CS 180: Project 4

Image Warping and Mosaicing

Meenakshi Mittal

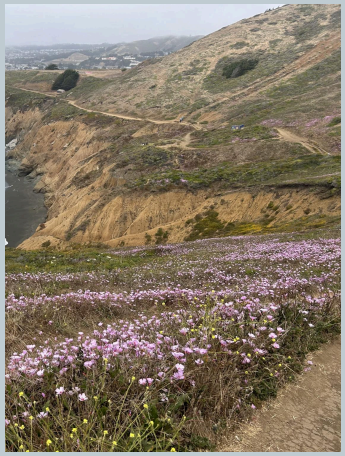
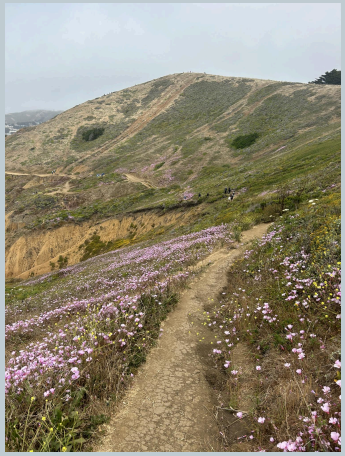
Project 4a:

In this project, we will stitch images together into mosaics. We will compile sets of images with some overlap between them, label correspondences between them, and use projective warping and blending to combine them into one image.

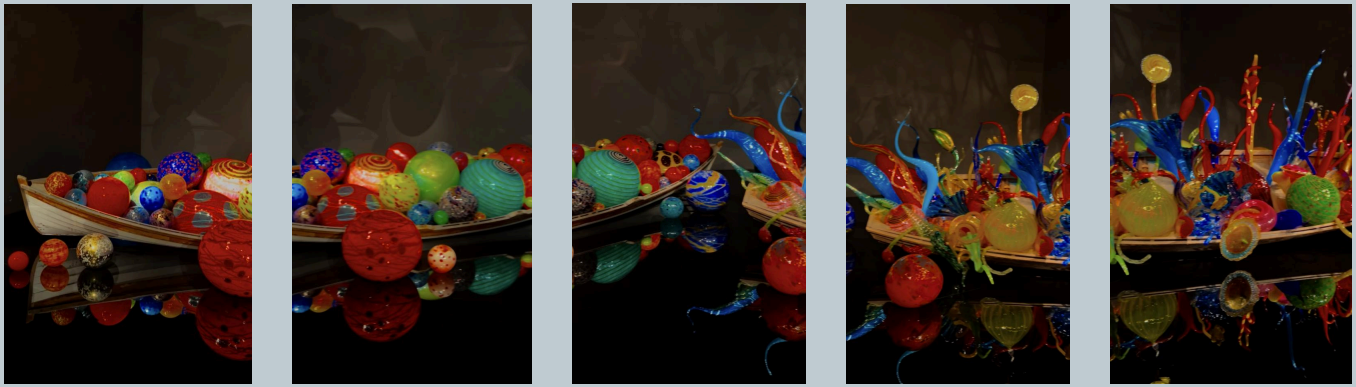
Part 1: Shoot and Digitize Pictures

First, I found 3 sets of images that I would like to mosaic together.

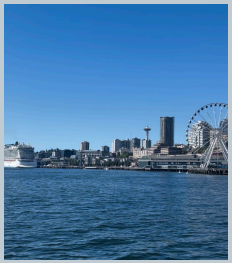
The first set consists of 3 images I took on a hike in Pacifica:

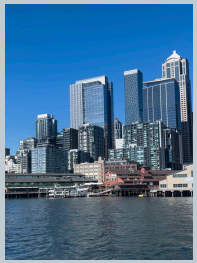
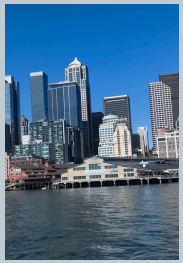




The second set consists of 5 screenshots of a panoramic video I took at the Chihuly glass museum in Seattle:



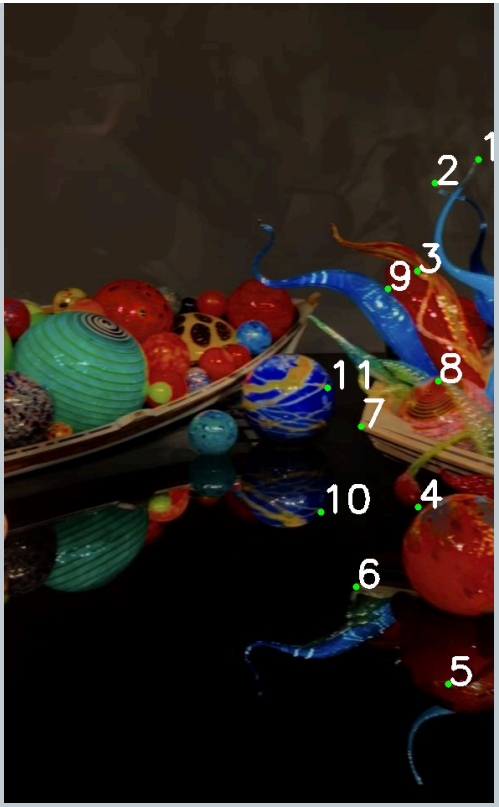
The third set consists of 6 screenshots of another panoramic video I took from a boat in Seattle:

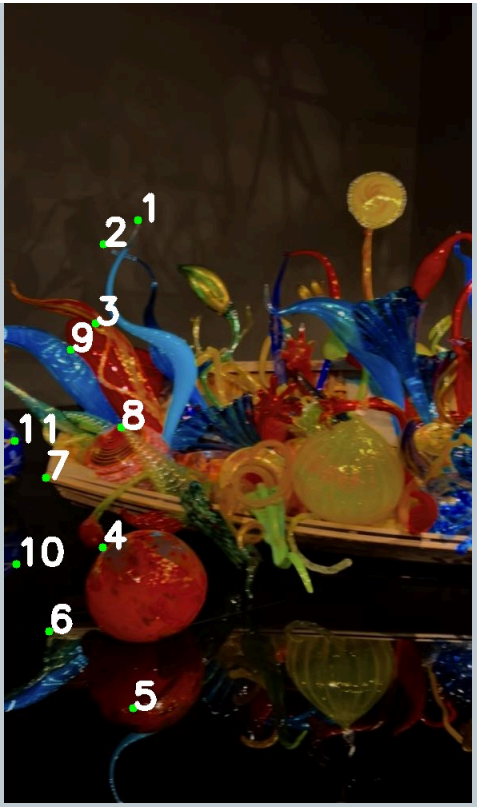



Part 2: Recover Homographies:

To compute the homography matrix between 2 images, we need to first define at least 4 correspondence points between them. Here is an example of labelled correspondence points between 2 of the glass museum images:





Once we have our labelled points, we use least squares regression to compute the homography matrix that best morphs one set of points to the other.

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$p' = H*p$$

Part 3: Warp the Images:

Now we can use the computed homography matrix to warp our images. We first identify the four corners of the original image and uses the homography matrix to determine where these corners will be located in the transformed image. This allows us to calculate the boundaries of the new image and create a new empty image of the correct size to later hold the transformed pixels. Then we use the inverse of the homography matrix to map each pixel in the output image to its nearest corresponding pixel in the original image. Finally, we map the color of the pixel from the original image back to the output image pixel.

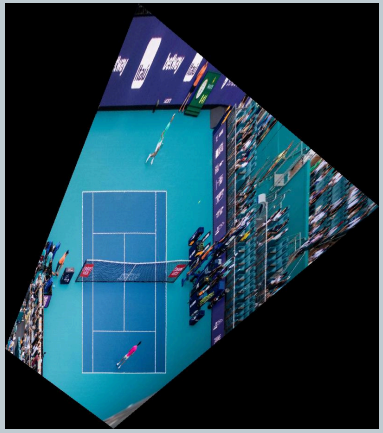
Below are a couple examples of image rectification-- in other words, we warp the original image to make the object appear rectangular.

For the laptop image, I found the dimensions of my laptop online and used those to construct a proportional rectangle. I warped the original image to the rectangle given by the coordinates [[0, 240], [0, 40], [280, 240], [280, 240]].

Original Laptop Image

Rectified Laptop Image

For the tennis court image, I found the dimensions of a tennis court online and used those to construct a proportional rectangle. I warped the original image to the rectangle given by the coordinates [[0, 390], [0, 0], [180, 0], [180, 390]].



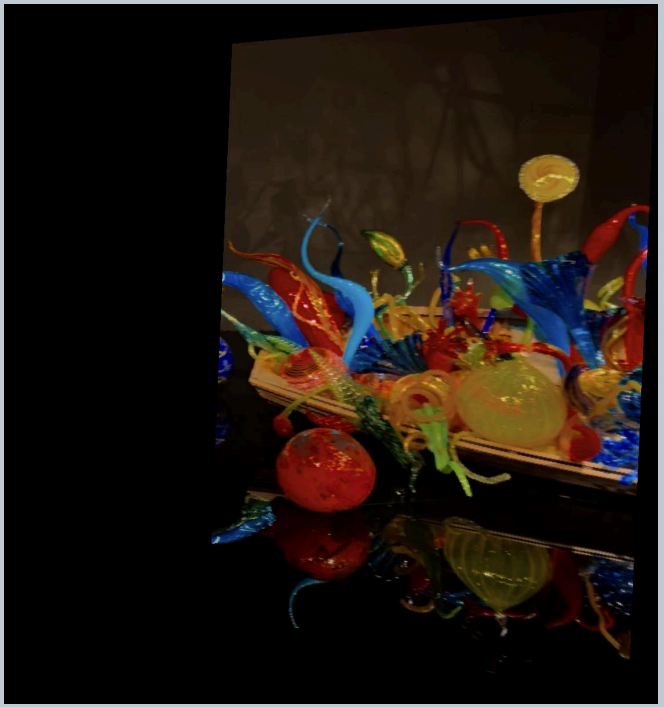
Original Tennis Court Image

Rectified Tennis Court Image

Looks like it works pretty well! We can move on to stitching our mosaics together.

Part 4: Blend Images into a Mosaic:

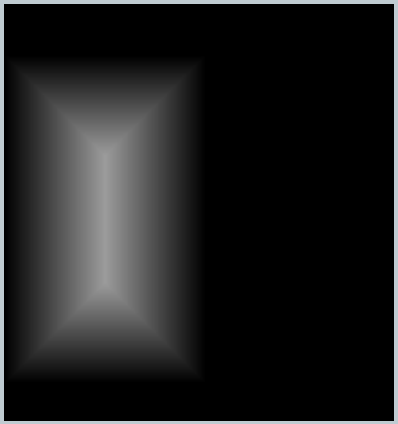
Now we can combine the ideas from above to morph our images together into a mosaic. I will demonstrate the stitching process using the 2 labelled images from the glass museum above. We will morph the 2nd image to the 1st one. First, we warp the 2nd image according to the computed homography matrix between the 2 images. We create a large enough bounding box to fit both images, and we can align them both in this box by the correspondence points.

