CIS581, Computer Vision
Project 3, Image Carving and Automatic 2D Image Mosaic
Due November 10, 3:00pm

Overview

This project is to be done individually.

This project will have two focuses: Image resizing utilizing seam carving, and image/mosaic stitching.

Part 1 will focus on the concepts of image resizing utilizing the principals supporting minimum-energy seam carving, and dynamic programming. The goal of this part of the project is to help you understand an important approach towards resizing images while attempting to preserve the integrity of important image information. The methodology closely follows that which is outlined in:

"*Seam Carving for Content-Aware Image Resizing*"; Avidan, S. & Shamir, A.; 2007

Part 2, on mosaicing, focuses on image feature detection, feature matching and image mosaic techniques. The goal of this project is to create an image mosaic or stitching, which is a collection of small images which are aligned properly to create one larger image. We will follow technique outlined in the following papers, which are available on the course website:

"*Multi-image Matching using Multi-scale image patches*", Brown, M.; Szeliski, R.; Winder, S. CVPR 2005

"*Shape Matching and Object Recognition Using Shape Contexts*", Belongie, S., Malik, J. and Puzicha, J. PAMI 2002: http://www.eecs.berkeley.edu/Research/Projects/CS/vision/shape/

Submission

Compress your files into a ZIP file named "3_<penn_user_name>.zip", which should contain 2 folders, one each for each part of this project, containing:

- For Part 1, your .m files for five (5) Matlab functions you need to complete: (`carv.m`, `cumMinEngVer.m` and `cumMinEngHor.m`, `rmVerSeam.m` and `rmHorSeam.m`)
- For Part 2, your .m files for the five (5) required Matlab functions (`anms.m`, `feat_desc.m`, `feat_match.m`, `ransac_est_homography.m`, `mymosaic.m`)
- any demo .m script(s) for reproducing/showing your results
- any additional .m files with helper functions for your code - this includes any third party code that you used, eg for harris corner detection
- the input images you used
- the resulting stitched resized image(s) and image mosaic(s) respectively.
- a PDF showing your results and describing any additional features of your implementation.
  - For Part 1, show the initial and final image(s), and a copy of the original image with seams colored in (like in the referenced paper, or notes covered in class).
  - Results for Part 2 are constituted by 1) the final stitched mosaic (which should already be included as a standalone image) and also some intermediary results showing 2) corner detection results of an image in red dots, 3) adaptive non-max suppression of the image (used to show corner detection results) in red dots, 4) final matching results (right before stiching) of all pairs of images used in mosaic in blue dots with outliers in red dots. If you used third party code, make it clear. Make the PDF as simple as possible.

Part 1: Image Carving and Resizing

For this part of the project, you will be implementing image resizing utilizing scene carving. The structure of this part of the project is based heavily on the methods outlined by Avidan & Shamir in their paper: "*Seam Carving for Content-Aware Image Resizing*", which you are strongly encouraged to read prior to starting this part of the project (a link to the paper will be made available on the course wiki, though finding it via a search engine should be very easy too).

You are tasked with filling in 5 functions: `carv.m`, `cumMinEngVer.m` and `cumMinEngHor.m`, `rmVerSeam.m` and `rmHorSeam.m`, wrapped by `carv_wrap.m`

`carv_wrap.m` has the following structure:
`[Ic, T, Mx, My, Ix, Iy, Ex, Ey] = carv_wrap(I, nr, nc, flag)`
and serves as a wrapper for resizing an image I, from size $m \times n$ to size *(m-nr)* $\times$ *(n-nc)*. For the purposes of this project, we will be testing you strictly on shrinking an image (making it smaller), though the ideas applied towards enlarging an image are very similar to those of shrinking one. The `flag` input is mainly only used while debugging or grading to enable more outputs.

The key outputs of concern are: I_c - the resized image, T - the transport map, M_x and M_y - the cumulative minimum energy along the vertical and horizontal respectively. I_x and I_y - Images with a single minimum energy seam (vertical and horizontal respectively) removed, and E_x and E_y - the cost of removing said seams, are outputs only used when we test your code during grading.

