CSC320 — Introduction to Visual Computing, Spring 2023

Assignment 3: Image Inpainting

Posted: Tuesday, February 28, 2023
Due: 11:59am, Tuesday, March 21, 2023
Late policy: 15% marks deduction per 24hrs, submission not accepted if > 5 days late

---

In this assignment you will implement and experiment with an image inpainting tool. The tool is based on the *Exemplar-Based Image Inpainting* technique by Criminisi *el al.* and will be discussed in tutorials this week. Your specific task is to complete the technique's implementation in the starter code. The starter code is based on OpenCV and the Kivy user interface design library.

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 basic operations such as computing image gradients and curve normals; (3) learn how to assess your implementation's correctness and the overall technique's failure points; and (4) get familiar with the event-based programming model used by typical user interfaces.

Bonus components: I am not including any explicit bonus components. If, however, you feel that your implementation/extensions perform much better than the reference solution please send me an email to discuss this.

Important: You are advised to start *immediately* by reading the paper (see below). The next step is to run the reference solution (if you can) as well as the starter code, and compare the differences in how they behave (*e.g.,* their choice of patches for each iteration). As in Assignment 2, you only need to understand a relatively small part of the code you are given. Once you "get the hang of it," the programming component of the assignment should not be too hard as there is relatively little python coding to do. What will take most of the time is internalizing exactly what you have to do, and how.

Testing your implementation: There are some restrictions when using Kivy. The library relies on OpenGL and graphics cards and imposes some restrictions on what computers can be used, and how they can be used. Specifically, you will *not* be able to run a kivy-based executable remotely via ssh. Therefore, you won't be able to test it by running remotely on the CS department's linux computers. That being said, the code is fully cross-platform so if you have run it successfully on your computer it should work fine when TAs test it.

Starter code & the reference solution

Use the following sequence of commands to unpack and run the starter code:

```
> cd ~
> tar xvfz inpainting.tar.gz
> rm inpainting.tar.gz
> cd CS320/A3/code
> python viscomp-gui.py -- --usegui
```

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. In addition to the starter code, I am 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.

To run the reference solution, download the `starter_with_encrypted_reference.zip` file from Dropbox. Then, implement Part A and Part B.1.4 (copy these over from your solutions in the starter code). The code should then run using the encrypted solution code. **Note:** The encrypted reference still depends on the starter file `kivy/viscomp.kv`, which you will need to complete (see Parts A.1 and A.2) before the encrypted reference can run properly.

320/A3/CHECKLIST.txt: Please read this form carefully. It includes details on the distribution of marks in the assignment. You will need to complete this form prior to submission, and it will be the first file markers look at when marking your assignment.

Part A: Kivy-Based User Interface (15 Marks)

Part A.1 The Run button (5 Marks)

The GUI is supposed to have a clickable 'Run' button at its lower-left corner, that is currently black. Clicking on that button should run the algorithm. If the method is not successful (*e.g.,* because one of its input images is missing) a popup window should be displayed with an error message. You need to extend the starter code to add this button to the GUI, and make sure it has the correct functionality. In particular, your GUI's behavior after pressing the button should be identical to the one of the reference implementation. To do this, you will need to complete the Kivy description file (`kivy/viscomp.kv`) and the file controlling the Kivy widgets (`inpaintingui/widgets.py`).

Part A.2 The Crosshairs (10 Marks)

After loading an image and clicking on it with the left mouse button, a red crosshair should appear, along with some text that shows the image coordinates of the pixel that was clicked. In the reference implementation, the crosshairs disappear after the button is released but in your implementation they never get erased. You need extend the starter code (`kivy/viscomp.kv` and `inpaintingui/viewer.py`) to ensure the crosshairs are erased correctly.

Part B: Exemplar-Based Image Inpainting (80 Marks)

The technique is described in full detail in the following paper (included with your starter code and also available here):

A. Criminisi, P. Pérez and K. Toyama, "Region Filling and Object Removal by Exemplar-Based Image Inpainting," *IEEE Transactions on Image Processing*, vol. 13, no. 9, 2004.