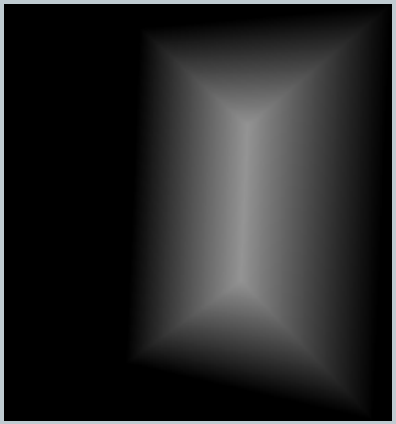


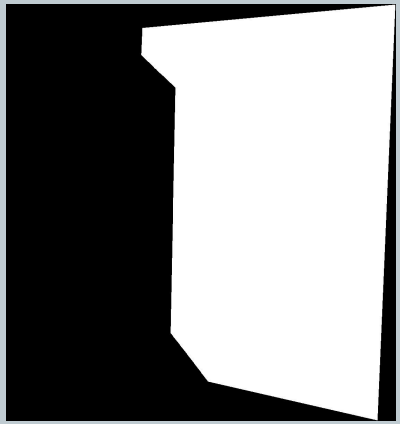
If we were to just place the images directly on top of one another, we would likely see some distinct lines separating the 2 images. Here is an example of the Seattle panorama stitched together this way, without any blending:

No Blending

We will use our Laplacian blending approach from Project 2 to smooth out these lines. We can construct our mask using the following trick: compute the distance transforms of both images, and set the mask to logical(dtrans1 > dtrans2). Here's what that looks like:





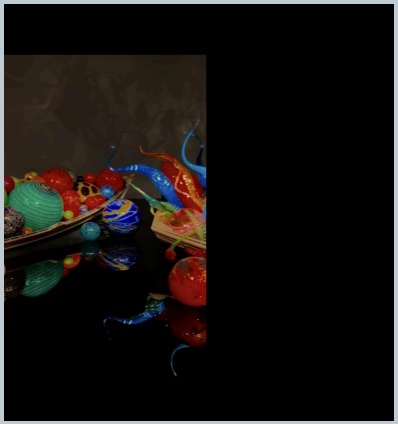
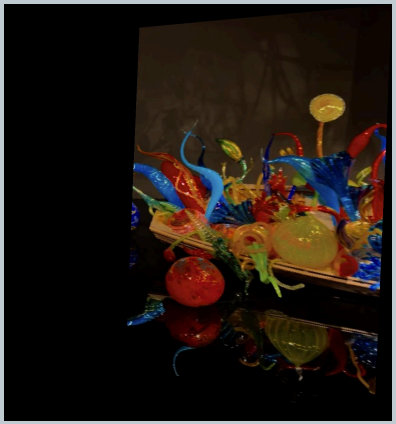
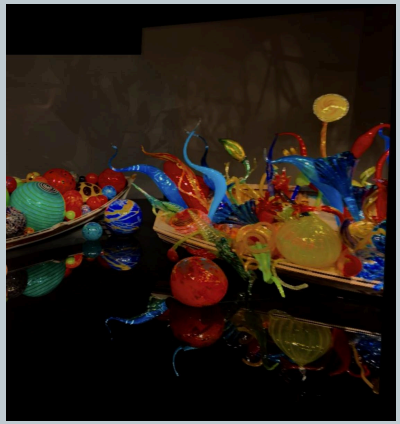