Within `carv_wrap.m`, we begin by computing the energy map, $e(I) = |\frac{\delta}{\delta x}I| + |\frac{\delta}{\delta y}I|$. We provide a function, `genEdgeMap.m` which computes this for you, in order to maintain consistency in the method by which the energy map is computed. Your tasks are to fill in the code that follows.

Task 1: COMPUTING CUMULATIVE MINIMUM ENERGY

You need to complete the codes for the functions `cumMinEngVer.m` and `cumMinEngHor.m`, which correspond to computing the cumulative minimum energy over the vertical and horizontal seam directions respectively.

You'll notice that Avidan & Shamir as well as our class notes define seams for an $n \times m$ image, I, as follows:

- A vertical seam $\mathbf{s^x} = \{s_i^x\}_{i=1}^n = \{(x(i), i)\}_{i=1}^n$, s.t $\forall i, |x(i) - x(i-1)| \leq 1$, where x is a mapping $x : [1, \ldots, n] \rightarrow [1, \ldots, m]$, i.e. the corresponding column for a seam pixel at row i along an 8-connected path from the top to the bottom of the image.

  Note that for any vertical seam, there only exists one seam pixel per row.

- Similarly, a horizontal seam is defined as $\mathbf{s^y} = \{s_j^y\}_{j=1}^m = \{(y(j), j)\}_{j=1}^m$, s.t $\forall j, |y(j) - y(j-1)| \leq 1$, where y is a mapping $y : [1, \ldots, m] \rightarrow [1, \ldots, n]$. As with the vertical seam, there exists one seam pixel per column along an 8-connected seam path.

For a vertical seam, the seam cost is defined as $E(\mathbf{s^x}) = E(\mathbf{I_{s^x}}) = \sum_{i=1}^n e(\mathbf{I}(s_i))$, and for horizontal seam, it is $E(\mathbf{s^y}) = E(\mathbf{I_{s^y}}) = \sum_{j=1}^m e(\mathbf{I}(s_j))$. We seek to identify the optimal

seam, $\mathbf{s}^*$, that minimizes the seam cost, i.e. $\mathbf{s}^* = \min_s E(\mathbf{s})$

To quote Avidan & Shamir:
The optimal seam can be found using dynamic programming. The first step is to traverse the image from the second row to the last row and compute the cumulative minimum energy **M** for all possible connected seams for each entry (i, j):

$$M_x(i,j) = e(i,j) + min(M_x(i-1,j-1), M_x(i-1,j), M_x(i-1,j+1))$$

$$M_y(i,j) = e(i,j) + min(M_y(i-1,j-1), M_x(i,j-1), M_x(i+1,j-1))$$

At the end of this process, the minimum value of the last row in $\mathbf{M_x}$ will indicate the end of the minimal connected vertical seam. Note that you have to record a backtrack table along the way. Hence, in the second step we backtrack from this minimum entry on $\mathbf{M_x}$ to find the path of the optimal seam. Also, in order to maintain consistency, whenever there are more than one minimum values, choose the one with smaller index. The definition of $\mathbf{M_y}$ for horizontal seams is similar.

Task 2: REMOVING A SEAM

For this task, you will fill in the two functions, `rmVerSeam.m` and `rmHorSeam.m`, which remove vertical and horizontal seams respectively. This task should be fairly simple. You should identify the pixel from $\mathbf{M_x}$ or $\mathbf{M_y}$ from which you should begin backtracking in order to identify pixels for removal (**rmIdx**), and remove those pixels from the input image. You will receive two inputs to each function, the corresponding cumulative minimum energy map, and the image. Utilizing these, you should output an image with one (appropriate) seam removed.

Task 3: DISCRETE IMAGE RESIZING (Optional)

Now that you're able to handle finding seams of minimum energy, and seam removal, we shall now tackle resizing images when it may be required to remove more than one seam, sequentially and potentially along different directions. When resizing an image from size $n \times m$ to $n' \times m'$ (we will assume $n' < n$ and $m' < m$), the sequence of removing vertical and/or horizontal seams is important.

For this task, fill in the function `carv.m`, making use of the method described below. Utilizing recursive calls of the functions completed in the previous 2 tasks, complete this function to output the resized image.

