CS543/ECE549 Assignment 3

Name: Hongbo Zheng
NetId: hongboz2

Part 1: Homography estimation

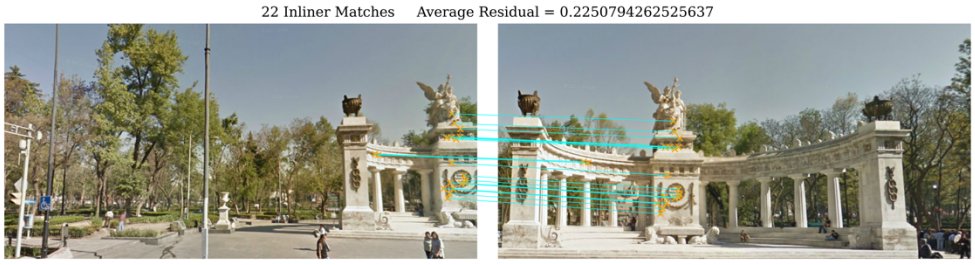
A: Describe your solution, including any interesting parameters or implementation choices for feature extraction, putative matching, RANSAC, etc.

First of all, both images are loaded and converted to grayscale. Then, the SIFT detector is used with the commands `cv2.SIFT_create()` and `sift.detectAndCompute` to acquire the feature points in both images since it's more robust than the Harris detector. After that, `scipy.spatial.distance.cdist` is used to compute the pairwise distance between the two descriptors. Finally, only the coordinates pairs that have the distance between them less than the distance threshold are kept, while the others are filtered out because their distances are larger than distance threshold.

Then, the RANSAC algorithm is run on the selected coordinates pairs returned by sift detector, which have distance less than distance threshold. The RANSAC algorithm is performed with 10000 iterations and a threshold of 0.75. Additionally, more iterations are also tried, but it does not show significant improvement. In each iteration of RANSAC, 4 pairs of points are randomly chosen from the filtered coordinates pairs. The matrix A is constructed according to the formula in the lecture with the 4 pairs of points. Then, matrix A is solved, and the singular vector with the smallest singular value is selected. The singular vector is then reshaped into 3x3 to construct the candidate Homography matrix H. After that, the matrix H is used to transform the key points into new coordinates. Then, the l2-norm is computed between the transformed coordinates with the original matched coordinates. If the l2-norm is less than the threshold, this pair is an inlier. The entire process is repeated with the number of iteration times, and the matrix H that has the maximum number of inlier is returned.

In the beginning, each image is put on a huge canvas, so there won't be overflow when images are stitched together. The Homography matrix H is passed to the function `cv2.warpPerspective` to apply perspective transformation to one of the images, called it image 0. Then, the perspective transformation of image 0 mask is generated to know where image 0 is located on the canvas. Similarly, image 1 mask is generated to know its location. After that, the overlapped region of image 0 and image 1 is calculated with `image 0 mask & image 1 mask`. Then, the overlapped region in image 1 mask is set to False to avoid plotting image 1 on top of image 0. After that, image 1 (excluding the overlapped region) is stitched with image 0 on the canvas. Finally, the min and max of nonzero coordinates are found to crop out the extra black borders of the canvas.

B: For the image pair provided, report the number of homography inliers and the average residual for the inliers. Also, display the locations of inlier matches in both images.



C: Display the final result of your stitching.

Part 2: Shape from shading

A: Estimate the albedo and surface normals

1) Insert the albedo image of your test image here:

yaleB01	yaleB02
	
yaleB05	yaleB07
	

2) What implementation choices did you make? How did it affect the quality and speed of your solution?

In the step “preprocess the data”, the following three steps are implemented:

1. Subtract the ambient image from each image

```
processed_imarray = imarray - ambimage[:, :, numpy.newaxis]
```

2. Set any negative values to zero

```
processed_imarray = numpy.clip(a=processed_imarray, a_min=0, a_max=255)
```

3. Normalize the resulting intensity to be between 0 to 1.

```
processed_imarray /= 255.0
```

This could potentially improve the runtime of the albedo, surface normals, surface height map computations and ensure numerical stability.

