CSC 440 Assignment 5: Dynamic Programming - Seam Carving

Out: Tuesday, Mar 27th

Due: Friday, Apr 6th, by 11PM

Introduction

In a 2007 paper, Avidan and Shamir describe a new method for resizing images with minimal visual disruption. The breakthrough is an algorithm for removing the lowest *energy* vertical or horizontal seam in the image.

Both the demonstration video and original paper will be posted to Sakai. Please watch the video so that the terminology used here is clear.

You are given an image, and your task is to calculate the best vertical seam to remove. A vertical seam is a connected path of pixels, with one pixel per row of the image. Two pixels are *connected* if they are vertically or diagonally adjacent. The best vertical seam is the one that minimizes the total *energy* of pixels in the seam.

Remember, it is a convention in most computer graphics systems that $(0, 0)$ is the upper-left pixel.

Here is the algorithm from the paper:

Subproblems

For each pixel (i, j), what is the lower-energy seam that starts at the top row of the image, but ends at (i, j)? image, but ends at (i, j)?

Relation

Let `dp[i,j]` be the solution to subproblem (i, j). Then `dp[i,j] = min(dp[i,j-1], dp[i-1,j-1], dp[i+1,j-1]) + energy(i,j)`

Assignment components

Implementation

Download a5.zip from Sakai and unzip it (please do not use your decompressor from assignment 4!)

You will need to install the Python Pillow library. You should be able to install it with:

```
pip install Pillow
```

or

```
easy_install Pillow
```

or download https://pypi.python.org/pypi/Pillow and from inside the archive run:

```
python setup.py install
```

In `resizable_image.py`, implement a function `best_seam(self, dp=True)` that returns a list of coordinates corresponding to the lowest-energy vertical seam to remove, e.g. `[(5, 0), (5, 1), (4, 2), (5, 3), (6, 4)]`.

The class ResizeableImage inherits from ImageMatrix. You should use the following components of ImageMatrix in your program:

- `self.energy(i,j)` returns the energy of a pixel. This takes $O(1)$ time, but the constant factor is large. If you call `energy` more than once on the same pixel, you should cache the result. This memoization is *separate* from the dynamic programming memoization (there are no subproblems involved) and so **you should still cache the energy for a pixel even if** `dp == False`.
- `self.width` and `self.height` are the width and height of the image.

Your implementation may be either bottom-up or top-down. But either way, it must respect the argument `dp`, which indicates whether or not dynamic programming should be used. If `dp == True`, then you should either use memoization or store the subproblem values in a table for re-use. If `dp == False`, then you should naively recompute those subproblems.

- test your code using `test_resizable_image.py`, and submit `resizable_image.py` via Gradescope.
- you can also view your code working by running `gui.py`
- you should try out your code on other images of your choosing, once it's working

Since the default behavior of `best_seam()` is to perform dynamic programming, the test suite provided will only test that variant.

However, for small inputs, turning off dynamic programming should be possible.

You **do not** need to document any invariants for this assignment. However, you will still be graded on the design and representation of your code.

Benchmarking

On `sunset_small.png` and several other *small* image files, and potentially on `sunset_full.png` (if the running time is not excessive), time your code (just like with Assignment 1) both **with** and **without** dynamic programming. You should test on enough inputs to be able to make convincing plots of running time. Clearly, there will be inputs on which only the dynamic programming solution is tractable, so I expect your naive plot will cover fewer data points. Think about what it means to vary the image size.

Plot the running times. You should upload to gradescope a PDF including your plots and the analysis below.

Analysis

Come up with an analysis of the asymptotic complexity of the algorithm, both *with* and *without* dynamic programming. Write out the recurrence relations. Pay attention to **how many** subproblems there are, and how much smaller they are. Solve the recurrence relations (you do not need to show your work for this) and provide the solutions in the comments. Finally, answer the question: do these solutions make sense given the results of the benchmarking?

For this assignment, your solution must be in Python.

Your solution should be simple enough that it works in both Python 2.7 and 3.x We will evaluate your solutions in Python 2.7.

Pair Programming

You will work with a partner for this entire assignment. Please refer to the pair programming policy in the syllabus.

Lateness

Submissions will not be accepted after the due date, except with the intervention of the Dean's Office in the case of serious medical, family, or other personal crises.

Grading Rubric

Your grade will be based on four components:

- Functional Correctness (25%)
- Benchmarking comparison (25%)
- Design and Representation (25%)
- Analysis of the algorithm (25%)

For this assignment, you do not need to specifically state any invariants. However, we still expect your code to be well-documented and well-structured.