You should read Sections I and II of the paper right away to get a general idea of the principles behind the method. Section II, in particular, is very important because it introduces the notation used in the rest of the paper as well as the starter code. Section III describes the algorithm in detail, with pseudocode shown in Table I. The starter code implements exactly what is shown in Table I; the only thing left for you to implement is the term $D(\mathbf{p})$ in Eq. (1). Sections IV and V are not strictly necessary to read, but they do show many results that should give you more insight into how your implementation is supposed to behave.

Part B.1. Programming Component (70 Marks)

You need to complete the implementation of three functions detailed below. A skeleton of all three is included in file *320/A3/code/inpainting/compute.py*. This file is where your entire implementation will reside.

In addition to these functions, you must also implement the associated image-loading and image-writing methods: readImage() and writeImage(). See the starter code for details on their input and output arguments, how they are called, and what values they should return.

Part B.1.1. Computing Gradients: The *computeGradient()* function (25 Marks)

This function takes three input arguments: (1) a patch $\Psi_{\mathbf{p}}$ from the image being inpainted, represented as a member of the class *PSI* from file *code/inpainting/psi.py*; (2) a binary OpenCV image F indicating which pixels have already been filled; and (3) the color OpenCV image I being inpainted. The function returns the gradient with the largest magnitude within patch $\Psi_{\mathbf{p}}$:

$$computeGradient(\Psi_{\mathbf{p}}, F, I) := \nabla \tilde{I}_{\mathbf{q}^*} \tag{1}$$

$$\text{where } \mathbf{q}^* = \arg \max_{\substack{\mathbf{q} \in \Psi_{\mathbf{p}} \\ F(\mathbf{q})>0 \\ \nabla \tilde{I}_{\mathbf{q}} \text{ is valid}}} |\nabla \tilde{I}_{\mathbf{q}}| \tag{2}$$

and all gradients are computed on the grayscale version, $\tilde{I}$, of color image I.

If image I was a regular image with no "missing" pixels, implementing this function would be trivial with OpenCV: you would just need the OpenCV function that converts color images (or image patches) to grayscale, and the OpenCV function that computes the horizontal and vertical components of the gradient.

The complication here is that not all pixels $\mathbf{q}$ in $\Psi_{\mathbf{p}}$ have been filled. As a result, applying those OpenCV functions will produce estimates of $\nabla \tilde{I}_{\mathbf{q}}$ that are incorrect/invalid for some pixels $\mathbf{q}$ in that patch. Your main task in implementing *computeGradient()* will therefore be to find a way to ignore those pixels so that the max operation in Eq. (2) is not corrupted by these invalid estimates.

Efficiency considerations: You should pay attention to the efficiency of the code you write but you will not be penalized for correct, yet inefficient, solutions. *Hint:* The reference implementation is 10-12 lines of code and includes no explicit looping over pixels.

Part B.1.2. Computing Curve Normals: The *computeNormal()* function (30 Marks)

Much like the previous function, this function also takes three input arguments and returns a 2D vector. The function's arguments are (1) a patch $\Psi_{\mathbf{p}}$ from the image being inpainted; (2) a binary OpenCV image F indicating which pixels have already been filled; and (3) a binary OpenCV image that is non-zero only for the pixels on the fill front $\delta\Omega$. The function assumes that the patche's center $\mathbf{p}$ is on the fill front and returns the fill front's normal $\mathbf{n}_{\mathbf{p}}$ at that pixel.

You are free to use whatever method you wish for estimating the curve normal. This includes using finite differences between $\mathbf{p}$ and its immediate neighbors on the fill front to estimate the tangent and normal at $\mathbf{p}$; fitting a curve to the fill front in the neighborhood of $\mathbf{p}$ and returning its normal at $\mathbf{p}$; and using any built-in OpenCV functions you wish for parts of these computations.

Hint: Depending on how it is implemented this function may involve a couple dozen lines of code, but can potentially be much less (and, of course, much more).