In the step “Estimate the albedo and surface normals”,

1. First reshape the imarray to `(hxw) x N` so that imarray is a 2d array, and each column represents an image with `hxw` features

```
imarray = numpy.reshape(a=imarray,
newshape=(imarray.shape[0]*imarray.shape[1], imarray.shape[2]))
```

2. Then get the g matrix by solving the least-squares solution of a linear system

```
g, res, rank, s = scipy.linalg.lstsq(a=light_dirs, b=imarray.T,
                                     cond=None)
```

3. Compute the `albedo_image` (norm of the g matrix), and use it to calculate the `surface_normals`

```
albedo_image = scipy.linalg.norm(a=g, axis=2)
surface_normals = g/albedo_image[:, :, numpy.newaxis]
```

Computing `abledo_image` and `surface_normals` in matrix form is much faster than looping through every pixels

In the step “Compute the surface height map by integration”,

1. Obtain partial derivatives

```
fx = surface_normals[:, :, 0]/surface_normals[:, :, 2]
fy = surface_normals[:, :, 1]/surface_normals[:, :, 2]
```

2. Row method: cumsum each row[0] + cumsum of entire fy

```
height_map = numpy.cumsum(a=fx, axis=1)[0] + numpy.cumsum(a=fy, axis=0)
```

3. Column method: cumsum of each row + cumsum of entire fy[: 0]

```
height_map = numpy.cumsum(a=fx, axis=1) +
numpy.reshape(a=numpy.cumsum(a=fy, axis=0)[:, 0], newshape=(-1, 1))
```

4. Average method: average of row method and column method

5. Random method

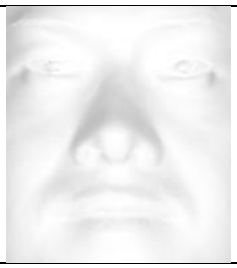
```
Initialize height map = numpy zeros with shape surface normals' h x w
Loop through h:
    Loop through w:
        Skip (0,0)
        Iterate number of paths to explore times:
            Generate path to destination (y, x)
            0 – fx
            1 – fy
            # of 0 = x
            # of 1 = y
            Path = 1d array of x 0s and y 1s
            Random shuffle Path
            Cumulative sum values and put into height map [y, x]
Height map /= number of paths to explore
```

3) What are some artifacts and/or limitations of your implementation, and what are possible reasons for them?

W-shape mouth appears when row integration method is performed. An example of it is shown below with subject yaleB07. This happens probably because the shading surrounds the nose and mouth makes the height map of the nose and mouth abnormal.

Weird chin slope happens when column integration method is performed. An example of it is shown below with subject yaleB07. This appears probably because the shading around the chin, the bottom left and bottom right of the image, makes the height map of the chin abnormal.

