is downsampled to 35 by 47. For the gradual resizing, the goal is to gradually downsampled the target image to the size 35 by 30 (ceil(194*30/169) = 35). This is done by first initialize a matrix Tinit to be 35 by 47. At each iteration of gradual resizing, the width dimension of Tinit will be decreased by 1 pixel using imresize. The search and vote function will be used to compare the source and Tinit images, and ensure that they satisfy the coherence and the completeness requirement of BDS. This step will be repeated until Tinit reaches its target size of 35 by 30.

1.1.2 Gradual Resolution Refinement. Once the target image achieved 35 by 47 in size, it was then upsampled to obtain the target size of 194 by 169. For the gradual upscaling, the target image is upscaled by a small amount using *imresize* function. I chose this value to be 16x14 pixels which closely correspond to the ratio of target size, 194:169. At the same time that the target image is upsampled, the source image is downsampled (keeping the aspect ratio the same) and has one dimension that is equal to that of the target; in my case, the height of the target and source are the same at each iteration of upsampling. Followed by the upsampling step is the search and vote of the source and target images. This gradual resolution refinement is repeated until the target size is reached.

1.1.3 Final Scale Refinement. Search and vote algorithm is called again for the final refinement.

1.2 Search and Vote

1.2.1 Naïve Approach. This algorithm is implemented in the *nnmex_naive* function. The patch size used is 7x7. Since this approach uses a brute-force scan through all patches in both images to find the best match, the implementation contains four for loops. For each patch p1 in image A, the algorithm scans all patches p2 in image B to find its best match. Please note that the size of nn is not the same as the target image but smaller. The size of nn is set to be (h1-ps)x(w1-ps)x3, which is the same size as the dotted region where the index of the patch is allowed to be at (Fig. 1).

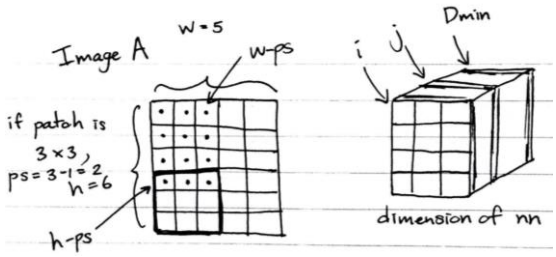

Fig. 1. This figure shows the dimension of nn to be the same size as the dotted region, where the index of the patch is within the boundary.

ALGORITHM 1: Naïve Search Algorithm

```
Initialization of variables
for i1 = 1:h1-ps
   for j1 = 1:w1-ps
      for i2 = 1:h2-ps
         for j2 = 1:w2-ps
            → get patches p1 and p2; find sum of square diff
              (Dnorm) between two patches
            → store all Dnorm values in a vector; store all index
              values of Dnorm in i2_ind and j2_ind vectors
            → find min value of Dnorm (Dmin) and update nn
              [Dmin min_ind] = min(D);
              nn(i1,j1,1) = i2_ind(min_ind);
              nn(i1,j1,2) = j2_ind(min_ind);
              nn(i1,j1,3) = Dmin;
         end
      end
   end
end
```

where h1-ps and w1-ps set the boundaries for the patch pixels; h1 and w1 are the height and weight of image A, h2 and w2 are the height and weight of image B; ps = patchSize-1.

1.2.2 PatchMatch Algorithm. This algorithm is implemented in the *nnmex_patchmatch* function. PatchMatch algorithm includes three main steps: (1) random initialization; (2) propagation; (3) random search for patch. The patch size and the dimension of nn remain to be the same as in the Naïve algorithm discussed earlier.

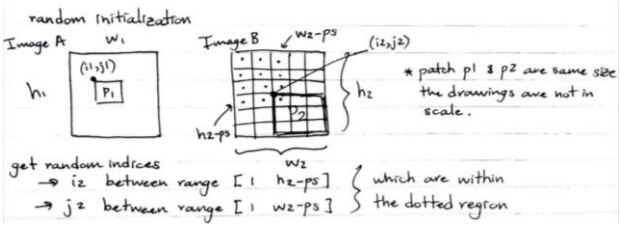

Fig. 2. Random initialization.

