Greg Matthews
CS 583 – Section 900
April 19th, 2017

Computer Vision
Project 1: Homography

Rectification:

Objective:

- Manipulate an image with a planar surface, and create a fronto-parallel image of it

Steps:

1. Constructing the matrix A that serves as an intermediate to solve for the Homography matrix H.
2. Computing H by taking the dot product of A^TA, then solving the problem of total least squares, and finding the eigenvector of A^TA with the smallest eigenvalue. The eigenvector is then constructed into a 3x3 matrix.
3. Forward warping source corners points and establish output corner points
$$h(x, y) = (x', y')$$
4. Computing the offset of the output corner points, so there are no negative indices
5. Inverse warping source image to target coordinate frame and use bilinear interpolation to get the pixel value
$$h^{-1}(x^1, y^1) = (x, y)$$

Challenges:

- One of the biggest challenges of the experiment was knowing what to do with the offset that was calculated from the coordinates of the warped output image. I could have either recomputed the homography matrix H by mapping the source corner points to the newly offset output coordinates, I could pass the offset values for (x, y) into the $warp\_homography()$ function, but I felt that was a bit tedious. The option I went with was simply recomputing the inverse homography matrix, by mapping the warped rectification bounds to the source bounds.

- The other big issue I experienced is evident by the image shown below. My corner points seemed to map correctly, with all 4 corners filling up the bounding box, however I could not understand why my image was experiencing a “hazing” effect, and also mirroring near the top and left boundaries. I deduced that the hazing is probably resulting from the image mirroring, which is affecting the bilinear interpolation algorithm. I started asking the question of whether a homography of multiple warped points can index to the <u>same</u> source point…it couldn’t have. So there had to be something that is causing the redundancy, then it struck me that I didn’t account for the negative indexing that could result from me offsetting the image boundaries! By coding a conditional statement to check for negative indexing and simply making them black pixels, I solved my problem.

Rectification Issue:

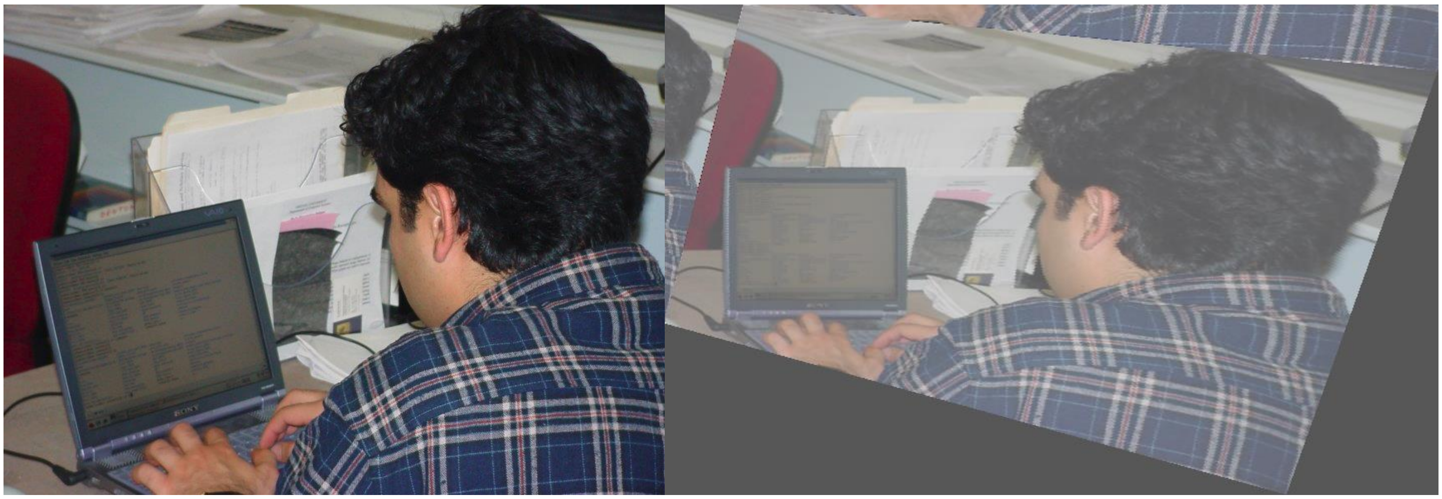


Image Rectification Results (Test Case):