Part B.1.3. Computing Pixel Confidences: The *computeC()* function (10 Marks)

This function takes three input arguments and returns a scalar. The function's arguments are (1) a patch $\Psi_{\mathbf{p}}$ from the image being inpainted; (2) a binary OpenCV image F indicating which pixels have already been filled; and (3) an OpenCV image that records the confidence of all pixels in the inpainted image I. It returns the value of function $C(\mathbf{p})$ shown on page 4 of the paper, *i.e.*, it assumes that the confidence of unfilled pixels is zero and returns average confidence of all pixels in patch $\Psi_{\mathbf{p}}$.

Hint: The reference implementation is less than 5 lines of code and includes no explicit looping over pixels.

Part B.1.4. The *readImage()* and *writeImage()* functions (5 Marks)

The image-reading and image-writing functionality is provided by these two functions located in *code/inpainting/algorithm.py*. You need to implement these functions before any image processing can take place.

Part C.1. Experimental evaluation (5 Marks)

Your task here is to put the inpainting method to the test by conducting your own experiments. Specifically, you need to do the following:

1. Run your inpainting implementation for 100 iterations on the test image pairs *(input-color.jpg, input-alpha.bmp)*, and *(Kanizsa-triangle-tiny.png, Kanizsa-triangle-mask-tiny.png)*. Save the partial results to directory *320/A3/report* using the following file naming convention: *X.inpainted.png, X.fillFront.png, X.confidence.png, X.filled.png* where X is either *input* or *Kanizsa*.
2. Capture two photos, *Source1* and *Source2*, with your own camera. Any camera will do—cellphone, point-and-shoot, action camera, etc. Be sure to reduce their size to something on the order of 300×200 pixels or less so that execution time is manageable (JPEG images are OK for this assignment).
3. Create binary masks, *Mask1* and *Mask2*, to mask out one or more elements in these photos. Use any tool you wish for this task.
4. **Important:** Your source photos and masks should *not* be arbitrary. You should choose them as follows: (a) the region(s) to be deleted should not be just constant-intensity regions, which are trivial to inpaint; (b) the pair *Source1,Mask1* should correspond to a "good" case for inpainting, *i.e.,* should be possible to find a patch radius that produces an inpainted image with (almost) no obvious seams or other visible artifacts; and (c) the pair *Source2,Mask2* should correspond to a "bad" case for inpainting, *i.e.,* it should not be possible to find a patch radius that produces an inpainted image free of very obvious artifacts.
5. Run your inpainting implementation on these two input datasets. Save the two sources, masks and inpainting results to directory *320/A3/report* using the file names *Source1.png, Mask1.png, Source1.inpainted.png* and *Source2.png, Mask2.png, Source2.inpainted.png*.

Part C.2. Your PDF report (10 Marks)

Your report should include the following: (1) why you think the pair *(Source1,Mask1)* represents

a good case for inpainting; (2) why you think the pair *(Source2,Mask2)* represents a bad case for inpainting; (3) discussion of any visible artifacts in the results from these two datasets (*i.e.,* what artifacts do you see and why you think they occured).

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: You will be submitting the completed CHECKLIST form, your code, your written report and images. Use the following sequence of commands on to pack everything up:

```
> cd ~/CS320/
> tar cvfz assign3.tar.gz A3/CHECKLIST.txt A3/code A3/report
```

Important: After uploading the tarfile to MarkUS, download it from MarkUS, unpack it and **verify that your code and all other assignment components are there!** Students in previous instances of the course have accidentally submitted the starter tarfile instead of their own code and results, leading to major issues with marking—and wasting a lot of time for all involved. Because of this, there will be **no accommodation** for such errors this term: if you submit the starter code instead of your own code and/or submit no results, you will receive a zero on those parts of the assignment; if you discover this error a day later and want to re-submit, the standard late submission policy will apply. As stated on the course website, the course's late policy includes a brief grace period to allow you to quickly resubmit your tarfile a few minutes after the submission deadline without any penalty if you discover omissions in your submitted tarfile.

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!!*