(1) Random initialization: For each patch p1 in image A, random indices (i,j) within the dotted region (Fig. 2) are assigned to and stored in nn, along with the sum of squared difference (Dnorm) of patches p1 and p2 at that random indices. At this point, a few patches might find a good match, while the others still have poor match. The indices of patch match, and Dnorm value are updated to nn. (Fig. 2)

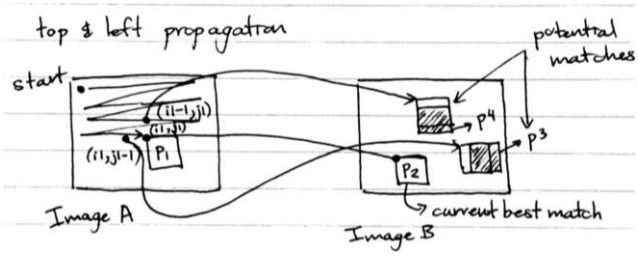

Fig. 3. Top and Left Propagation.

(2) Propagation: This is a method to improve matching patch p2 for each patch p1. Propagation is done in two ways: (1) top and left; (2) bottom and right. In top and left propagation, for each patch p1 at (i1,j1), the left and top patches at (i1,j1-1) and (i1-1,j1) have matching patches in image B (Fig. 3). For the left and top matching patch in image B, two new patches p3 and p4 are obtained by shifting the original patches right and down respectively by one pixel. By comparing Dmin of the current best patch for p2 with Dmin of p3 and p4, this improves the probability of getting better patch match. The bottom and right propagation is done similarly.

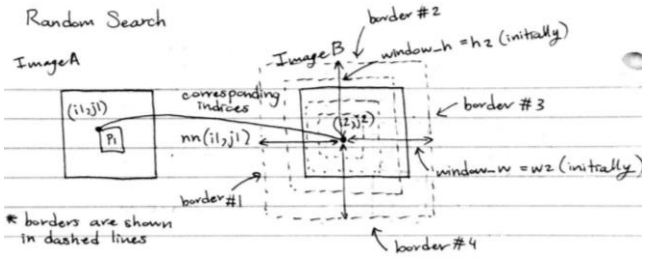

Fig. 4. Random Search.

(3) Random search for patch at (i1,j1): This method further improves the match by finding a new patch at random indices within the specified region shown in Fig. 4. The boundary is defined by the square box (dashed line) of height 2*window_h and weight 2*window_w, where the initial index (i2,j2) is at the center. The initial values for window_h and window_w are h2 and w2 respectively, and will be reduced by half at each iteration of search. For each iteration of search, a random index is sampled, and a potential patch match is found at that index. If-else statements are used to clamp the values of the boundary to the valid region, which is the region within the boundary in Fig. 4 & dotted region in Fig. 2.

ALGORITHM 2: PatchMatch Algorithm

```
Random initialization of nn for all(i1,j1)
for n = 1:niters
        for i1 = 1:h1-ps
                for j1 = 1:w1-ps
                    → propagate from top & left of (i1,j1)
                    → random search for patch at (i1,j1)
                end
        end
        for i1 = h1-ps:-1:1
                for j1 = w1-ps:-1:1
                    → propagate from bottom & right of (i1,j1)
                    → random search for patch at (i1,j1)
                end
        end
```

1.2.3 Voting. This algorithm is implemented in the my_votemex function. First a Tout matrix, where the values of patches from source will be added on, and a C matrix, a count matrix, are initialized to zero. To satisfy the coherence property of BDS, a match patch p2 (source image) is added in the position of p1 (target image) in Tout. This is done for all patches p1. Patch p2 can be obtained by using ann. At the same position, update the count matrix C by 1.

