CSC320 — Introduction to Visual Computing, Spring 2019

Assignment 3: The PatchMatch Algorithm

Posted: Friday, March 1, 2019
Due: 12:00pm, Wednesday, March 20, 2019
Late policy: 15% marks deduction per 24hrs, submission not accepted if > 5 days late

---

In this assignment you will implement the PatchMatch algorithm. This algorithm is described in a paper by Barnels *et al.* and will be discussed in class or in tutorials next week. Your specific task is to complete the technique's implementation in the starter code. The starter code is based on OpenCV and is supposed to be executed from the command line, not via a Kivy GUI.

Goals: The goals of the assignment are to (1) get you familiar with reading and understanding a research paper and (partially) implementing the technique it describes; (2) learn how to implement a more advanced algorithm for efficient image processing; and (3) understand how the inefficiencies you experienced in matching patches in the previous assignment can be overcome with a randomized technique. This algorithm has since become the workhorse of a number of image manipulation operations and was a key component of Adobe's Content-Aware Fill feature. See https://research.adobe.com/project/content-aware-fill/ for more details.

Important: As in the previous assignments, you are advised to start *immediately* by reading the paper (see below). The next step is to run the starter code, and compare it to the output of the reference implementation. Unlike Assignment 2, where you implemented a couple of small functions called by an already-implemented algorithmic main loop, here your task will be to implement the algorithm itself. This requires a much more detailed understanding of the algorithm as well as pitfalls that can affect correctness, efficiency, etc. Expect to spend a fair amount of time implementing after you've understood what you have to do.

Testing your implementation on CDF: Unlike Assignment 2, it is possible to develop and run your code remotely from CDF.

Starter code & the reference solution

Use the following sequence of commands on CDF to unpack the starter code and to display its input arguments:

```
> cd ~
> tar xvfz patchmatch.tar.gz
> rm patchmatch.tar.gz
> cd CS320/A3/code
> python viscomp.py --help
```

Consult the file *320/A3/code/README_1st.txt* for details on how the code is structured and for guidelines about how to navigate it. Its structure should be familiar by now as it is similar to previous assignments. In addition to the starter code, I am providing the output of the reference solution for a pair of test images in *320/A3/code/results/jaguar2/*, along with input parameters and timings.

As I have explained in previous assignments, you should not expect your implementation to produce *exactly* the same output as the reference solution; tiny differences in implementation might lead to

slightly different results. This is not a concern, however, and the TAs will be looking at your code as well as its output to make sure what you are doing is reasonable.

320/A3/CHECKLIST.txt: Please read this form carefully. It includes information on the course's Academic Honesty Policy and contains details the distribution of marks in the assignment. You will need to complete this form prior to submission, and this will be the first file markers look at when grading your assignment.

The PatchMatch Algorithm (100 Marks)

The technique is described in full detail in the following paper *Barnes_Siggraph2009.pdf* (available here):

C. Barnes, E. Shechtman, A. Finkelstein and D. B. Goldman, "PatchMatch: A Randomized Algorithm for Structural Image Editing," *Proc. SIGGRAPH 2009.*

You should read Section 1 of the paper right away to get a general idea of the principles behind the method. The problem it tries to address should be familiar to you given that the algorithm you worked with in A2 relied on a "nearest-neighbor search" procedure for identifying similar patches for inpainting. In fact, Criminisi *et al*'s inpainting algorithm is cited in Section 2 of the paper as a motivation for PatchMatch. You should read Section 2 as well, mainly for context and background.

The algorithm you are asked to implement is described in full in Section 3. The algorithm's initialization, described in Section 3.1, has already been implemented. Your task is to implement the algorithm's basic iteration as described in Section 3.2 up to, but not including, paragraph *Halting criteria*. The starter code uses the terminology of Eq. (1) to make it easier for you to follow along.

For those of you who are more theoretically minded and/or have an interest in computer science theory, it is worth reading Section 3.3. This section is not required for implementation but it does help explain why the algorithm works as well as it does.

Sections 4 and 5 of the paper describe more advanced editing tools that use PatchMatch as a key component. They are not required for your implementation, and Section 4 in particular requires a fair amount of background that you currently don't have. Read these sections if you are interested in finding out all the cool things that you can do with PatchMatch.

