CSC320 — Introduction to Visual Computing, Winter 2019

Assignment 1: Triangulation Matting

Posted: Wednesday, January 9, 2019
Due: noon, Wednesday, January 30, 2019
Late policy: 15% marks deduction per 24hrs, 0 marks if > 5 days late

---

In this assignment you will implement and experiment with Smith and Blinn's triangulation matting technique to be discussed in class next week. Your task is to implement a command-line version of the technique using Python, Numpy and the OpenCV library.

Goals: The goals of the assignment are to (1) get you familiar with OpenCV and NumPy; (2) learn how to formulate and solve linear systems of equations involving image measurements; (3) experiment with a very basic image analysis and manipulation technique; and (4) learn how to assess the technique's failure points.

Bonus components: The assignment also includes a couple of "bonus" questions for those of you who want to dig a bit deeper into either the programming or on the conceptual side of the technique. You should only tackle these components if you are confident with the rest of your submission.

Important: While the due date for the assignment is almost 3 weeks away, you are strongly advised to start immediately. It will take quite some time to familiarize yourself with OpenCV, Numpy and the starter code. 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 time is internalizing exactly what you have to do, and how.

Triangulation Matting (100 Marks)

The technique is covered in Week 2's lecture slides and is described in full detail in the following paper (included on website under readings):

A. R. Smith and J. F. Blinn, "Blue Screen Matting," Proc. ACM SIGGRAPH, pp. 259–268, 1996.

You are strongly encouraged to read the paper's first few pages (up to page 262) to understand the technique's broader context. Those pages, however, are not required for implementing the technique. You should also read the the Week 2 lecture slides now in order to get started and to have questions/clarification requests ready when the technique is discussed in class.

Part 0. Starter code & the reference solution

Use the following sequence of commands on CDF to unpack the starter code:

```
> cd ~
> tar xvfz matting.tar.gz
> rm matting.tar.gz
```

Consult the file 320/A1/partA/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 compiled, statically-linked binary format (OS X and CDF/Linux only). You should run this binary to see how your own implementation should behave, and to make sure that your implementation produces the correct output.

320/A1/CHECKLIST.txt: Please read this form carefully. It includes information on the course's Academic Honesty Policy and contains details on the distribution of marks in the assignment. You will need to complete this form prior to submission, and will be the first file markers look at when grading your assignment.

Part 1.1 The triangulationMatting() method (50 Marks)

This is the key function you must write, which implements the triangulation matting technique. The (skeleton) method is contained in file 320/A1/partA/matting/algorithm.py.

Its behavior should be as follows. The method takes as input four color images: two images of the same foreground object against two different backgrounds (called compA and compB in the starter code), and images of the backgrounds by themselves (called backA and backB, respectively). If these four input images have been loaded in memory already, triangulationMatting() computes one grayscale and one color image—alphaOut and colOut—which hold the foreground object's alpha and color values, respectively. The method returns True if it succeeds in creating these outputs and False, along with an error string, if it does not (e.g., if one or more of the four input images are not loaded already).

In addition to this method, 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.

Efficiency considerations: You should pay attention to the efficiency of the code you write, as it is easy to write highly-inefficient image analysis code with python. Solutions whose timings are significantly higher than the reference implementation (which is not optimized for speed) will lead to mark deductions.

Part 1.2. The createComposite() method (10 Marks)

This method, which you must also write, implements the matting equation. The method takes as input three images: a grayscale image alphaIn for the foreground object's alpha; a color image colIn for its color; and a background color image backIn. If these three images have already been loaded in memory, createComposite() computes computes one color composite, compOut, according to the matting equation. Return values follow the conventions of the triangulationMatting() method.

Part 2. Bonus component: Efficient implementation (up to 10 Marks)

Implementations that beat the reference solution's speed by a significant amount—without compromizing correctness—will receive up to 10 bonus marks. To get these marks you must include in your submitted report (see below) what optimizations you used, and your timings will be verified by the TAs.

Part 3. Experimental evaluation & report (25 Marks)

Your task here is to put the triangulation matting method to the test by conducting your own experiments: use your a camera you own to capture images suitable for triangulation matting, and then run the technique on them. Any camera will do—cellphone, point-and-shoot, action camera, etc. You are welcome to use the reference implementation for these experiments (i.e., you don't need to have your code fully functional in order to get started on this part of the assignment).

