CpSc 2120: Algorithms and Data Structures

Instructor: Dr. Brian Dean
Webpage: `http://www.cs.clemson.edu/~bcdean/`
Handout 17: Homework #4

Fall 2020
MWF 9:05-9:55
Flour 132

1 Fun with Shortest Paths

In this assignment we'll showcase some interesting ways to use shortest paths to solve three different problems. Some starter code and data is available here:

`/group/course/cpsc212/f20/hw04/`

2 Problem One: It's All a Blur...

The first problem we consider is taking a black-and-white image and executing a "blur" effect. Specifically, we want to make the Clemson tiger paw image `paw.ppm` below on the left transform into the image below on the right.

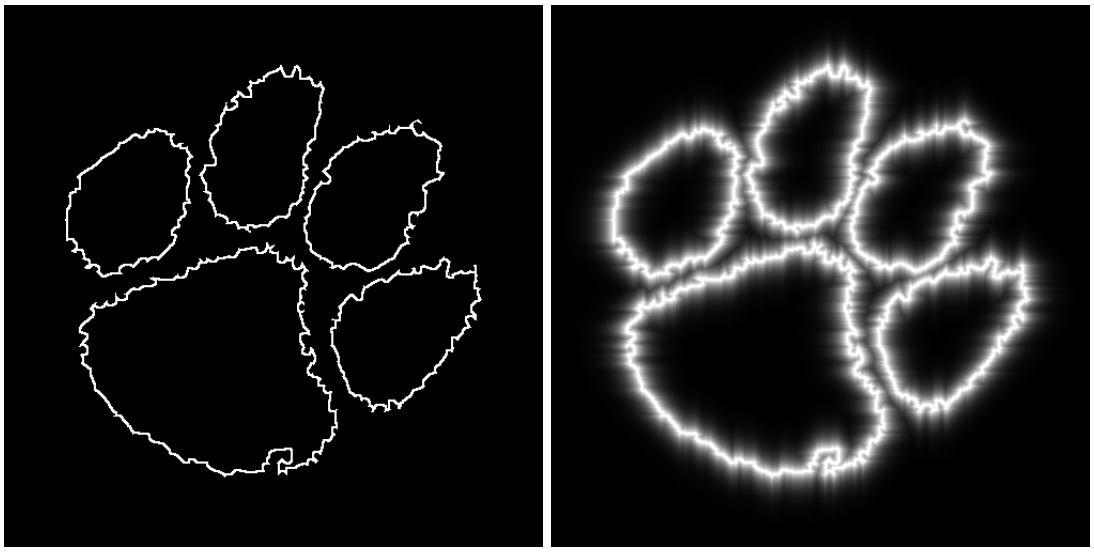

Figure 1: Tiger paw outline and blurred version thereof

The file `blur.cpp` contains starter code for this part, which is based on what we wrote during our USA map demo lecture. It reads in `paw.ppm` and calls a function `calculate_blur` (which you are to write), then writes the resulting modified image out as `paw2.ppm`. To display a ppm image on

the screen while logged into the `virtual.computing` terminal, you can use the command `display paw.ppm`. If needed, you can also convert between image formats using a command like `convert paw2.ppm paw2.png`.

The `calculate_blur` function you write should modify the paw image to add the blur effect. Recalling that the color white consists of red, green, and blue levels all set to 255, your function should set the red, green, and blue levels of each pixel p to the value 255×0.9^d, where d is the minimum number of steps required to get from p to the nearest white pixel in the image. A "step", here, is defined as moving north, south, east, or west to an adjacent pixel. Observe that the 255×0.9^d formula leaves white pixels white (since $d = 0$), and it otherwise makes pixel intensity decay exponentially with distance from a white pixel.

When you are done with this part, please submit your `blur.cpp` file. Your code should run in linear or nearly-linear time in the size of the image for full credit.