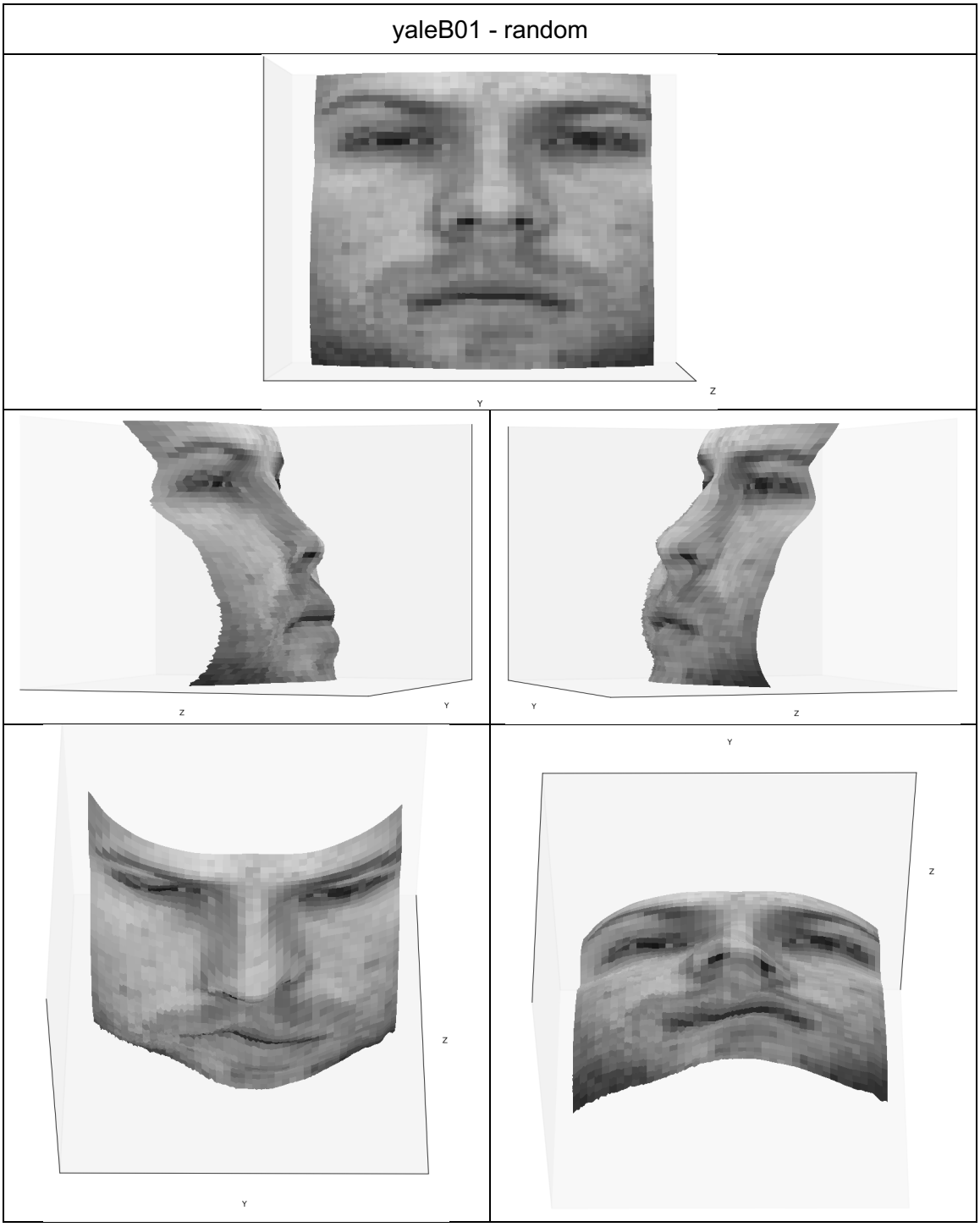
4) Display the surface normal estimation images below:

yaleB01 normal x	yaleB01 normal y	yaleB01 normal z
		
yaleB02 normal x	yaleB02 normal y	yaleB02 normal z
		
yaleB05 normal x	yaleB05 normal y	yaleB05 normal z
		
yaleB07 normal x	yaleB07 normal y	yaleB07 normal z
		

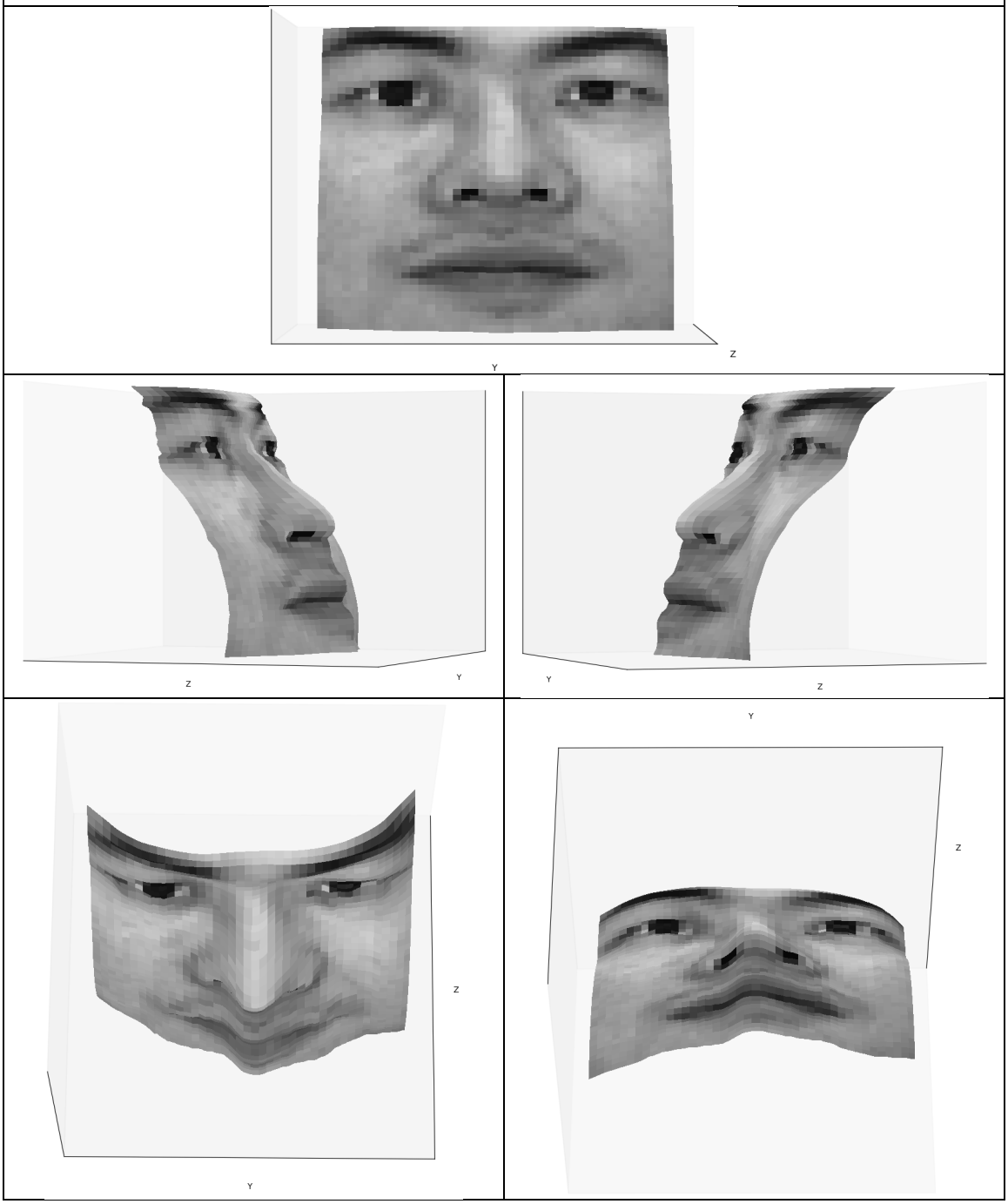
B: Compute Height Map

5) For every subject, display the surface height map by integration. Select one subject, list height map images computed using different integration method and from different views; for other subjects, only from different views, using the method that you think performs best. When inserting results images into your report, you should resize/compress them appropriately to keep the file size manageable -- but make sure that the correctness and quality of your output can be clearly and easily judged.

