Assignment 2: Creating Panoramas

Before you start:
Download the assignment's files here.

The project is graded 1/3 written assignment and 2/3 code assignment.

Written Assignment:
Once you have finished the coding assignment (described next), you'll need to finish the written assignment. The written assignment is in the file "Written Assignment 2.docx".

Coding Assignment:
For this assignment you'll be implementing a panorama image stitcher. This requires the implementation of a Harris corner detector, RANSAC alignment and an image stitcher. Feel free to reuse any code from your last assignment.

In the project, you should only have to update one file "Project2.cpp. " The buttons in the UI will call the corresponding functions in "Project2.cpp. "

What to turn in:
To receive credit for the project, you need to turn in the completed file "Project2.cpp", the requested images for each task, a note about your implementation of extra points or anything special in your project, and the written assignment.

Tasks:

Step 1: Implement Harris corner detector. To do this, you'll need to write two routines:

a. SeparableGaussianBlurImage - This function is the same as the Gaussian blur function you wrote for Assignment 1, but it computes the values on a single channel floating point image. Your code should be nearly identical to the first assignment.
b. HarrisCornerDetector - Compute the Harris corner detector using the following steps:
    i. Compute the x and y derivatives on the image stored in "buffer" (see code.)
    ii. Compute the covariance matrix H of the image derivatives. Typically, when you compute the covariance matrix you compute the sum of (dx)^2, (dy)^2 and (dx)(dy) over a window or small area of the image. (NOTE: these are not second derivatives. These are first derivatives squared and multplication of derivatives in x and y directions.) To obtain a smooth result for better detection of corners, use the function SeparableGaussianBlurImage to get average for (dx)^2, (dy)^2, and (dx)(dy) in a Gaussian weighted window. Use the value of "sigma" as the standard deviation of the Gaussian.
    iii. Compute the Harris response using determinant(H)/trace(H).
    iv. Find peaks in the response that are above the threshold "thres", and store the interest point locations in "interestPts. "

Required: Open "Boxes.png" and compute the Harris corners. Save an image "1a.png" showing the Harris response on "Boxes.png" (you'll need to scale the response of the Harris detector to lie between 0 and 255.) When you're debugging your code, I would recommend using "Boxes.png" to see if you're getting the right result.

Open "Rainier1.png" with tab "Image 1" selected, and open "Rainier2.png" with tab "Image 2" selected. Compute the Harris corner detector. Save the images "1b.png" and "1c.png" of the detected corners (red crosses overlaid on the images, use DrawInterestPoints.)

Step 2: Implement MatchInterestPoints. To do this you'll need to follow these steps:
a. Compute the descriptors for each interest point. This code has already been written for you.
b. For each interest point in image 1, find its best match in image 2. The best match is defined as the interest point with the closest descriptor (L1-norm distance).
c. Add the pair of matching points to "matches".
d. Display the matches using DrawMatches (code is already written). You should see many correct and incorrect matches.

Required: Open "Rainier1.png" with tab "Image 1" selected, and open "Rainier2.png" with tab "Image 2" selected. Compute the Harris corner detector and find matches (press the two buttons in sequence). Save the images "2a.png" and "2b.png" of the found matches (green lines overlaid on the images, use DrawMatches).

Step 3: Compute the homography between the images using RANSAC (Szeliski, Section 6.1.4). Following these steps:

- a. Write Project(x1, y1, x2, y2, h). This should project point (x1, y1) using the homography "h". Return the projected point (x2, y2). Hint: See the slides for details on how to project using homogeneous coordinates.
- b. Write ComputeInlierCount(h, matches, numMatches, inlierThreshold). ComputeInlierCount is a helper function for RANSAC that computes the number of inlying points given a homography "h". That is, project the first point in each match using the function "Project". If the euclidian distance of the projected point to the second point is less than "inlierThreshold", it is an inlier. Return the total number of inliers.
- c. Write RANSAC (matches , numMatches, numIterations, inlierThreshold, hom, homInv, image1Display, image2Display). This function takes a list of potentially matching points between two images and returns the homography transformation that relates them. To do this follow these steps:
    - a. For "numIterations" iterations do the following:
        - i. Randomly select 4 pairs of potentially matching points from "matches".
        - ii. Compute the homography relating the four selected matches with the function "ComputeHomography. "
        - iii. Using the computed homography, compute the number of inliers using "ComputeInlierCount".
        - iv. If this homography produces the highest number of inliers, store it as the best homography.
    - b. Given the highest scoring homography, once again find all the inliers. Compute a new refined homography using all of the inliers (not just using four points as you did previously.) Compute an inverse homography as well (the fourth term of the function ComputeHomography should be false), and return their values in "hom" and "homInv".
    - c. Display the inlier matches using "DrawMatches".

Required: Open "Rainier1.png" with tab "Image 1" selected, and open "Rainier2.png" with tab "Image 2" selected. Compute the Harris corner detector, find matches and run RANSAC. Save the images "3a.png" and "3b.png" of the found matches (green lines overlaid on the images, use DrawMatches.) You should only see correct matches, i.e. , all the incorrect matches from the previous step should be removed. If you see all or some incorrect matches try running RANSAC with a larger number of iterations. You may try tuning the other parameters as well.

Step 4: Stitch the images together using the computed homography. Following these steps:

- a. Write BilinearInterpolation(image, x, y, rgb). This is what we've done in Assignment 1.
- b. Write Stitch(image1, image2, hom, homInv, stitchedImage). Follow these steps:
    - i. Compute the size of "stitchedImage. " To do this project the four corners of "image2" onto "image1" using Project ~~and "homInv"~~. Allocate the image.
    - ii. Copy "image1" onto the "stitchedImage" at the right location.
    - iii. For each pixel in "stitchedImage", project the point onto "image2". If it lies within image2's boundaries, add or blend the pixel's value to "stitchedImage. " When finding the value of image2's pixel use BilinearInterpolation.

Required: Open "Rainier1.png" with tab "Image 1" selected, and open "Rainier2.png" with tab "Image 2" selected. Compute the Harris corner detector, find matches, run RANSAC and stitch the images. Save the stitched image as "4.png". It should look like the image "Stitched.png".

Bell and Whistles (extra credit)

(Whistle = 0.01 point, Bell = 0.02 points)

Create a panorama that stitches together the six Mt. Rainier photographs, i.e. , Rainier1.png, ... Painier6.png. The final result should look similar to "AllStitched.png".

Create your own panorama using three or more images. You must capture the images yourself, and not find them on the web. I would recommend downsampling them before stitching, i.e. , make them approximately the same size as the images in the homework assignment.

🔔 Implement a new image descriptor or detector that can stitch the images "Hanging1.png" and "Hanging2.png". Save the Stitched image and the "match" images. In case you're wondering - No, you cannot rotate "Hanging2.png" by hand before processing, that's cheating.

🔔 Do a better job of blending the two images. That is, make the seams between the two images invisible. One possible method to do this is to use center-weighting. See Szeliski, Sections 9.3.2 - 9.3.4.

🔔 🔔 Implement a new image descriptor and/or detector that can stitch the images ND1.png and ND2.png. Save the Stitched image and the "match" images.