3 Problem Two: Seams Easy Enough...

To ensure he has adequate funds for tuition and pancakes all semester, John Perez decides to start a side business, "John's taxi service", charging students to transport them across campus. Unfortunately, he didn't pick the best business name, since John Pascoe already runs "John's shuttle service", and Jonathan Daniel" already runs "Jonathan's transportation, Inc". To help advertise his new business, John Perez decides to rent a billboard next to campus, displaying an 800×400 image he proudly created using his favorite drawing tool, Microsoft Paint:

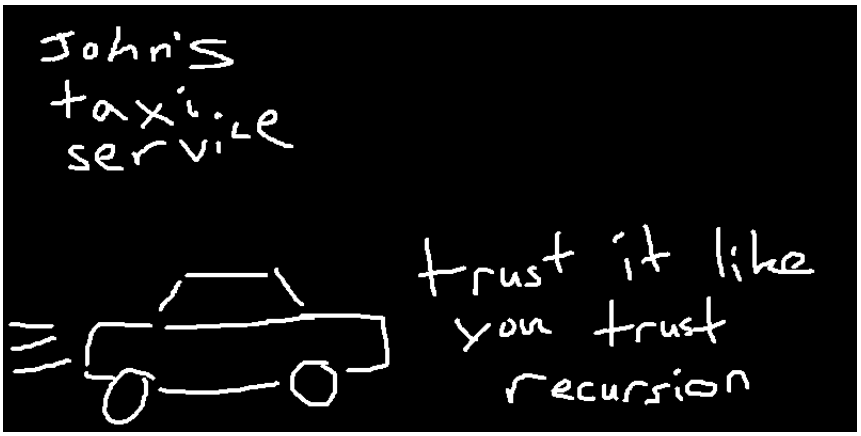

Figure 2: Advertising John Perez's new business

Unfortunately, John realizes that the billboard he has rented can only accommodate a *square* image, so he needs to compress his rectangular image to only 400×400 pixels in size. He wants to do this in a way that preserves a much of the beautiful text and graphics as possible.

Being a fan of fancy algorithms, John wants to try using a technique called *seam carving* to shrink the image. The technique is explained well in a Youtube video you can find if you search for "seam carving Avidan Shamir" (Avidan and Shamir are the authors of the paper presented in the video). A shortest path calculation is used to find a 1-pixel-wide "seam" (a path from the top of the image down to the bottom of the image that ideally cuts through the most "boring" parts of

the image) that is then removed to shrink the image horizontally by one pixel. Since the seam avoids interesting regions of the image, this method effectively shrinks an image while preserving its key points of detail. The seam starts at a pixel in the top row, ends at a pixel in the bottom row, and in every step, it moves south, southeast, or southwest by 1 single pixel. Therefore, the seam involves precisely one pixel per row of the image:

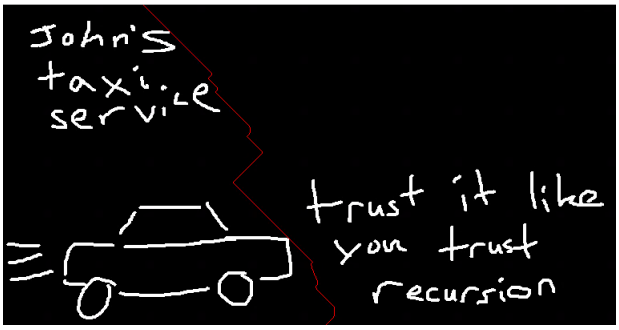

Figure 3: Billboard image with a seam, shown in red.

After finding and removing 400 seams, the billboard image (now square) looks like this:

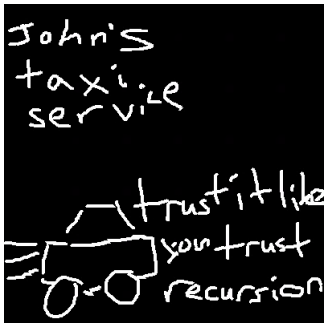

Figure 4: Billboard image with 400 seams removed

Observe how the text at the top remains largely unaffected, while the bottom half compresses together so as to preserve as much detail as possible.