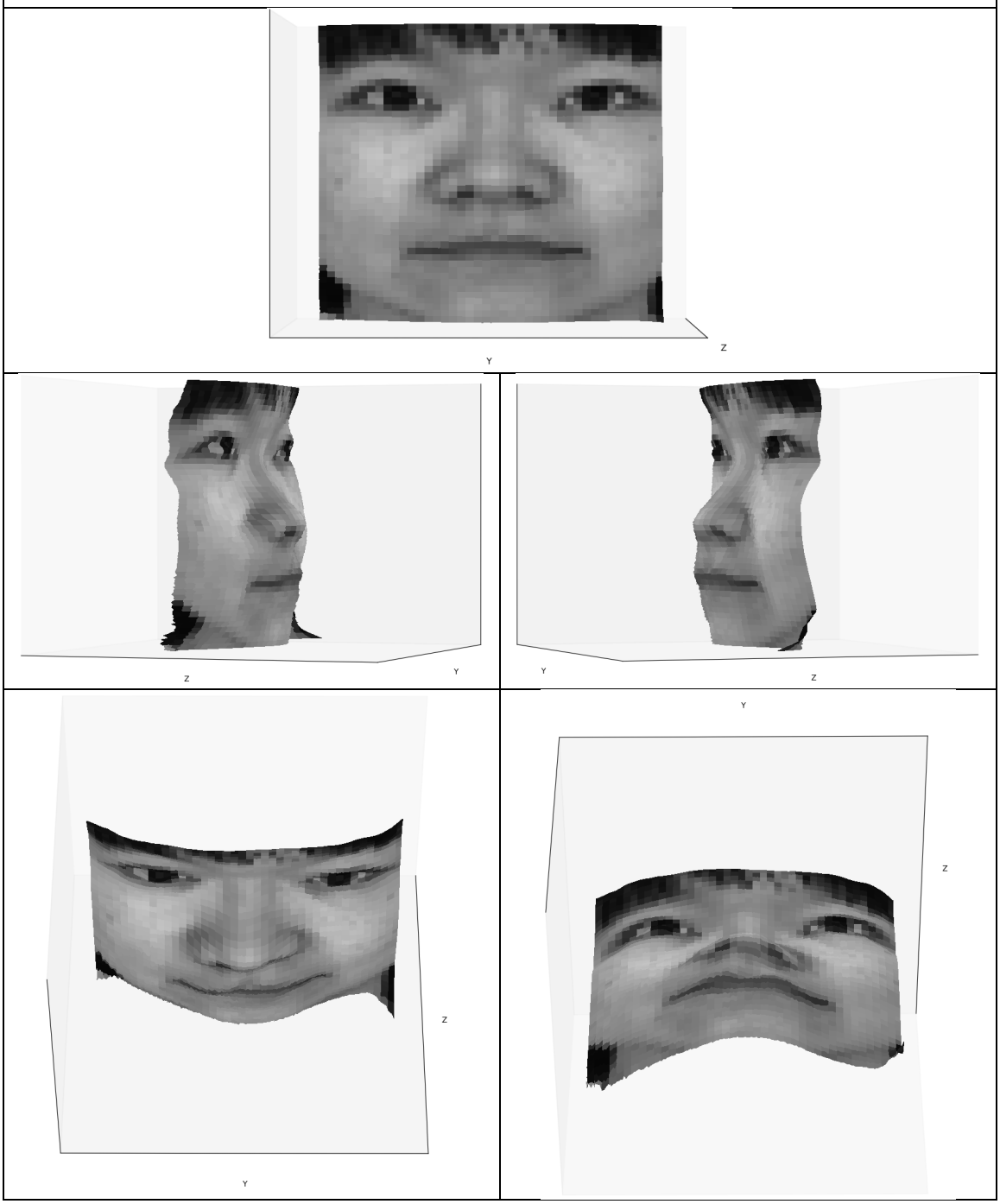
yaleB01 - random



