CSC320 — Introduction to Visual Computing, Fall 2022

Assignment 4: The PatchMatch Algorithm

Posted: Thursday, November 17, 2022
Due: 11:59am, ***Wednesday****, December 7, 2022 (this is NOT the usual Thursday due date)*
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 Barnes *et al.* and will be discussed in tutorial this 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.

Bonus components: I am not including any explicit bonus components. There are a number of possibilities for extending your implementation, however, that I can discuss with interested students on a one-on-one basis. This includes an iOS/Android implementation, a GPU implementation, or even implementation of editing tools described in Barnes *et al*'s paper. Please email Kyros if interested. Bonus points will be awarded depending on complexity of the implementation, etc.

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 3 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: Unlike Assignment 3, it is possible to develop and run your code remotely from Teaching Lab machines.

Starter code & the reference solution

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

```
> cd ~
> tar xvfz patchmatch.tar.gz
> rm patchmatch.tar.gz
> cd CS320/A4/code
> python viscomp.py --help
```

Consult the file *320/A4/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, along with input parameters and timings.

I am also providing a fully-featured reference solution in an encrypted format (with sourcedefender) to see how your own implementation should behave, and to make sure that your implementation produces the correct output. That being said, you should not expect your implementation to produce *exactly* the same output as the reference solution as 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.

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/A4/CHECKLIST.txt: Please read this form carefully. 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 (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 A3 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/A4/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 A3 implementation for image reading and writing that are not provided in the starter code. See the file *320/A4/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 more likely you will get full marks on your report.

Place your report in file *320/A4/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. You will be submitting the completed CHECKLIST form, your code, your written report and images. Use the following sequence of commands to pack your code and results:

```
> cd ~/CS320/
> tar cvfz assign4.tar.gz A4/code A4/results A4/report/report.pdf A4/CHECKLIST.txt
```

Upload the gzipped tarfile to MarkUs.

PatchMatch is a popular algorithm. C++, Python and OpenCV implementations are just a 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. As in previous assignments, the course policy is included below.

Academic Honesty. Cheating on assignments has very serious repercussions for the students involved, far beyond simply getting a zero on their assignment. I am very saddened by the fact that isolated incidents of academic dishonesty occur almost every year in courses I've taught. These resulted in very serious consequences for the students that took part in them. Note that academic offences may be discovered and handled *retroactively*, even after the semester in which the course was taken for credit. The bottomline is this: you are not off the hook if you managed to cheat and not be discovered until the semester is over!

You should never hand down code to students taking the course in later years, or post it on sites such as GitHub. This will likely cause a lot of trouble both to you and to the other students!

Each assignment will have a written component and most will also have a programming component. The course policy is as follows:

- **Written components:** All reports submitted as part of your assignments in CSC320 are *strictly individual work*. No part of these reports should be shared with others, or taken from others. This includes verbatim text, paraphrased text, and/or images used. You are, however, allowed to discuss these components with others at the level of ideas, and indeed you are welcome to brainstorm together.
- **Programming components (if any):** Collaboration on a programming component by individuals (whether or not they are taking the class) is encouraged at the level of ideas. Feel free to ask each other questions, brainstorm on algorithms, or work together on a (virtual or real) whiteboard. *Be careful, however, about copying the actual code for programming assignments or merely adapting others' code*. This sort of collaboration at the level of artifacts is permitted if explicitly acknowledged, but this is usually self-defeating. Specifically, you will get zero points for any portion of an artifact that you did not transform from concept into substance by yourself. If you neglect to label, clearly

and prominently, any code that isn't your own or that you adapted from someone else's code, that's academic dishonesty for the purpose of this course and will be treated accordingly.

There are some circumstances under which you may want to collaborate with someone else on the programming component of an assignment. You and a friend, for example, might create independent parts of an assignment, in which case you would each get the points pertaining to your portion, and you'd have the satisfaction of seeing the whole thing work. Or you might get totally stuck and copy one subroutine from someone else, in which case you could still get the points for the rest of the assignment (and the satisfaction of seeing the whole thing work). But if you want all the points, you have to write everything yourself. *These collaborations must be explicitly acknowledged in the* `CHECKLIST.txt` *file you submit.*

The principle behind the above policies is simple: I want you to learn as much as possible. I don't care if you learn from me or from each other. The goal of artifacts (programming assignments) is simply to demonstrate what you have learned. So I'm happy to have you share ideas, but if you want your own points you have to internalize the ideas and them craft them into an artifact by yourself, without any direct assistance from anyone else, and without relying on any code taken from others (whether at this university or from the web).

Online dissemination: Please be aware that the supplied starter code is copyrighted and that a strict non-dissemination rule applies to it. You are not to upload, email or otherwise transmit any portion of that code—modified or not—to others under any circumstances.

A final note of caution: The course's assignments cover widely-used techniques, some of which have been used in prior instalments of this course. Code for some of the assignments—and even answers to some written questions—may be just a mouse click (or a google search) away. You must resist the temptation to search for, download and/or adapt someone else's solutions and submit them as your own *without proper attribution.* Accidentally stumbling upon a solution is no excuse either: the minute you suspect a webpage describes even a partial solution to an assignment, either stop reading it or cite it in the checklist you submit. Simply put: if an existing solution is easy for you to find, it is just as easy for us to find it as well. In fact, *you can be sure we already know about it!!*