The following method is derived directly from the works of Avidan & Shamir:

We define the search for the optimal order as an optimization of the following objective function:

$$\min_{\mathbf{s^x},\mathbf{s^y},\alpha} \sum_{i=1}^{k} E(\alpha_i \mathbf{s_i^x} + (1-\alpha_i)\mathbf{s_i^y})$$

where $k = r + c, r = (m - m'), c = (n - n')$ and α_i is used as a parameter that determine if at step i we remove a horizontal or vertical seam. $\alpha_i \in \{0, 1\}, \sum_{i=1}^{k} \alpha_i = r, \sum_{i=1}^{k}(1-\alpha_i) = c$

We find the optimal order using a transport map $\mathbf{T}$ that specifies, for each desired target image size $n \times m$, the cost of the optimal sequence of horizontal and vertical seam removal operations. That is, entry $T(r, c)$ holds the minimal cost needed to obtain an image of size $n - r \times m - c$. We compute $\mathbf{T}$ using dynamic programming. Starting at $T(0, 0) = 0$ we fill each entry (r, c) choosing the best of two options - either removing a horizontal seam from an image of size $n - r \times m - c + 1$ or removing a vertical seam from an image of size $n - r + 1 \times m - c$:

$$T(r,c) = min(T(r-1,c) + E(\mathbf{s^x}(\mathbf{I_{n-r-1\times m-c}})), T(r,c-1) + E(\mathbf{s^y}(\mathbf{I_{n-r\times m-c-1}})))$$

where $\mathbf{I_{n-r\times m-c}}$ denotes an image of size $n - r \times m - c$, $E(\mathbf{s^x}(\mathbf{I}))$ and $E(\mathbf{s^y}(\mathbf{I}))$ are the cost of the respective seam removal operation. We can store a simple binary map which indicates which of the two options was chosen in each step of the dynamic programming. Choosing a left neighbor corresponds to a vertical seam removal while choosing the top neighbor corresponds to a horizontal seam removal. Given a target size $n' \times m'$ where $n' = n - r$ and $m' = m - c$, we backtrack from $T(r, c)$ to $T(0, 0)$ and apply the corresponding removal operations. Note that in Matlab, indexes begin by 1 instead of 0, so in practice you will backtrack from $T(r + 1, c + 1)$ to $T(1, 1)$. Also, for the sake of consistency, $T(r, c) = T(r - 1, c) + E(\mathbf{s^x}(\mathbf{I_{n-r-1\times m-c}}))$, if $T(r - 1, c) + E(\mathbf{s^x}(\mathbf{I_{n-r-1\times m-c}})) = T(r, c - 1) + E(\mathbf{s^y}(\mathbf{I_{n-r\times m-c-1}}))$.

Further notes:

In addition, in order to calculate $E(\mathbf{s^x}(\mathbf{I}))$ and $E(\mathbf{s^y}(\mathbf{I}))$, you have to record a table for $\mathbf{I_{n-r\times m-c}}$. More specifically, $\mathbf{TI}$ is the trace table and should be of size $nr + 1 \times nc + 1$. $\mathbf{TI\{1, 1\}}$ is the source image, and $\mathbf{TI\{r + 1, c + 1\}}$ is the image $\mathbf{I_{n-r\times m-c}}$, which is the source image removed r rows and c columns.

Also, due to MATLAB's use of 1-indexing, note that all indices referenced with 0-indexing in the method discussed above will be +1 in both rows and columns (e.g. $T(0, 0) \rightarrow T_{matlab}(1, 1)$).

Due to this being the first time that we are including image resizing utilizing carving and retargetting, we don't have an FAQ section for this part of the project. If you have questions, you are encouraged to communicate your queries during office hours, or on Piazza

Part 2: Image Mosaicing

Function Specifications

This part of the project requires you to write a number of functions. The specifications for each have been included on the Project page of the course website under the "Project-3 Part 2 Function Specifications" link. Please make sure your code follows those exactly.

Task 1: CAPTURE IMAGES

For this project, you need to capture multiple images of a scene, which you will use to create image mosaic. In general, you should limit your camera motion to purely translational, or purely rotational (around the camera center).

Bonus points will be given for interesting selection of images.

Task 2: AUTOMATIC CORRESPONDENCES

You need to implement the following steps.