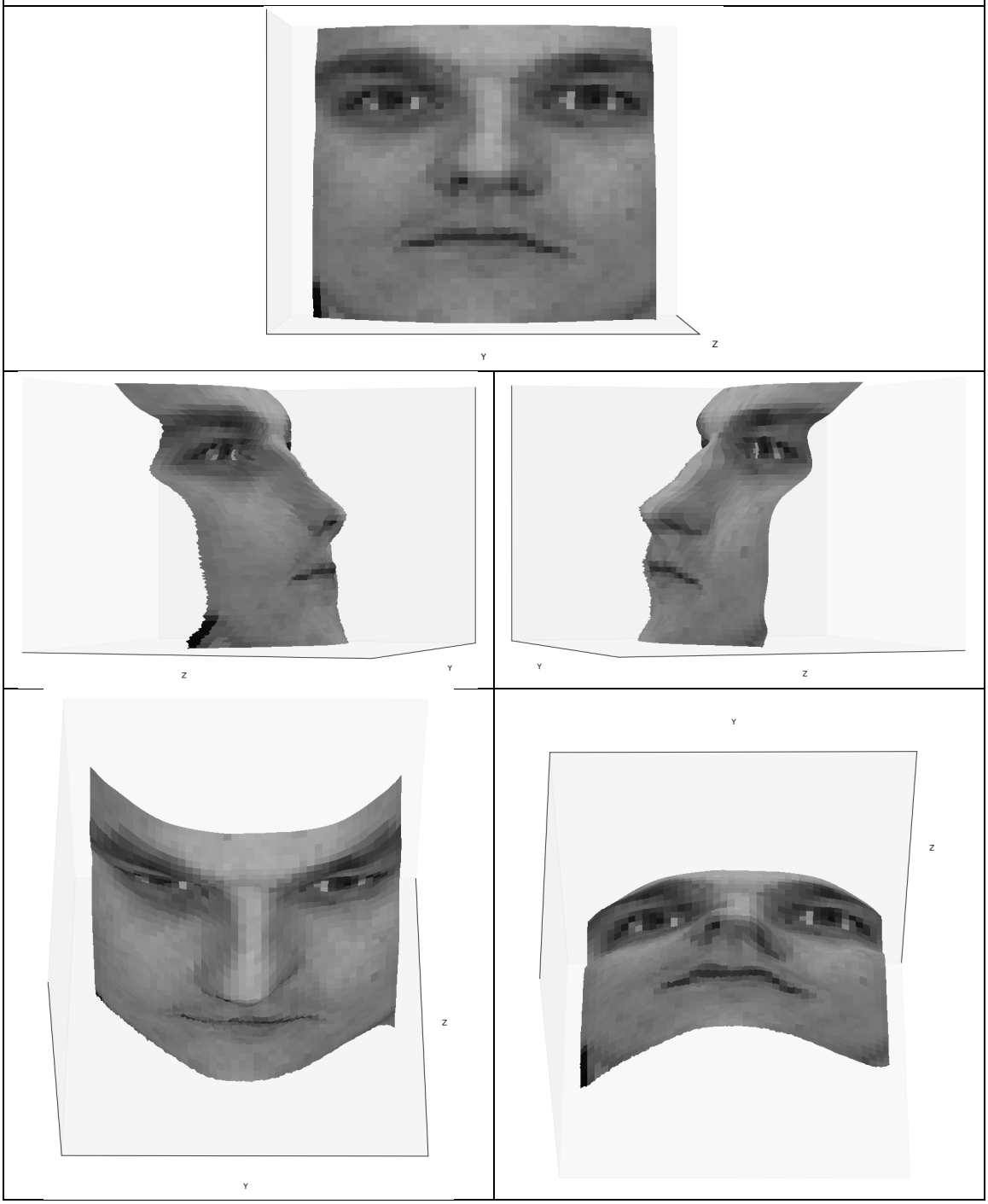
yaleB02 - random



