Panoramic Image Generation

Alex Rasla
UC Santa Barbara
alexrasla@ucsb.edu

1 Algorithms

Throughout this process, I implemented my own algorithms to compute homography matrices and warp images on top of eachother. My homography algorithm is relatively standard: I used the SIFT feature detector created by OpenCV to match key features between overlapping images, and then computed the best homography matrix between the two images using the RANSAC sampling method. Using this homography matrix, I was able to warp the current image onto the base image, or use matrix multiplication to warp the image onto the panoramic image if it already exists. The warping process was more complex: I needed to correctly map subsequent images to the base image by multiply the previous homography matrices and keep track of the shifts in the width and height boundaries when adding this new projective image. These shifts along the axis were a result of each subsequent projective image having a larger height or width than the previous. This homography computation and warping process was first done by starting from the base image, which I chose to be the image in the middle of the dataset, and adding 5 images to the right, and then adding 5 images to the left of the base image.

2 Data

In order to test my implementation, I used the "family_house" dataset on Gauchospace. Once my implementation was finished, I extensively tested my stitching program on the rest of the datasets provided such as west_campus, yellowstone2, ucsb4, etc. Finally, once my image was able to stitch these datasets together correctly, I created my own dataset using 10 pictures I took from my apartment balcony. From these, I created my own unique panoramic image which is shown in 3.

3 Libraries

In order to match features between two images, I used OpenCV's SIFT detector. This feature detector was able to create accurate matches between consecutive, overlapping images. To create my own panoramic images, I used numpy vectorization techniques to do pixel slicing. This was necessary in order to speed up my runtime from several minutes to $<= 30$ seconds from my initial nested for loop implementation.

4 Implementation Details

In order to compile and run my program, simply run the command below:

```
python3 stitching.py --images [images directory]
```

The program stitches 10 images together because when trying to stitch more, the projective images become extremely large and stretched which lowers the overall quality of the panoramic image. To do this, it first sorts the image in numerical order, takes the middle image of the dataset as the base image, then stitches 5 images to the right of the base and 5 images to the left of the base respectively. My program is able to stitch more than 10 images together, but the original base image becomes too small to see, and the final panoramic image pixel sizes become on the order of 10^4 because of the warped images.

5 Results

Figure 1, 2, and 3 show some of my image stitching results from both Gauchospace datasets and my own. The runtime to stitch 10 images is around 30 seconds, and the homography is quite accurate as we can see with the correct projective images. Occasionally, the RANSAC algorithm does not produce an accurate homography matrix which creates a poor panoramic image.

Figure 1: Panoramic image generated from the family_house dataset on Gauchospace

Figure 2: Panoramic image generated from the west_campus1 dataset on Gauchospace

6 Discussion

Throughout this experiment, I encountered many problems with calculating the homography and warping the images; however, solving these problems made me truly understand the process of how images are stitched. When first calculating the homography matrix, I assumed the top 50 feature matches with the least distances were the best points to use. However, I quickly noticed the projective images' bounding boxes calculated from the homographies were very inaccurate. Thus, I researched and implemented the RANSAC algorithm on top of the homography matrix calculations in order to get the best one, which significantly improved the homography matrix's accuracy and consistency.

Once I calculated an accurate homography matrix and mapped one projective image onto a base image, I assumed the process would be easily extended to multiple images. However, I quickly realized it was substantially more difficult and computationally intensive. Thus, I research different image mapping techniques, and ultimately decided to use matrix multiplication to calculate mappings between two non-consecutive images like $H_{13} = H_{12} \times H_{23}$. Once this was solved, I had to ensure the previous panoramic image was shifted correctly along both the x and y axis with the new taller and wider projective image. This was an extremely tedious process because I had to keep track of the previous panoramic image shifts along each dimension and how many pixels to expand the next panoramic image by. I also noticed that when warping some of the projective images, the loss in accuracy caused by each matrix multiplication was compounded onto the next images which sometimes caused poor final results.

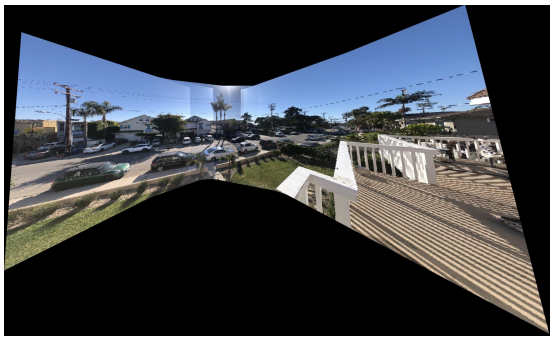

Figure 3: Panoramic image generated from my own personal dataset

Once I was able to correctly stitch multiple images, I had to deal with slow runtime issues. My initial implementation had a nested for loop that accessed each pixel, checked if it was in the projective image's bounding box, and changed its pixels accordingly. This quickly proved to be extremely computationally heavy, especially when reaching image sizes over 10^3 pixels on a dimension. As a result, I used numpy array slicing (vectorization) to both add the previous panoramic image to the new one and overlay the new projective image. In future iterations, I would have liked to include cylindrical coordinate mapping to reduce image sizes to speed up runtime even further, and linear blending to smoothen the seams between two images and increase the quality of the panorama.