Starter code for this part is provided in `main.cpp`, `graphics.cpp`, and `graphics.h`, and compiles on the `virtual.computing` platform with the supplied Makefile. Be sure to use the OpenGL-enabled login for `virtual.computing`, since the code here actually gives you an interactive graphical interface for testing your algorithms. You'll be modifying just the `main.cpp` file, which currently contains some of the same image manipulation code from the previous part, as well as some code to support rendering, user interaction, and seam removal. When you run the code, it will pull up the billboard image and respond to commands from the keyboard. Pressing 'q' quits.

There are two steps you'll need to complete for this section. First, you'll want to import your solution from the preceding part and fill in the `calculate_blur` function. The 'b' key toggles display of the blurred / non-blurred image, so you can check that your code is working easily.

The main task you need to complete here is filling in the `calculate_seam` function, so it uses a

shortest path calculation to find and draw a seam consisting of red pixels on the current image. Letting the *intensity* of a pixel be its red color value (which is the same as its green and blue color values, as our image is greyscale), you want to find a seam that minimizes the sum of pixel intensities along the path of the seam through the blurred image (so it tends to stay as far away from bright white regions of high detail as possible).

To test your algorithm, press 's' in the visualization code, and this will call your `calculate_seam` function, displaying the seam it computes as a red path across the image; a second later, it will shrink the image using this seam. Pressing 's' repeatedly will continue to shrink the image. You can also press '2' through '9' to remove several seams in a batch fashion.

When you are confident that your seam calculation code is working well, please submit `main.cpp`. Your code for finding a single seam should run in nearly linear time in the image size (possibly times logarithmic factors) for full credit. As you can guess, you are *not* to go looking for pre-existing seam-carving code on the internet to solve this part.

4 This Part's Even Cooler...

Word has gotten out that the transportation business is lucrative at the moment, so as a result every single student in CpSc 2120 has now started a taxi company!

Interestingly, there are only 16 landmarks $L_1 \ldots L_{16}$ across campus that serve as pick-up / drop-off locations for these services — these are listed in the file `places.txt`. Each transportation service follows a different route for its pick-ups and drop-offs, as detailed in the file `transit.txt`. Each line in `transit.txt` describes a single transportation service, starting with the name of the student running the service and the cost the service charges for a single leg of transit (i.e., picking up at some location, and dropping off at the next location on the route for that service), finally each line describes the route for a service, in terms of 15 drop-off locations $D_1 \ldots D_{15}$. If a student is picked up from landmark L_i, that student is then dropped off at destination D_i. For example, John Perez charges \$6.42 per leg of transit; if he picks up a student from Schilleter dining hall (L_1), that student is dropped at Death Valley (D_1), and if he picks up a student from Cooper library (L_2), that student is dropped off at Littlejohn Collosium (D_2).

The Clemson exchange (L_{16}) is the only location from which no service picks up. This makes sense, of course, because once you've reached the Clemson exchange, why would you want to go anywhere else? The ice cream there is quite delicious.

Your goal is to determine a minimum-cost sequence of transportation services to take such that you will be guaranteed to reach the Clemson exchange *no matter where you start*. As your solution, print the list of student names whose services are used, along with the total cost of them all. For example, if your answer is "Nick_Rabon Edward_Ng Kellen_Hass", then this means wherever you start, you take one leg of Nick Rabon's transportation service, then one leg of Edward's service, and then one leg of Kellen's service. This can of course cause you to end up at different places depending on where you start, but you want to choose a sequence of services that guarantees reaching the Clemson exchange. If you reach the Exchange early, before using all the legs in your path, that's fine (since once you reach the Exchange, you stay there).

No starter code is provided for this part. The main hint here is to think about what should be the graph in which you are computing shortest paths here. Your "state" isn't a single location, but rather it's a subset of locations at which you could possible reside. Initially, your state is the set of

all locations (since you could start anywhere), and you want to finally reach a state where you are 100% certain of being located at the Exchange.

5 Submission and Grading

Make sure your code compiles on the lab machines before submitting. Final submissions are due by 11:59pm on the evening of Friday, December 4. No late submissions will be accepted.