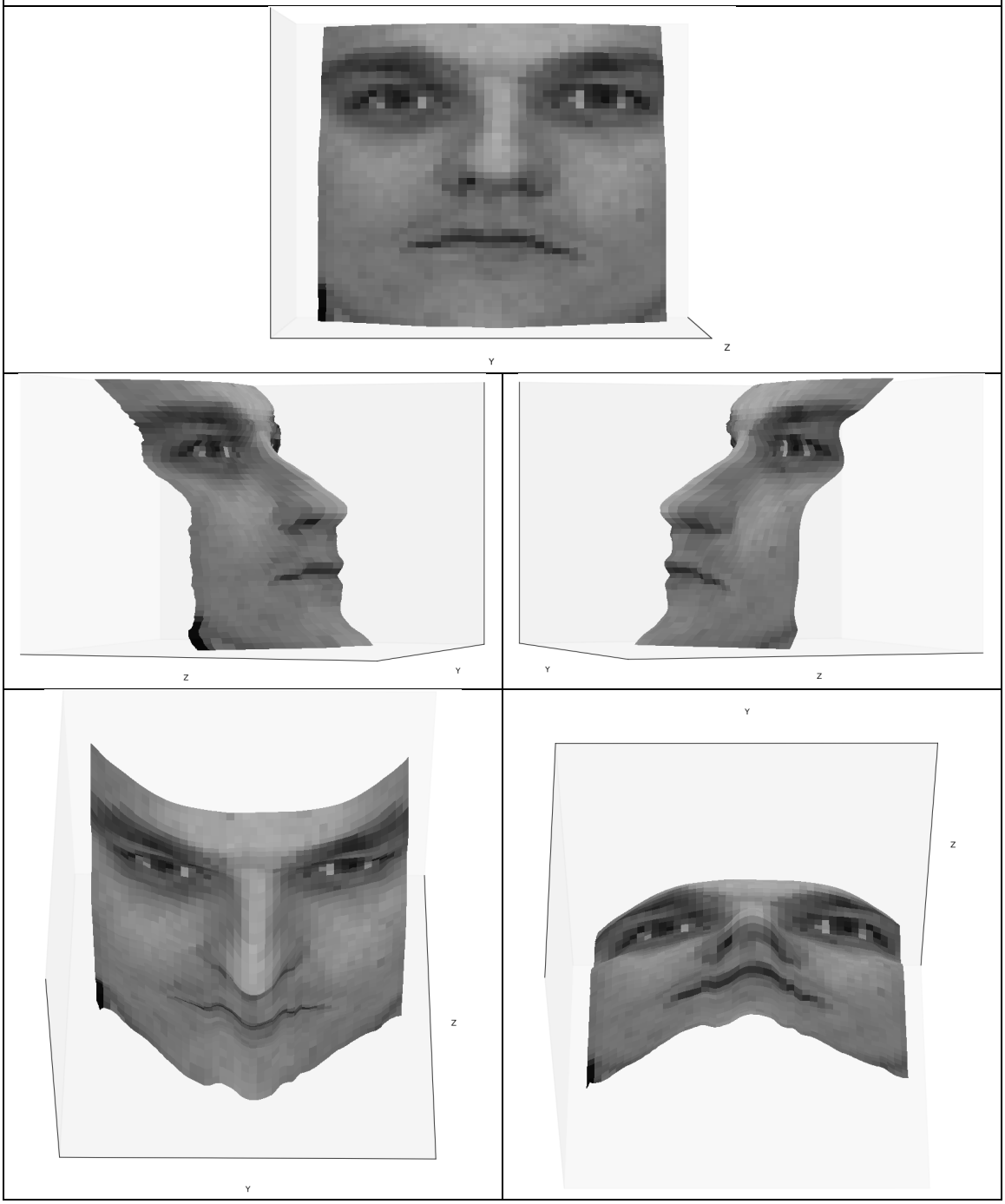
yaleB05 - random