Part 1. Programming Component (90 Marks)

You need to complete to implement the two functions detailed below. A skeleton of both is included in file *320/A3/code/algorithm.py*. This file is where your entire implementation will reside.

In addition to these functions, you will need to copy a few lines of code from your A1 implementation for image reading and writing that are not provided in the starter code. Like in Assignment 2, this requires no effort other than verbatim line-by-line copy from your Assignment 1 code. See *320/A3/code/README_1st.txt* for details.

Part 1.1. The *propagation_and_random_search()* function (65 Marks)

This function takes as input a source image, a target image, and a nearest-neighbor field f that assigns to each patch in the source the best-matching patch in the target. This field is initially quite poor, *i.e.,* the target patch it assigns to each source patch is definitely not the most similar patch in the target. The goal of the function is to return a new nearest-neighbor field that improves these patch-to-patch correspondences. The function accepts a number of additional parameters that control the algorithm's behavior. Details about them can be found in the starter code and in the paper itself.

As explained in the paper, the algorithm involves two interleaved procedures, one called *random search* (50 marks) and the other called *propagation* (15 marks). You must implement both, within the same function. You are welcome to use helper functions in your implementation but this is not necessary (the reference implementation does not).

The starter code provides two flags that allow you to disable propagation or random search in this function. As you develop your implementation, you can use these flags for debugging purposes, to isolate problems related to one or the other procedure.

Part 1.2. The *reconstruct_source_from_target()* function (15 Marks)

This function re-creates the source image by copying pixels from the target image, as prescribed by the supplied nearest-neighbor field. If this field is of high quality, then the copied pixels will be almost identical to those of the source; if not, the reconstructed source image will contain artifacts. Thus, comparing the reconstructed source to the original source gives you an idea of how good a nearest-neighbor field is.

Details of the function's input and output parameters are in the starter code.

Part 1.3. Efficiency considerations (10 Marks)

You should pay attention to the efficiency of the code you write. Explicit loops cannot be completely avoided in *propagation_and_random_search()* but their use should be kept to an absolute minimum. This is necessary to keep the running time of your code to a reasonable level: the input images you are supplied are quite large and it will take a *very* long time to process them if you use too many loops. No explicit loops are needed in *reconstruct_source_from_target()*.

Solutions that are no more than 50% slower than the reference implementation will receive full marks for efficiency. Less efficient implementations will have some of those points deducted depending on how much they deviate from this baseline.

Part 2. Report and Experimental evaluation (10 Marks)

Your task here is to put the PatchMatch to the test by conducting your own experiments. Try it on a variety of pairs of photos; on two adjacent frames of video; on "stereo" image pairs taken by capturing a photo of a static scene and then adjusting your viewpoint slightly (*e.g.,* a few centimeters) to capture a second photo from a different point of view. Basically, run it on enough image pairs to understand when it works well and when it doesn't.

At the very least, you must show the results of running the algorithm on the supplied source/target image pairs, using command-line arguments like those specified in the file *test_images/jaguar2/README.txt*.

Your report should highlight your implementation's results (nearest neighbor field, reconstructed source, *etc*) and discuss how well your algorithm performs, and the conditions in which it doesn't work well. The more solid evidence (*i.e.,* results) you can include in your report PDF to back up your arguments and explanations, the better.

Place your report in file *320/A3/report/report.pdf*. You may use any word processing tool to create it (Word, LaTeX, Powerpoint, html, *etc.*) but the report you turn in must be in *PDF format*.

What to turn in:

```
CHECKLIST.txt
report.pdf
algorithm.py
patchMatch.py
```

WARNING: Re: Academic Honesty

PatchMatch is a popular algorithm. C++ and OpenCV implementations are just a mouse click (or google search) away. You must resist the temptation to download and/or adapt someone else's code and submit it without appropriate attribution. This is a serious academic offence that can land you in a lot of trouble with the University.

Be aware that if an existing implementation is easy for you to find, it is just as easy for us to find as well—in fact, you can be sure that we already have found and downloaded it.