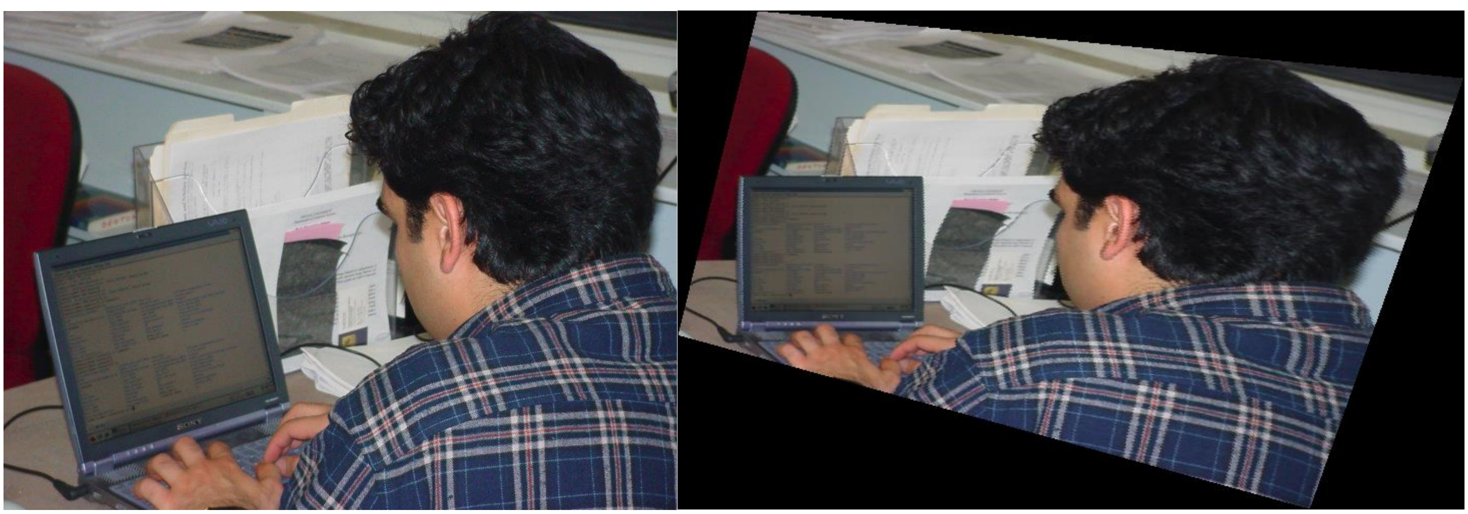
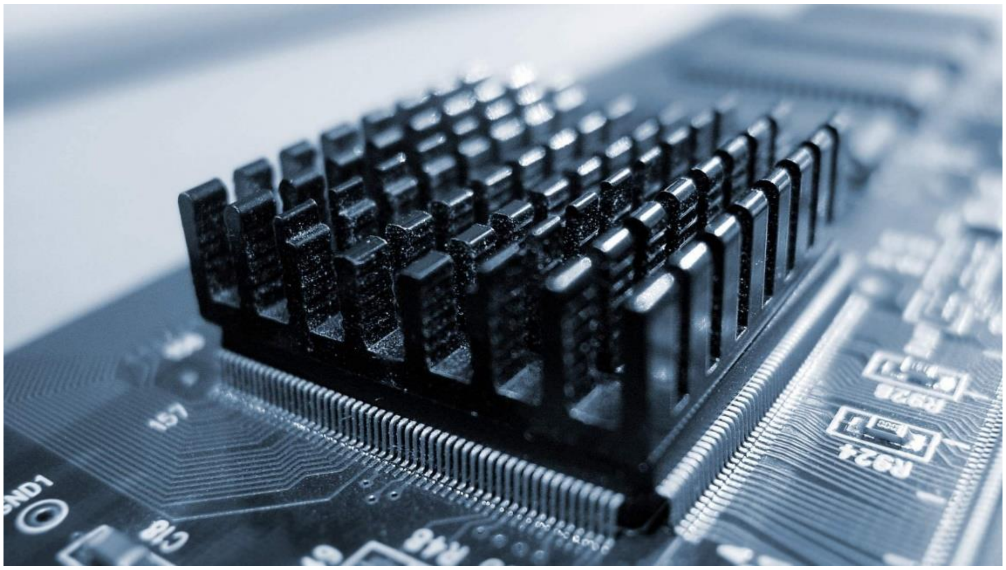
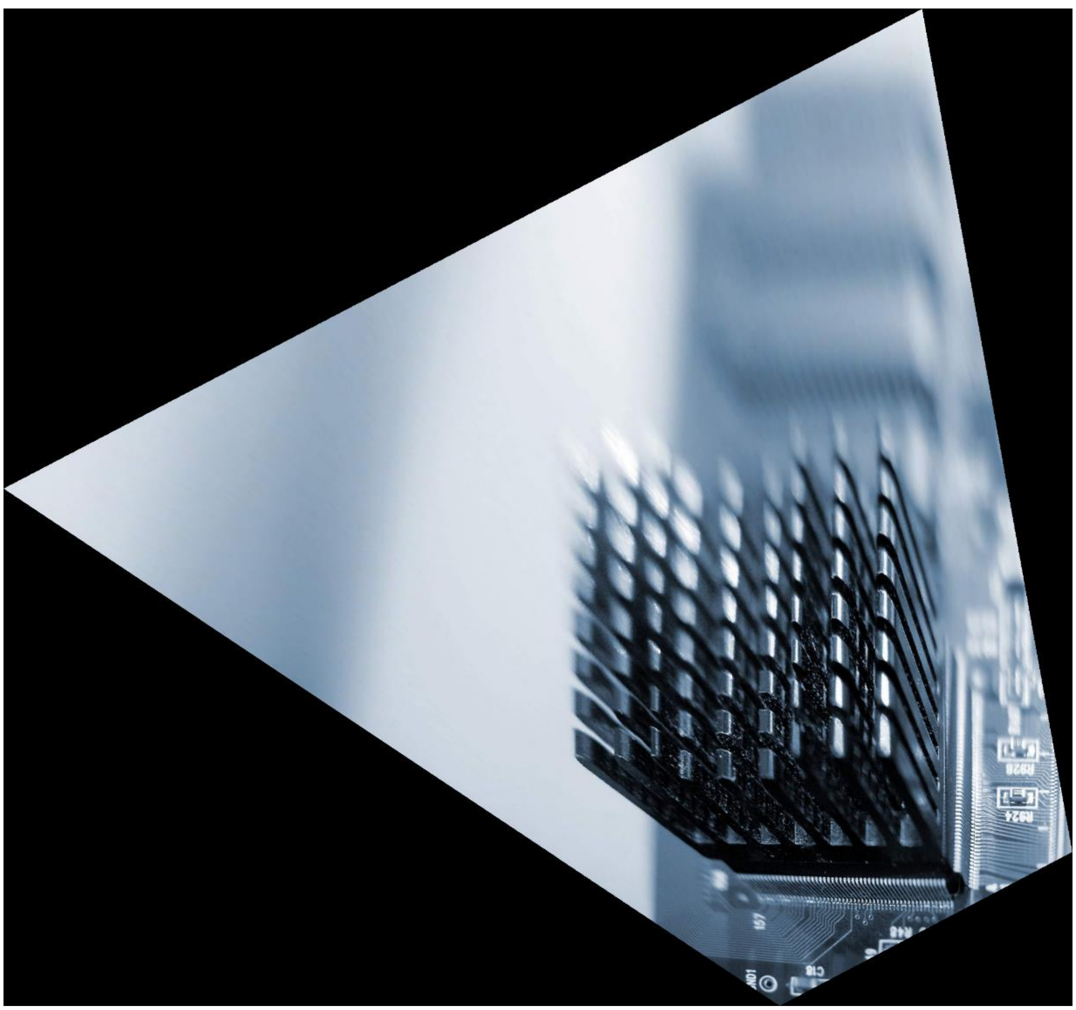