1. Detecting corner features in an image. Recommand to use Matlab `cornermetric` with proper arguments. But, you could follow method outlined in lecture note on Interest Point Detector to build your own image corner detector (for example derived from edge detector). You can probably find free "harris" corner detector on-line, and you are allowed to use them. Software links are provided on the course website.
2. (`anms.m`) Implement Adaptive Non-Maximal Suppression. The goal is to create an uniformly distributed points. See section 3 of the reference paper, as well as lecture notes.

3. (`feat_desc.m`) Extracting a Feature Descriptor for each feature point. You should use the subsampled image around each point feature. Dont worry about rotation-invariance just extract axis-aligned 8x8 patches. Note that it's extremely important to sample these patches from the larger 40x40 window to have a nice big blurred descriptor. Don't forget to bias/gain-normalize the descriptors.

   As a bonus, you can implement the Geometric Blur features.

4. (`feat_match.m`) Matching these feature descriptors between two images. Remember to use the trick to compare the best and second-best match (take their ratio).

TASK 3: RANSAC and IMAGE MOSAIC

Not all the matches computed in Task 2 will be correct. One way to remove incorrect matches is by implement the additional step of RANSAC (see lecture notes).

1. (`ransac_est_homography.m`) Use a robust method (RANSAC) to compute a homography. Use 4-point RANSAC as described in class to compute a robust homography estimate.

   Recall the RANSAC steps are:

   (a) Select four feature pairs (at random), p_i, p_i^1

   (b) Compute homography H (exact). Code is available for this on the Project page.

   (c) Compute inliers where $SSD(p_i^1, Hp_i) < thresh$

   (d) repeat step a-c

   (e) Keep largest set of inliers

   (f) Re-compute least-squares H estimate on all of the inliers

2. (`mymosaic.m`) Produce a mosaic by overlaying the pairwise aligned images to create the final mosaic image. Also, check check Matlab functions, e.g `imtransform` and `imwarp`. If you want to implement `imwarp` (or similar function) by yourself, you should apply bilinear interpolation when you copy pixel values. As a bonus, you can implement smooth image blending of the final mosaic.

FAQ

Q: Suppose I have the matrix cimg which contains corner strengths for each pixel. Do I need to set a threshold to get corners candidates? And then I only need to calculate radius for each corner candidate. Or I need to calculate radius for all pixels?

A: Ultimately you want corners that are at distinctive locations and well distributed throughout the image. If you don?t threshold at all then in areas that are relatively con- stant (say a plane white wall) you?ll get some corners due to noise that won?t end up being matches anyway since the location was essentially random. If you threshold too much ini- tially, however, you might only get corners in a few strong regions and cut out fainter but justified corners. Since for the panorama the overlap between images (and hence corners that can correctly match between two images) is a small region near one of the boundaries, you want to be sure that some corners respond in all portions of the image. The anms will then prune the corners that survived the thresholding to a manageable set that is also nicely distributed.

Q: I was having a problem getting the feat desc to account for the window size at the edges. I tried scaling the size of the window based on the location of the corner, but this would obviously not match with the $P_{(64\times n)}$ matrix output.

A: You could try zero-padding those corners. Doing so makes it more likely that the corner won?t find a match or will find a close ssd score to other corners that got zero padded. You could also remove corners within 20 pixels of the edge from consideration so that all of your features can fit a window around them. Your region of overlap will probably be large enough to get plenty of matches past the 20 pixels on the boundary. Additionally assuming pixels near the boundary in one image aren?t also near the boundary in the other images (some instances where this assumption isn?t true) then they will likely be poor matches and get not make it past your threshold for matching anyway.

Q: In RANSAC, the part of the algorithm that calls for $SSD(p_i^1, Hp_i)$, I am assuming this is talking about the locations of the features and not the feature descriptors since the ransac_est_homography doesn?t see the descriptors. At any rate, how is this SSD computed and why don?t we just use euclidean distance?

A: SSD on the coordinates would be the euclidean distance squared. There is little value in taking a square root because we have to set an arbitrary threshold anyway.

Q: For the mymosaic function, are there any assumptions that can be made about the order of the images passed into the function? Meaning, can I assume that the order that the images are passed in is the order I should be stitching? Or do we need to compare every image with each other to find the ones that match best?

A: Assume that the order they come in is the order they stitch together.