CSC320 — Introduction to Visual Computing, Fall 2022

Assignment 2: Beier-Neely Image Morphing

Posted: Monday, October 10, 2022
Due: noon, Thursday October 27, 2022
Late policy: 15% marks deduction per 24hrs, 0 marks if > 5 days late
Submission website: `https://markus.teach.cs.toronto.edu/2022-09`

In this assignment you will implement and experiment with *Beier-Neely image morphing*, a technique for morphing between two images by interactively specifying corresponding lines in them. The technique was used for the production of Michael Jackson's "Black or White" video back in 1991, and proved extremely influential. You can read more about its backstory here. In addition to completing the implementation of Beier-Neely morphing, you will also implement *bilinear interpolation* and *super-sampling*, both of which are essential for producing high-quality morphs. Esther Lin covered the technique in Tutorial#5, which is now available online on Quercus.

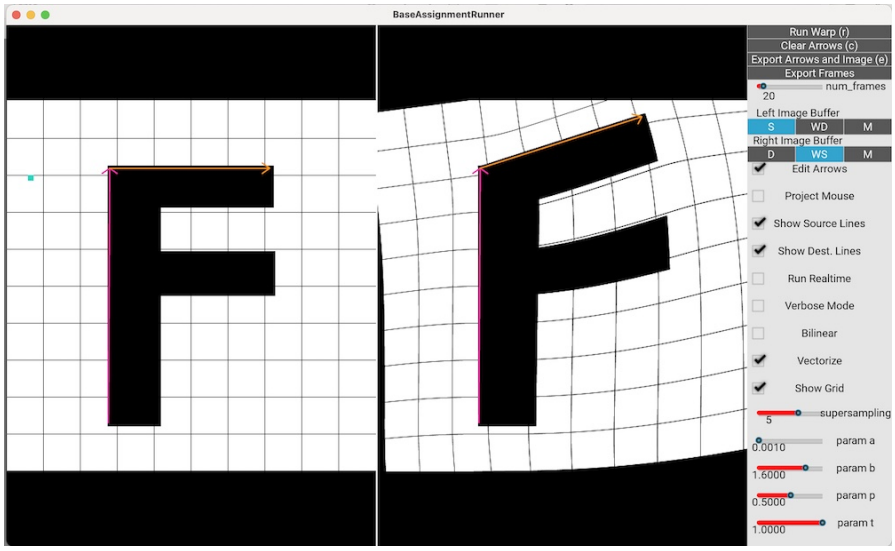

Figure 1: You will make a cool image morphing tool!

Goals: The goals of this assignment are to (1) give you a understanding of more general forms of image warping that the linear warps in A1; (2) implement interpolation and anti-aliasing and gain first-hand experience of the rather dramatic image quality improvements they provide, and (3) last but not least, read your very first computer vision paper and learn how to turn its algorithmic descriptions into actual code!

Reference solution: We are providing a reference implementation of the algorithm that you can use to compare to your own. The reference code is encrypted, so you can only run it, not see it.

Bonus: Our baseline expectation is that you will rely on for-loops for your implementation. For-loops are fine for prototyping—and you will get full marks if your implementation is correct—but they result in very, very slow code. You are welcome to take advantage of NumPy and vectorization to speed up your code; if you do so, you will receive bonus marks depending on its speed relative to

the reference solution.

Important: While the due date for the assignment is 2.5 weeks away, it is strongly advised that you read Beier and Neely's paper and go through the supplied code in the next 3-4 days, and then begin coding as soon as possible. Plan to spend some time looking at the code already supplied, as you will have to depend on it for your implementation.

Location of your code: Your implementation will wholly reside in `viscomp/algos/a2.py`. You will not need to make modifications to anything else in the codebase.

Testing your implementation: Your implementation will be tested on the teaching labs Linux machines, by running `app/a1/a2_tests.py`, possibly more tests added. Make sure you can run these tests by following the instructions below and check that your output looks fine.

Overview of Beier-Neely Image Morphing

Given two source images, $I0$ and $I1$, the user specifies a set of corresponding lines in them to establish a correspondence between "features" in one image and "features" in the other. These corresponding "line pairs" are then used to warp the input images so that pixels "follow" the lines around them. To produce an intermediate image between $I0$ and $I1$, the line pairs are linearly interpolated to create an in-between position for the lines. This is done using linear interpolation with a single interpolation parameter t. In this interpolation, $t = 0$ produces a set of lines whose position is identical to that of image $I0$ while for $t = 1$ they are identical to image $I1$. The best way to understand the technique is to try out the reference implementation. See the file `README.md` for instructions on how to do this.

Part A.1 Basic Field Warping Algorithm (50 Marks)

The main component of the morphing technique is an image warping algorithm called *Field Warping* that you will have to implement. First, read through Beier & Neely's paper to understand the overall method and the context. An important skill that you will need to develop while reading research papers such as this one is to focus only on those parts of the paper that you must fully understand to complete the assignment—you can ignore parts of the paper that require background knowledge you don't have, or you can ask me or the TAs to explain those parts of the paper in more detail. Specifically, read Sections 1 & 2 for background and motivation behind the work. The sections you are asked to fully understand and implement are Sections 3.2 and 3.3.