Distance Transform of 1st Image

Distance Transform of 2nd Image

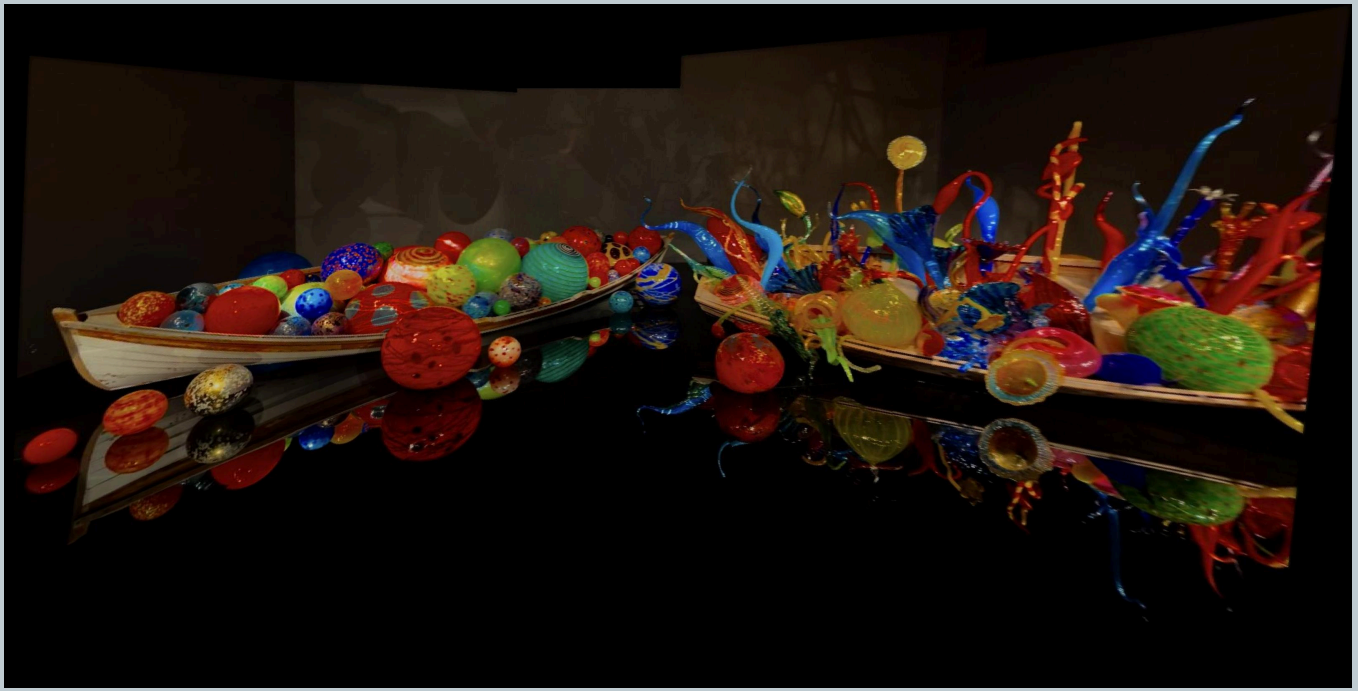
Mask

Now we can morph the images together! We use a simple 2 layer Laplacian pyramid with a Gaussian kernel size of 35.

Now we have everything we need to seamlessly blend our images into mosaics. I created the mosaics by adding one image to the mosaic at a time. For the glass museum images I started in the middle and worked outwards, while for the Seattle images I started at the edges and worked inwards. Here are the full stitched and blended panoramas for all 3 sets of images.

Pacifica



Glass Museum

Seattle

Project 4b:

Now, we will try to stitch the images automatically. This section closely follows the procedure used in "Multi-Image Matching using Multi-Scale Oriented Patches" by Brown et al. We will first detect Harris corners in the images we want to match, then use Adaptive Non-Maximal Suppression to get an even distribution of 500 points with high corner strength, then extract feature descriptors for each point and find the best matches between the feature descriptors for both images, and finally use 4-point RANSAC to compute the best homography between the images. Once we have the best homography, we can simply use the code from part 4a to stitch and blend the images together.

For the majority of this section, I will use the following 2 images from the glass museum photo set to demonstrate each step:

Image 1

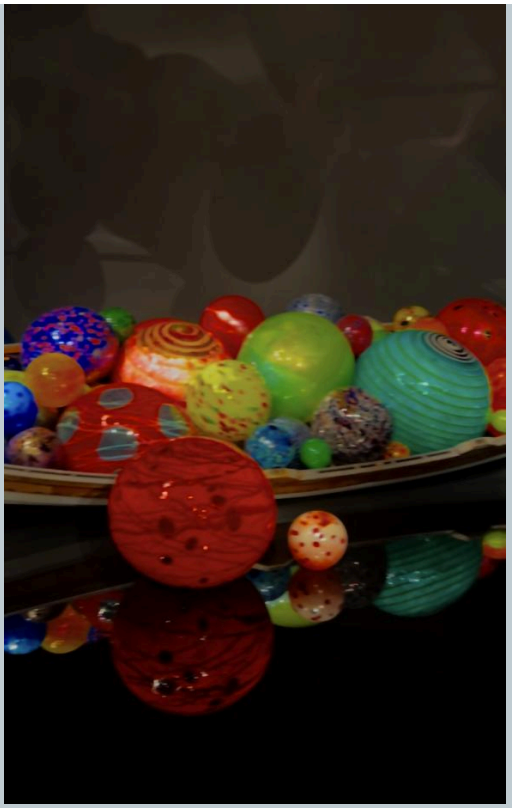


Image 2

Part 1: Harris Interest Point Detector

For this part, I simply used the provided code to find the Harris corners for each image, along with their corner strengths. Most of my images outputted a very high number of Harris corners. Here are the Harris corners overlaid on the glass museum images:

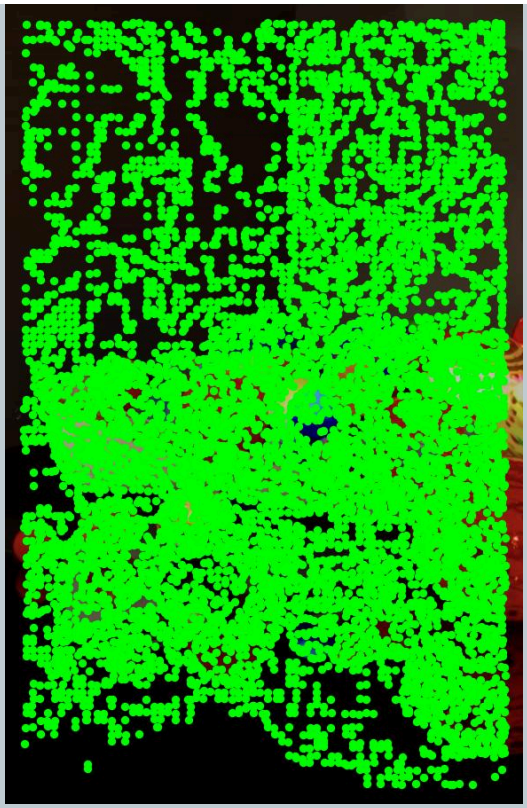


Image 1 Harris Corners

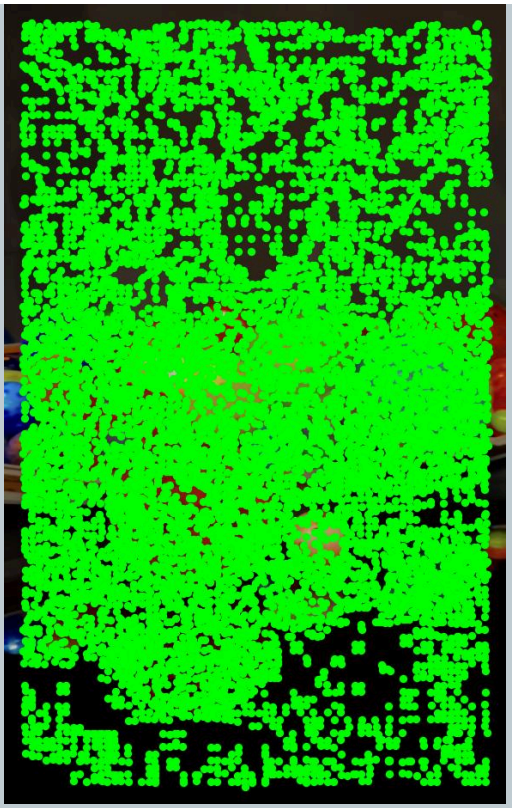


Image 2 Harris Corners

Part 2: Adaptive Non-Maximal Suppression

Clearly, this is way too many points to reasonably deal with. We want to somehow limit the number of candidate points. A first (misguided) approach would be to just increase the threshold for the corner strength, only keeping points with corner strength higher than the threshold. Here is what we get from this approach:

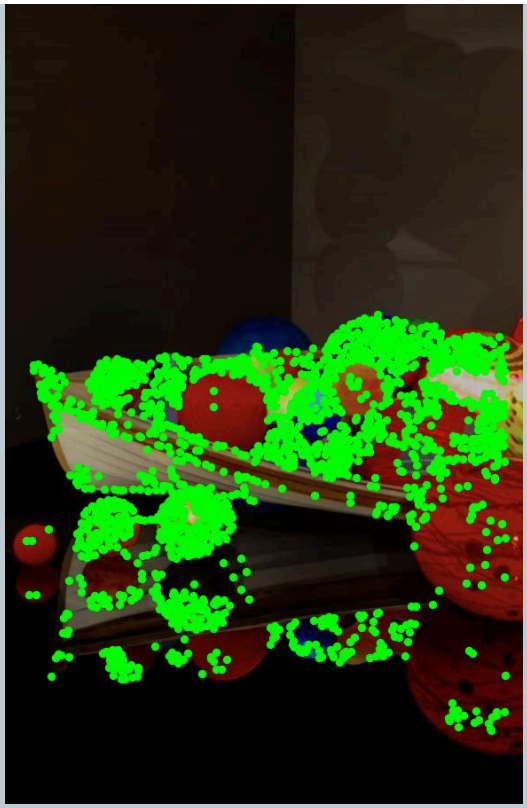


Image 1 Strongest Corners

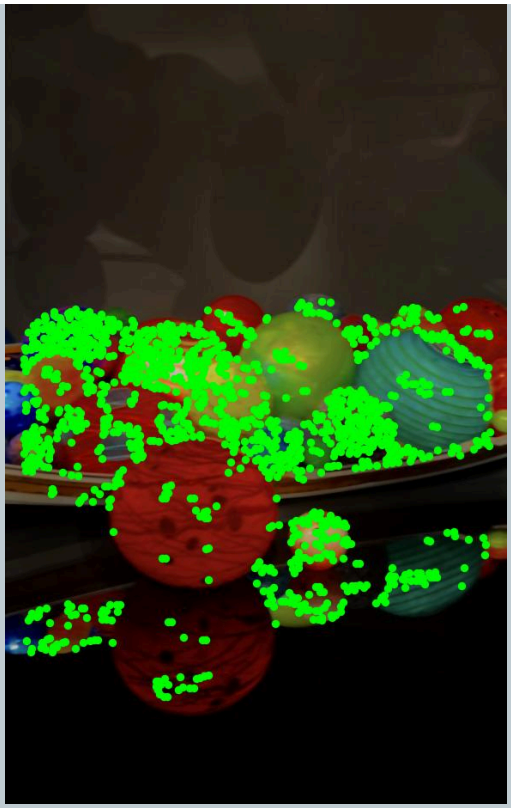


Image 2 Strongest Corners

This is still not very ideal. All the points are just clustered together now, which will likely cause us some issues down the road when we attempt to compute the homographies. From part A, we saw that we got the best results when we selected correspondence points that were spread throughout the overlapping region. To emulate this, we implement a technique from the paper known as Adaptive Non-Maximal Suppression. For each point, we compute the surrounding radius within which no other point has a higher corner strength. We keep some strength threshold for the scenario where a weaker point is to be placed within a stronger point's radius. Then we simply keep the 500 points with the largest radii. Here is what the ANMS strategy gives us:

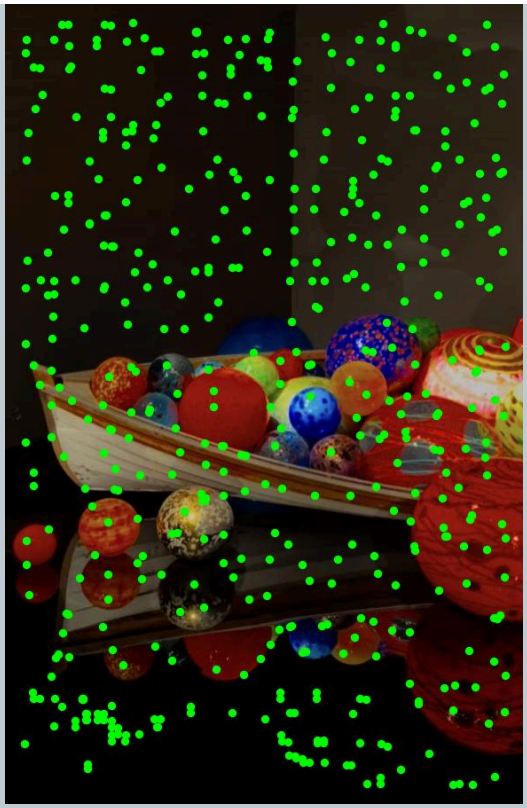


Image 1 ANMS

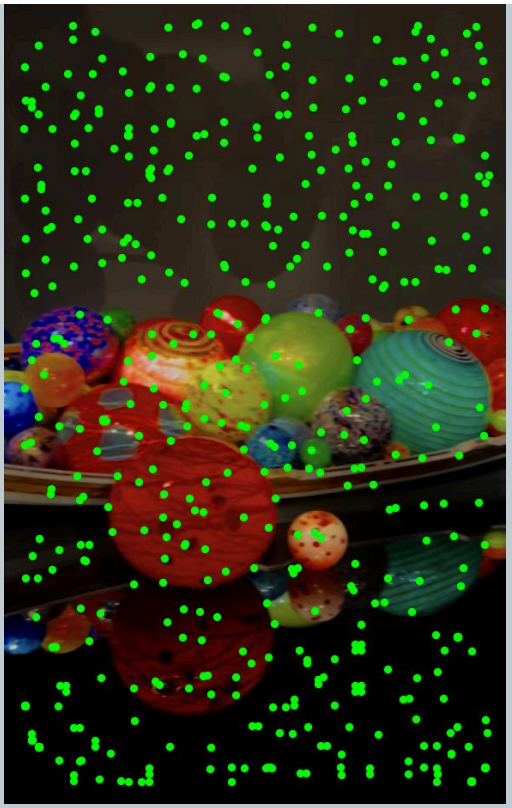
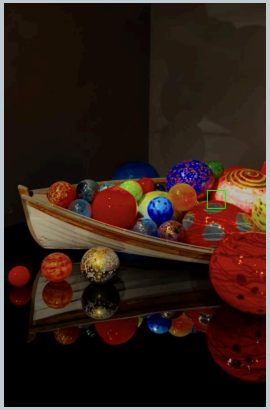


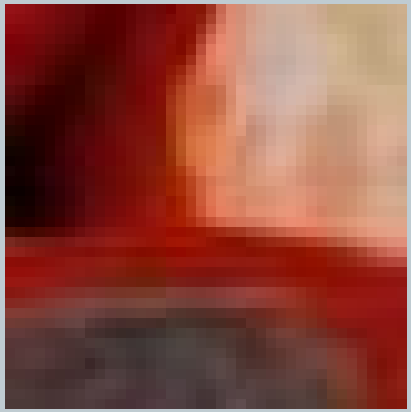
Image 2 ANMS

Works pretty well! Now we move on to matching the points between the two images.

Part 3: Extracting Feature Descriptors

For each of our remaining points, we want to extract feature descriptors that will allow us to compare and match points between the two images. We do this by extracting 40x40 pixel regions around each feature point, downsizing it to 8x8 pixels (with anti-aliasing), and then standardizing the resulting image (subtracting the mean and dividing by the standard deviation). Here is what that looks like:





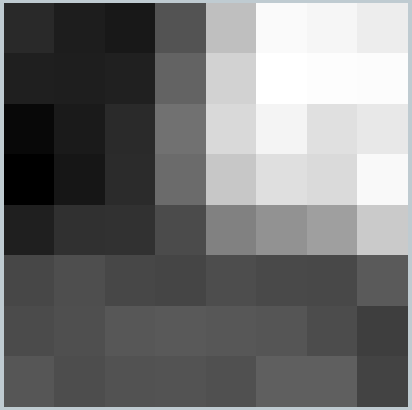


Image 1 Feature Box

Image 1 Feature Zoomed In

Image 1 Feature Scaled and Standardized

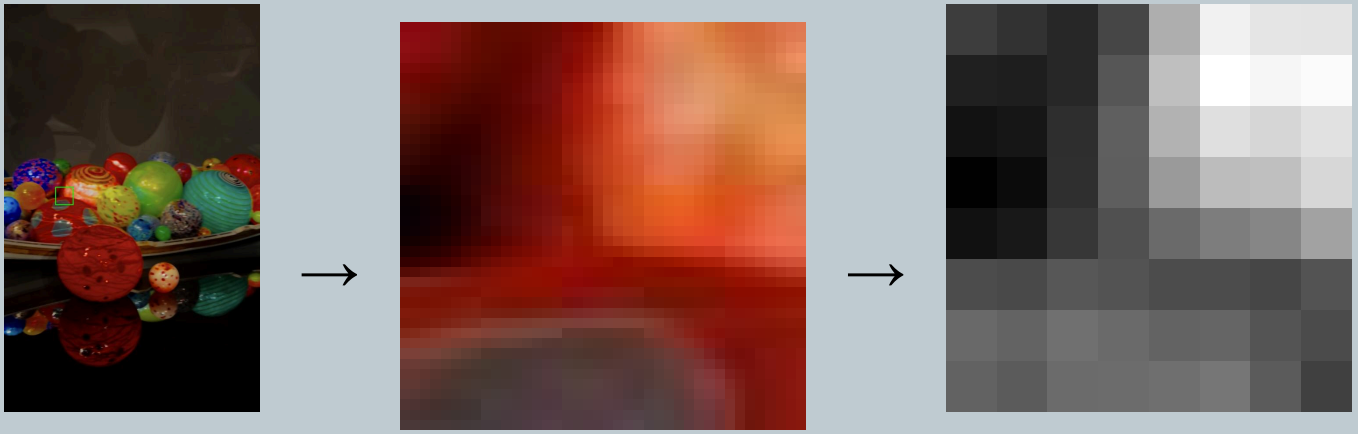


Image 2 Feature Box

Image 2 Feature Zoomed In

Image 2 Feature Scaled and Standardized

Part 4: Matching Feature Descriptors Between Two Images

Note that the feature selected for the above visualization is the same for both images. How did I find this feature? Our next step is to compute the squared distances between every feature descriptor for image 1 and every feature descriptor for image 2. In order to determine which points to keep and which to throw away, we have to be a bit careful. We could naively compute the nearest neighbor of each feature and keep all matches that are below some error threshold. However, this will result in a large number of incorrect matches, as seen in the paper. We instead use Lowe's trick, where we keep track of the top 2 nearest neighbors for each feature, and then compute the ratio 1-NN error / 2-NN error. Now, if we keep all matches for which this ratio is below some threshold (I chose 0.3, using the graph given in the paper), we find that most of the matches are correct. This is due to the fact that the feature points in one image may match very well to multiple different feature points in the other image. Going just off of the 1st nearest neighbor may give back one of the incorrect matches. On the other hand, the ratio 1-NN error / 2-NN error will only be

low if the 1st best match is good and the 2nd best match is poor and has high error. Thus, this method keeps only the "unique" matches between the images, which prevents storing similar-looking but incorrect matches.

Here are the matching points computed between the two images. We can see that the selected points line up quite well:

