Programming Assignment 2: Seam Carving

Frequently Asked Questions

We haven't offered this assignment before, so we don't know what kinds of questions students will have. We will add to the list as needed.

How do I manipulate images in Java? Use our <u>Picture</u> data type (which is part of stdlib.jar) and the <u>Color</u> data type (which is part of the java.awt library). Here is some more information about the <u>Color</u> and <u>Picture</u> data types. <u>Luminance.java</u> and <u>Grayscale.java</u> are example clients that use the Color and Picture data types, respectively.

I noticed that the Picture API has a method to change the origin (0, 0) from the upper teft to the lower left. Can I assume (0, 0) is the upper left pixel? Yes.

Must the arguments to removeHorizontalSeam() and removeVerticalSeam() be minimum energy seams? No. These methods should work for any valid seam (and throw an exception for any invalid seam).

Testing

Clients. You may use the following client programs to test and debug your code.

- <u>PrintEnergy.java</u> computes and prints a table of the energy of an image with filename provided on the command line.
- <u>ShowEnergy.java</u> computes and draws the energy of an image with filename provided on the command line.
- <u>ShowSeams.java</u> computes the horizontal seam, vertical seam, and energy of the image with filename provided on the command line. Draws the horizontal and vertical seams over the energy.
- <u>PrintSeams.java</u> computes the horizontal seam, vertical seam, and energy of the image with filename provided on the command line. Prints the horizontal and vertical seams as annotations to the energy. Many of the small input files provided also have a printseams.txt file (such as <u>5x6.printseams.txt</u>), so you can compare your results to the correct solution.
- ResizeDemo.java uses your seam removal methods to resize the image. The command line arguments are filename, W and H where W is the number of columns and H is the number rows to remove from the image specified.
- <u>SCUtility.java</u> is a utility program used by most of the above clients.

Sample input files. The directory <u>seam</u> contains the client programs above along with some sample image files. For convenience, <u>seam-test.zip</u> contains all of these files bundled together. You can also use your own image files for testing and entertainment.

Possible Progress Steps

These are purely suggestions for how you might make progress. You do not have to follow these steps.

- 1. Start by writing the constructor as well as picture(), width() and height(). These should be very easy.
- 2. From there, write energy(). Calculating Δ_x^2 and Δ_y^2 are very similar. Using two private methods will keep your code simple. To test that your code works, use the client PrintEnergy described in the testing section above.
- 3. To write findVerticalSeam(), you will want to first make sure you understand the topologial sort algorithm for computing a shortest path in a DAG. Do *not* create an EdgeWeightedDigraph. Instead

construct a 2d energy array using the energy() method that you have already written. Your algorithm can traverse this matrix treating some select entries as reachable from (x, y) to calculate where the seam is located. To test that your code works, use the client PrintSeams described in the testing section above.

- 4. To write findHorizontalSeam(), transpose the image, call findVerticalSeam(), and transpose it back.
- 5. Now implement removeVerticalSeam(). Typically, this method will be called with the output of findVerticalSeam(), but be sure that they work for any seam. To test that your code words, use the client ResizeDemo described in the testing section above.
- 6. To write removeHorizontalSeam(), transpose the image, call removeVerticalSeam(), and transpose it back.

Optimizations

- 1. There is no need to create a new Picture object after removing a seam —instead, you can maintain the color information in a 2D array of integers. That is, you can defer creating a Picture object until required to do (when the client calls picture()). Since Picture objects are relatively expensive, this will speed things up.
- 2. Reuse the energy array and shift array elements to plug the holes left from the seam that was just removed. You will need to recalculate the energies for the pixels along the seam that was just removed, but no other energies will change.
- 3. Don't explicitly transpose the Picture until you need to do so. For example, if you perform a sequence of 50 consecutive horizontal seam removals, you should need only two transposes (not 100).
- 4. Is it faster to traverse the energy matrix in row-major order or column-major order? *Hint*: Recall that in Java a "2D array" is really an array of arrays.
- 5. Consider using System.arraycopy() to shift elements within an array.

Challenge for the bored

- 1. Your energy() method implemented the dual gradient energy function. Try out other energy functions.
- 2. Implement an interactive object-removal feature: The user highlights an area of the image, and that portion of the image is forced to zero energy. Rows and columns are then successively removed until every pixel in that zero-energy region has been removed.