To satisfy the completeness property of BDS, first search through each index in the source image, and find the corresponding index in the target image. The position of the index can be found by using bnn. Then, add p2 (source image) to Tout at the position of p1 (target image). At the same position, update the count matrix C by 1. Lastly, average Tout matrix by using element-wise division with the count matrix C.

ALGORITHM 3: Voting Algorithm

```
Initialize a zero matrix for Tout and C
for i = 1:ann_h
        for j = 1: ann_w
                Satisfy coherence requirement
                → get index using ann, add patch p2 to Tout
                → at the same position, update count by 1

                Satisfy completeness requirement
                → get index using bnn, add patch p2 to Tout
                → at the same position, update count by 1

                → Average Tout by count matrix C
        end
end
```

2 RESULTS AND DISCUSSION

2.1 Retargeting Results

The retargeting results using the given and my implementation for search and vote are shown in Fig. 5. Speaking of my implementation (left image), although the distance between the man and the bull is greater than that of the expected results (Fig. 6. left), all the main contents (the bull, man, and the stick) are preserved with details. Although I have ensured that the aspect ratio of the image does not change, because the image dimension is not always a whole number during the resizing, this may have affected the original ratio. Another way to improve my result is to do the gradual re-targeting or upsampling more slowly.

Comparing to the left image in Fig. 5, my search and vote implementation produced a similar result in terms of the content of the image, but blurrier. This might be because my search and vote algorithms treat a colored pixel as one number by combining the RBG value together, instead of doing search and vote for each RGB layers.

Fig. 5. My retargeting results using *nnmex* and *votemex* for search and vote (left) and *nnmex_patchmatch* and *my_votemex* for search and vote (right).

2.2 Computation Time of Naïve and PatchMatch

Using the *nnmex_naive* and *nnmex_patchmatch* functions, the computation time of one iteration of search utilizing a 7x7 patch on *test_35x46.png* and *test_35x47.png* is approximately 100 and 2-3 seconds for the Naïve and PatchMatch methods respectively. Since the Naïve approach utilizes a brute-force scan to find best patch match, time complexity is much greater than the PatchMatch method, and tend to increase exponentially with an increase in image dimension. The time complexity for Naïve approach is O(n2) for an image of size nxn.

Unfortunately, I was not able to produce a retargeting result using a naïve method (due to the long computation time); however, speaking of the quality of the retargeted image, although PatchMatch method does not search for all patches but search for patches at random through propagation and random search, this method will produce a result with decent quality (comparable to that of naïve search). The quality of PatchMatch approach can be improved with more iterations, niters, in algorithm 2. Increasing the number of iterations will also increase the randomization and probability of finding a good patch match.

2.3 Comparison of Patch-based Bidirectional Similarity measure and Seam Carving method

For the example of *SimakovFarmer*, a seam carving method shows a better result. As shown in Fig. 6, the details of the background (trees, grass, etc.) are mostly preserved using the seam carving algorithm (right image), whereas the background in BDS image is more blurred out, especially in the region between the man and the bull (left image). Since seam carving utilizes the method of removing a seam (horizontal seam or vertical seam) with the lowest energy, the important contents or details with higher energy are usually preserved. In this case, image of *SimakovFarmer* does not have a lot of high-energy seam, therefore, seam carving method is a good choice. The main objects that are high-energy contents are the man and the bull.

Fig. 6. Comparison of retargeting results using Patch-based Bidirectional Similarity measure (left) and Seam Carving method[1] (right).

3 CONCLUSIONS

In conclusion, the PatchMatch algorithm for search operation provides a way to optimize the image retargeting algorithm with a faster computation time as compared to the naïve search method. For *SimakovFarmer* image, Seam Carving is shown to produced better results due to the fact that the important content of the image is sparse, meaning that the high-energy content is not clustered together.

REFERENCES

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

[2] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "PatchMatch," *ACM SIGGRAPH 2009 papers on - SIGGRAPH 09*, 2009.

[1] Using Seam Carving algorithm from HW8.