The main thing we will be looking for here is the degree to which you were able to (1) drive the method to its limits, and (2) understand what these limits are, i.e., the range of conditions under which we should expect the technique to work well. To that end, you should experiment with different imaging conditions—different foreground objects, different materials, different backgrounds, different light levels, etc. The specific objects and the total number of examples you show are not as important as the conclusions they help you draw. To that end, you are encouraged to try the technique on many examples and only later decide which ones to show and discuss in your report. We expect that you will need to capture at least 5-10 sets of input images to get a sense of the triangulation technique's strengths and weaknesses.

Your report (PDF format only): Write a report detailing your experiments and place it in the file 320/A1/partA/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. At the very least, your report should include the following: (1) the procedure(s) you used to capture your images; (2) the range of imaging conditions you tried; (3) an assessment of the technique's limits as explained above; (4) a set of figures showing experimental results that support your conclusions; and (5) if you think that none of your experiments were successful, explain why and describe the steps you took to overcome your difficulties.

When showing results from a specific experiment, you should include (a) the four input images; (b) details on the camera you used and its settings, (c) the alpha and color images computed by triangulation matting; (d) any composites you created from those images (only if needed to support your conclusions); and (e) zoomed-in and cropped versions of the above (only if needed to support your conclusions). In all cases, make sure that the images in your report are large enough to view comfortably on a regular printed page (e.g., Figures 1 and 2 in Smith and Blinn's paper are good examples to follow).

The directory 320/A1/partA/report/ should contain nothing other than your report. In particular, save the images you captured for your experiments and for your report but do not submit them with your code (this would take too much space). Be prepared, however, to supply them upon request by the TA or the instructor.

Part 4. Written question (15 Marks)

Apply triangulation matting to the images in 320/A1/partA/test_images/small and save the results in tif format (so that no details are lost to compression). Observe that the alpha matte on the right side of the vase has zero or near-zero intensities whereas on its left side the intensities are low but not zero. For example, expressed as an integer in the range $[0, 255]$, alpha$[829, 520] = 0$ whereas alpha$[810, 40] = 29$. The latter implies that there is a foreground "object" to the left of the vase, and that the object is not totally transparent (e.g., $\alpha = 29/255 \approx 0.11$). Why would this pixel (along with many others to the left of the vase) have non-zero alpha? Try to be as concrete as possible.

Your answer should be included in your PDF report: report.pdf

Part 5. Bonus component: Live matting (up to 10 Marks)

A major limitation of the basic triangulation matting technique is that the foreground object must be photographed sequentially against two different backgrounds while it remains motionless. As a result, using the technique to compute the alpha of a moving foreground object is highly non-trivial. In theory, one way to do this would be to somehow capture images of the foreground object against two different backgrounds simultaneously. Suggest techniques that might let you do this, and justify your answer. If you wish, you can do a literature search to help you solve this problem. If you do find any related work, be sure to cite it in your report and explain how it relates to your answer.

Your answer should be included in your PDF report: report.pdf

---

What to turn in: Please submit the following files on MarkUs:

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

along with example output images from your report.

Bonus policy: You are strongly advised to finish all the required components of an assignment before tackling the bonus questions: submissions with even one required part substantially incomplete (or incorrect) will not be awarded any bonus marks.

Working on non-CDF machines: You are welcome to work on the assignment on a non-CDF machine. That being said, your implementation must run on the Linux CDF machines in order to be marked by the TAs. You should therefore test your code on CDF frequently, to make sure it has no CDF-specific issues.

All required packages have already been installed on CDF/Linux. Here are the steps I followed to install them on OS X 10.10+ :

1. Install NumPy and related packages for scientific computations:
   `http://stronginference.com/ScipySuperpack/`
2. Install OpenCV 3.1:
   `http://blogs.wcode.org/2014/10/howto-install-build-and-use-opencv-macosx-10-10/`

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):
   `http://docs.opencv.org/3.1.0/d6/d00/tutorial_py_root.html#gsc.tab=0`
4. Matplotlib User Guide (especially Chapter 5.5):
   `http://matplotlib.org/1.5.1/Matplotlib.pdf`
5. NumPy Reference Guide (especially Chapters 1.4, 1.5, 3.17.1-3.17.5):
   `http://docs.scipy.org/doc/numpy/numpy-ref-1.10.1.pdf`