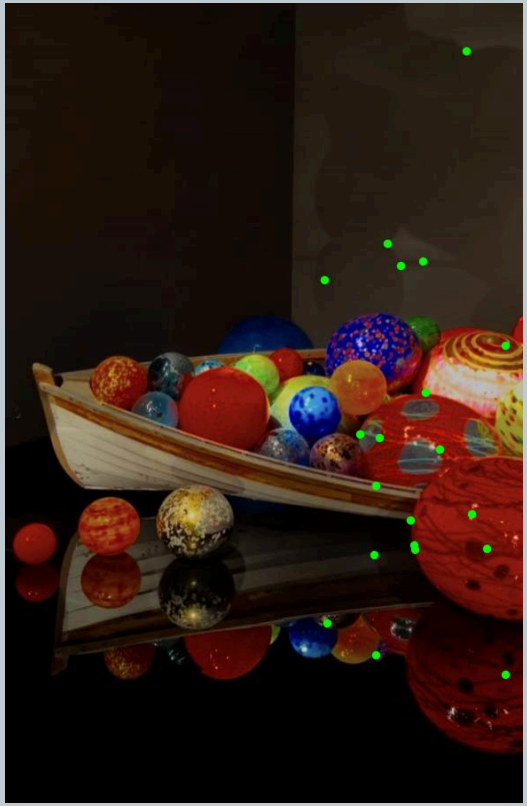


Image 1 Matches

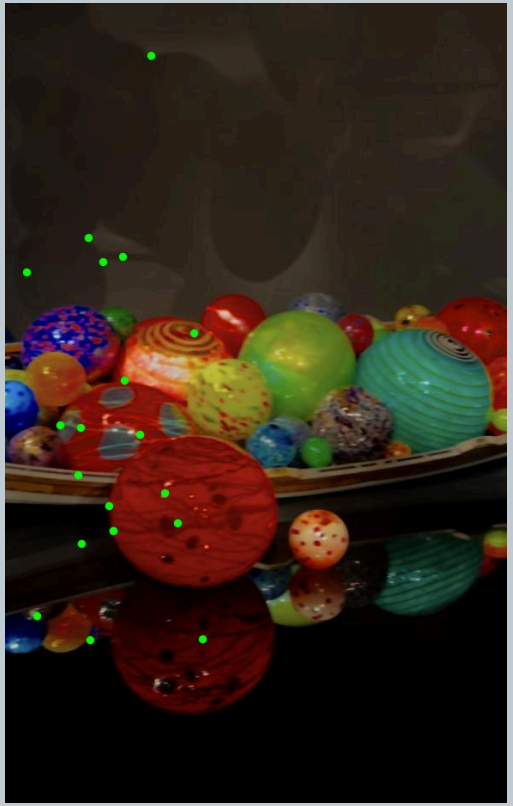
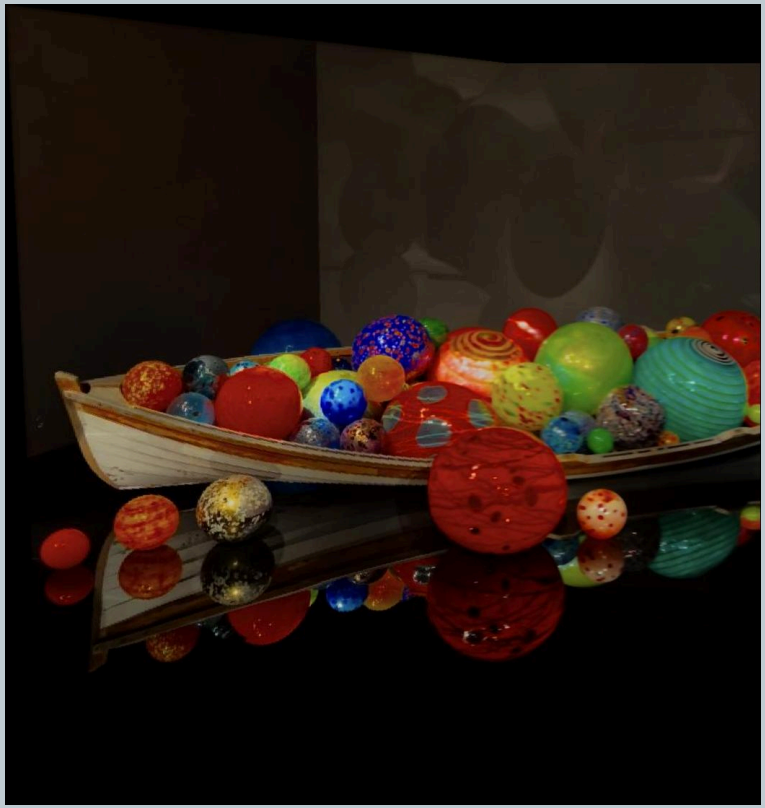


Image 2 Matches

Part 5: RANSAC to Compute Homography

We can now move on to finding the optimal homography matrix to mosaic the images together. While our feature matching method appears to work quite well, there is still a lot of room for error. We use a Random Sample Consensus (RANSAC) approach to mitigate the potential error. The approach is quite

straightforward: we randomly sample 4 of our matches, compute the homography matrix that transforms one set of the points to the other (reused from part A), and then count the number "inliers" for that homography. Inliers are point matches that agree with our homography-- in other words, a point match is an inlier if the distance between the transformed point from image 1 (H*p) and its corresponding point in image 2 (p') is "small" (I chose a threshold of 4 pixels). We repeat this sampling process many times (I chose 10000), and keep the homography that has the most inliers. Then we simply use our stitching process from part A to mosaic the images together. Here are the 2 images finally stitched together:



Mosaiced Imaged

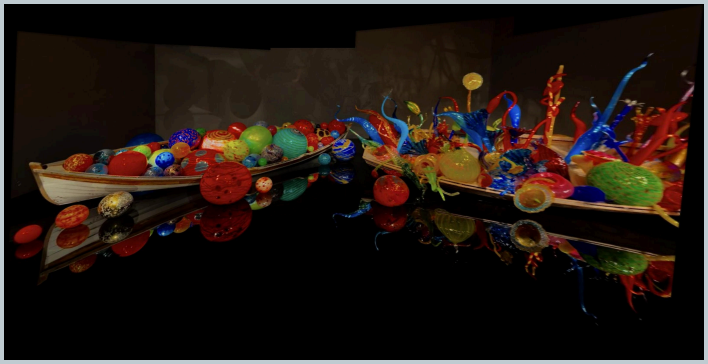
Part 6: Making Mosaics

We will recreate the mosaics from part A, this time using the automatic stitching technique. We use the same blending techniques from part A, using the same mask and the same Laplacian pyramid approach.

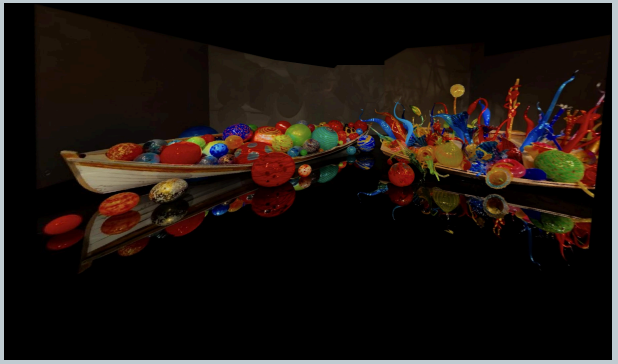
Here are the results side by side:

Manually Stitched Pacifica

Automatically Stitched Pacifica



Manually Stitched Glass Museum



Automatically Stitched Glass Museum

Manually Stitched Seattle

Automatically Stitched Seattle

The End!

We have finally reached the end of our mosaic stitching journey. I am quite pleased with the results-- I prefer the automatically stitched mosaic over the manual one in some cases, particularly in the Seattle mosaic. Also, in the Pacifica mosaic, there is a pretty noticeable misalignment in the manually stitched version in the mountain in the far distance. The automatically stitched version does not contain this artifact. I think this whole project was very cool and very fun to implement. I mostly appreciate the use of clever algorithms and tricks in the paper we replicated, as we get impressive looking results without an excessive amount of compute or complexity. I like that a lot of the ideas we used, like ANMS, feature matching, Lowe's trick, and RANSAC, are not necessarily the most straightforward, but each one is very intuitive and makes sense in the process. I also like that the project was based around replicating a research paper on our own, which is arguably a valuable skill for any path one may choose to go down.