Computer Graphics – Spring 2016

Exercise 1 – Seam Carving

Submission: Monday, June 6th 2016 at 23:55

Seam carving is a novel algorithm for resizing images while maintaining as much information as possible from the source image. A "seam" in this context is an 8-connected path of pixels from the top of the image to the bottom or from the left to the right.

Seam carving uses a dynamic programming method to compute a directed energy map over the image. Using this map it finds a seam with the least energy. Removing this seam produces a smaller image and applying this process repeatedly allows reducing the size of the image freely and changing its aspect ratio.
In order to increase the size of the image, seam carving finds *k* seams with the least energy and duplicates them in ascending order.

Part 1

Please read the Seam Carving paper here:
http://www.faculty.idc.ac.il/arik/papers/imret.pdf

In essence, your seam carving implementation should follow these general steps:

1. Compute the energy function over the image (see below). This should produce a numeric value for every pixel.
2. Decide on a seam direction – vertical or horizontal. To simplify, you can transpose the image when a horizontal seam is chosen (the rest of the steps assume a vertical seam).
3. Compute the dynamic programming map from the second row to the bottom.
4. Find the lowest energy seam by selecting the path with the least energy:
    - Start from the bottom most pixel with the lowest value
    - Repeat by proceeding upwards to its lowest energy neighbor
    - Take notice to choose only one pixel per column/row
5. Remove the seam, adjusting each row to its new size which is one pixel short of the old size.

Note that these are the steps for reducing the size of the image by one row or column.
For increasing the image size, refer to the article for a slightly different procedure.

Energy Function

1. Implement the following energy function: for every pixel calculate the Red, Green and Blue value differences from the pixel to its 8 neighbors and sum up their values. This is essentially the gradient of the pixel.
2. Add local entropy to the energy function above. The entropy for pixel *i* is computed over a 9x9 window centered at *i* as follows:
$$H_i = -\sum_{m=i-4}^{i+4}\sum_{n=i-4}^{i+4} p_{mn} log(p_{mn})$$
where:
$$p_{mn} = f(m,n) / \sum_{k=i-4}^{i+4}\sum_{l=i-4}^{i+4} f(k,l)$$
and:
$$f(m,n) = greyscale\ value\ at\ pixel\ (m,n)$$
Note that your energy function should be a weighted combination of the two terms.
3. Run your program both with and without the local entropy and qualitatively evaluate the difference.

Part 2

Please read Section 5 (up to 5.2, not including) in the paper Improved Seam Carving for Video Retargeting:
http://www.faculty.idc.ac.il/arik/SCWeb/vidret/vidretLowRes.pdf

Now incorporate forward energy into your solution from part 1.
Find images that emphasize the contribution given by forward energy and run both parts on them to compare.

Implementation

1. Your solution should be implemented Java.
2. Use the package ImageIO.
3. Your program should receive the following command line arguments:
<input image filename> <output # columns> <output # rows> <energy type> <output image filename>
Where:
<input image filename> = Full path to the input image
<output # columns> = Number of columns of the resized output image
<output # rows> = Number of rows of the resized output image
<energy type> = A boolean argument where '0' = regular energy without entropy term, '1' = regular energy with entropy term and '2' = forward energy

<output image filename> = Full path to the output image (where your program will write the output image to)
Note that you need to be able to deal with all combinations of increase/decrease in image size. For example, you may be given an input where the number of columns decreases but the number of rows increases.

4. Energy map computation:
   a. Assume we are now computing the energy value for pixel *i* (R*i*,G*i*,B*i*).
   Pixel *i* has 8 neighbors numbered 1-8. Then, for neighbor no. 1 (R*1*,G*1*,B*1*) we have:
   val1 = (abs(R*i* - R*1*) + abs(G*i* - G*1*) + abs(B*i* - B*1*)) / 3
   And so the complete energy value for pixel *i* would be:
   energy_*i* = val1 + val2 + ... + val8
   In boundary cases (where the pixel is on the image boundary) only take into account actual neighbors. Therefore if a pixel has only five neighbors then its energy sum will be composed of five terms.
   b. In order to avoid image cropping (from boundary pixels having unfair advantage with their low energy), please normalize the energy value per pixel by its number of neighbors. For example, if pixel *i* has 8 neighbors and its energy is energy_*i*, then its normalized energy will be: energy_*i* / 8
5. Seam direction selection:
   Section 4.2 in the Seam Carving paper suggests a method to find the optimal seam removal order. Implementing this method is a bonus. You may choose one of the simple removal orders if you prefer to skip the bonus.
6. Submit all you code and an executable .jar file titled <id1>_<id2>.jar inside a zip file titled <id1>_<id2>.zip

General instructions

1. Submission is in pairs.
2. Submit your zip files through the Moodle site of the course.
3. Questions can be posted on the designated forum in Moodle.