For this part of the assignment, you will implement the following functions:

1. `backward_mapping` (10 marks): This is the top-level function you must implement by completing the skeleton provided in the starter code. This function iterates over the destination pixels, calls the single- or the multiple-line pair Field Warping algorithm to find the source pixel and performs interpolation and/or super-sampling (if enabled) to fill the destination pixel. To implement it, you should implement the following helper functions listed below. You will receive 5 marks if you implement the basic version of this routine, without super-sampling or interpolation.
2. `calculate_uv` (5 marks): This function should implement Equations (1) and (2) from the paper.
3. `calculate_x_prime` (5 marks): This function should implement Equation (3).

4. `single_line_pair_algorithm` (10 marks): This function should implement the pseudo-code above Figure 1 in the paper, making use of functions `calculate_uv` and `calculate_x_prime`.

5. `multiple_line_pair_algorithm` (20 marks): This function should implement the second pseudo-code from the bottom right of page 37 of the paper. This can make use of the `single_line_pair_algorithm`.

Part A.2 Super-Sampling (15 Marks)

Extend your implementation of function `backward_mapping` to include super-sampling. Your implementation should densely sample the destination pixel's footprint according to the function's `supersampling` parameter; compute the floating-point coordinates in the source image that correspond to each sample; and calculate the destination pixel's value as the average of the source image's value at those coordinates. *Note that if interpolation is enabled, the value of the source image at a floating-point coordinate should be computed using bilinear interpolation*.

Part A.3 Bilinear Interpolation (15 Marks)

A reference implementation of nearest-neighbour interpolation is provided in function `interpolate_at_x` for evaluating an image at non-integer floating-point coordinates. Extend it support bilinear interpolation as well.

Part B: Experimentation (15 Marks)

As in Assignment 1, your task is to run the algorithm on your *own* data, as well as to "push" the algorithm to its limits, until image quality breaks down and artifacts begin to appear. Specifically:

1. Show *at least* one example where image quality is poor if bilinear interpolation is disabled.

2. Show *at least* one example where image quality is poor if super-sampling is disabled.

3. Save the results of running your implementation on the provided test suite.

4. Take your own photos using a smartphone camera or equivalent and create an image sequence of at least 20 frames that morphs between them. *Try to be creative.* If you cannot get your implementation to work or it is too slow, you can use the reference solution for this task.

5. All your results, including (1) the source and destination images, (2) lines employed, and (3) the parameter settings to reproduce the morphs and/or warps, should be placed in directory `app/a2/results`.

Part C: Write a lab report (10 Marks)

Your report should include your name, student number, and student username along with a discussion of your experimentation results. Specifically: (a) compare the results of your implementation and the reference implementation, pointing out differences between them and their possible causes; (b) identify shortcomings of the field field morphing algorithm, ie. instances where the algorithm produces noticeable artifacts; (c) explain the reason behind the quality improvement in items (1) and (2) of your experiments when bilinear interpolation and/or supersampling are enabled.

Your report should be in PDF format. LaTeX, Word or even powerpoint can be used to create the submitted PDF. Bear in mind, however, that the report should be in a readable form, not just a bunch of photos with annotations and/or a couple of sentences. Assume the marker knows the context of all questions, so do not spend time repeating material from this handout or from the lecture slides.

Place your report as a file `app/a2/report/report.pdf`. **Important:** Save the report as a `.pdf`, not as `.docx` or other format!

Part D: Bonus points (5-20 Marks)

Especially creative morphs will receive 5 bonus marks.

Efficient implementations that employ vectorization for field morphing, super-sampling and/or bilinear interpolation will receive bonus marks. A fully correct implementation that is at least 75% as fast as the reference implementation will receive 10 bonus marks. Vectorized implementations that also support bilinear interpolation will receive 5 bonus marks on top of that. Implementations that are faster than the reference and support bilinear interpolation will receive 20 bonus marks. *Caution:* No bonus marks will be given to implementations that are fast but not fully correct.

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 everything up.

```
cd morphing
cd ..
tar cvfz assign2.tar.gz morphing
```

Important: Make sure to unpack to check that you actually have your reports and everything in there!

Setting up your environment & tour of the code

You should first take a look at `README.md` inside the repository. This has information on setting up your environment, installing the code, and running the reference solution and your own code.

If you are using a Mac with Apple Silicon, you will need to set up your environment for python 3.9 for this assignment. This is due to a limitation of the code-encryption tool used for the reference solution. If you are unable to run python 3.9, we will provide a reference-free solution so you can continue using an earlier version of python. This limitation does not apply to other Macs or non-Mac platforms.

Freely-available resources on NumPy, OpenCV, Computer Vision Programming, etc:

1. Short NumPy tutorial by Olessia Karpova (CSC420, 2014):
   https://github.com/olessia/tutorials/blob/master/numpy_tutorial.ipynb
2. Jan Erik Solem, *Programming Computer Vision with Python*, O'Reilly Media, 2012 (preprint):
   http://programmingcomputervision.com/ (look at Chapters 1 and 10)
3. *OpenCV-Python documentation* (Intro to OpenCV, Core Operations, Image Processing in OpenCV):
   https://docs.opencv.org/3.4.13/d6/d00/tutorial_py_root.html#gsc.tab=0
4. *Matplotlib User Guide* (especially Chapter 2.1.4):
   https://matplotlib.org/3.2.2/Matplotlib.pdf
5. *NumPy Reference Guide* (especially Chapters 1.4, 1.5, 4.17):
   https://numpy.org/doc/stable/numpy-ref.pdf

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).

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!!