Image Rectification Results (Experiment 1):

Image Rectification Results (Experiment 2):

Image Rectification Results (Experiment 3):





Compositing:

<u>Objective:</u>

- Using 2 images, composite a part of one image onto another image by using the homography of corresponding regions

<u>Steps:</u>

1. Compute the homography that warps the rectification points to be composited to the source coordinate frame
2. Inverse warp both the image to be composited and the mask to the source image frame
3. Iterate through the mask indices, and check whether the pixel value is 1, 0, or intermediate, to choose whether to get the pixel value from the source of target image.

<u>Challenges:</u>

- The compositing part of the lab was very intuitive and logical, therefore it didn’t take me long to understand the process. The only real issues I experienced with this part of the lab, is checking for equality of an array with an array, as well as noting that the pixel values are floating point, for which I had to convert to integer to check for the 1/0 value case. Of course there’s other ways to do this, but that seemed the most intuitive for my implementation.

<u>Image Compositing Results (Test Case):</u>

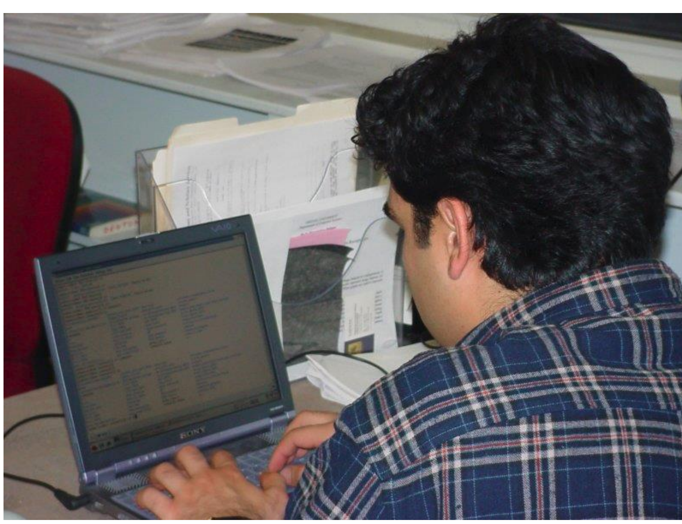



Image Compositing Results (Experiment 1):

Image Compositing Results (Experiment 2):