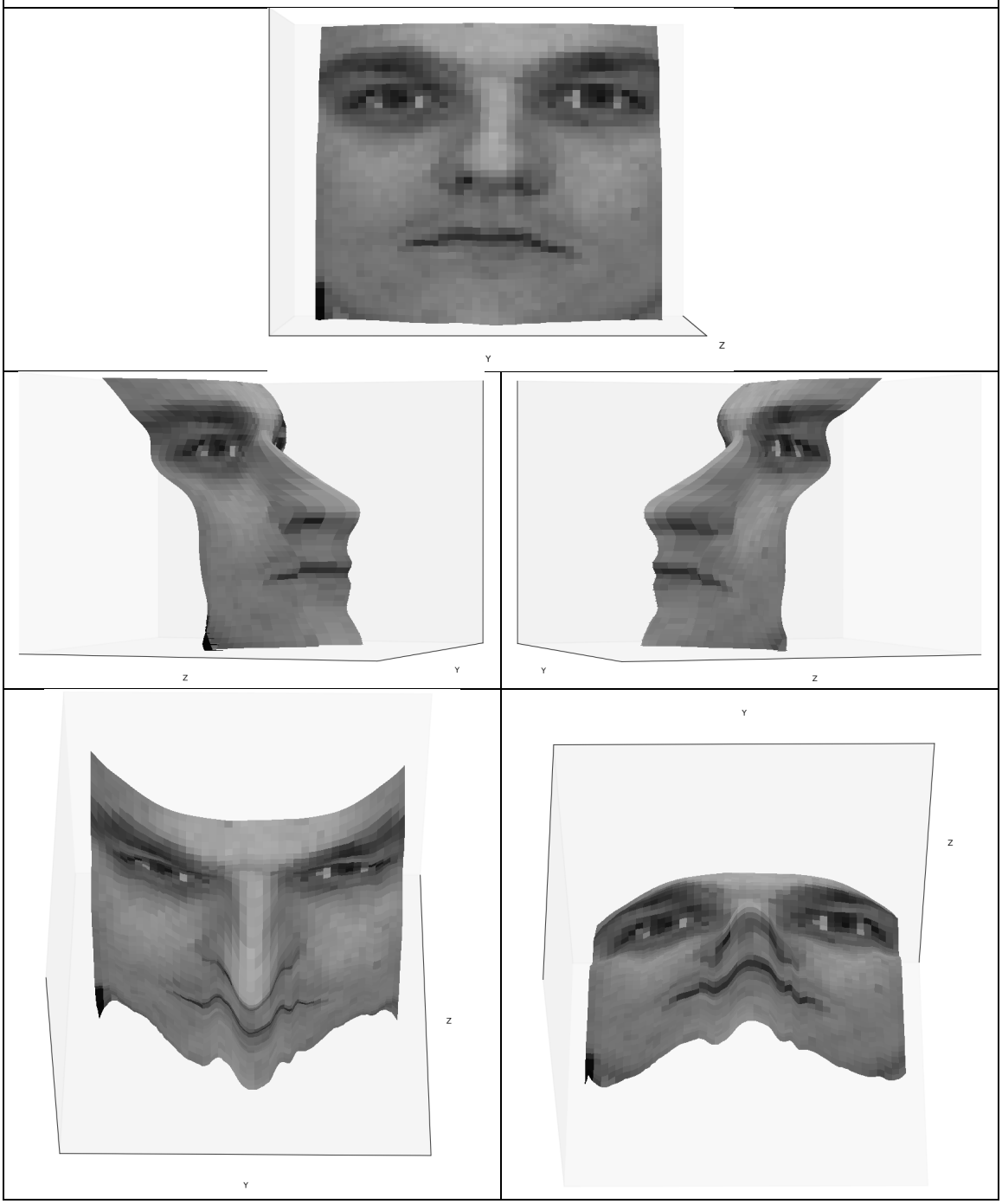
yaleB07 - random



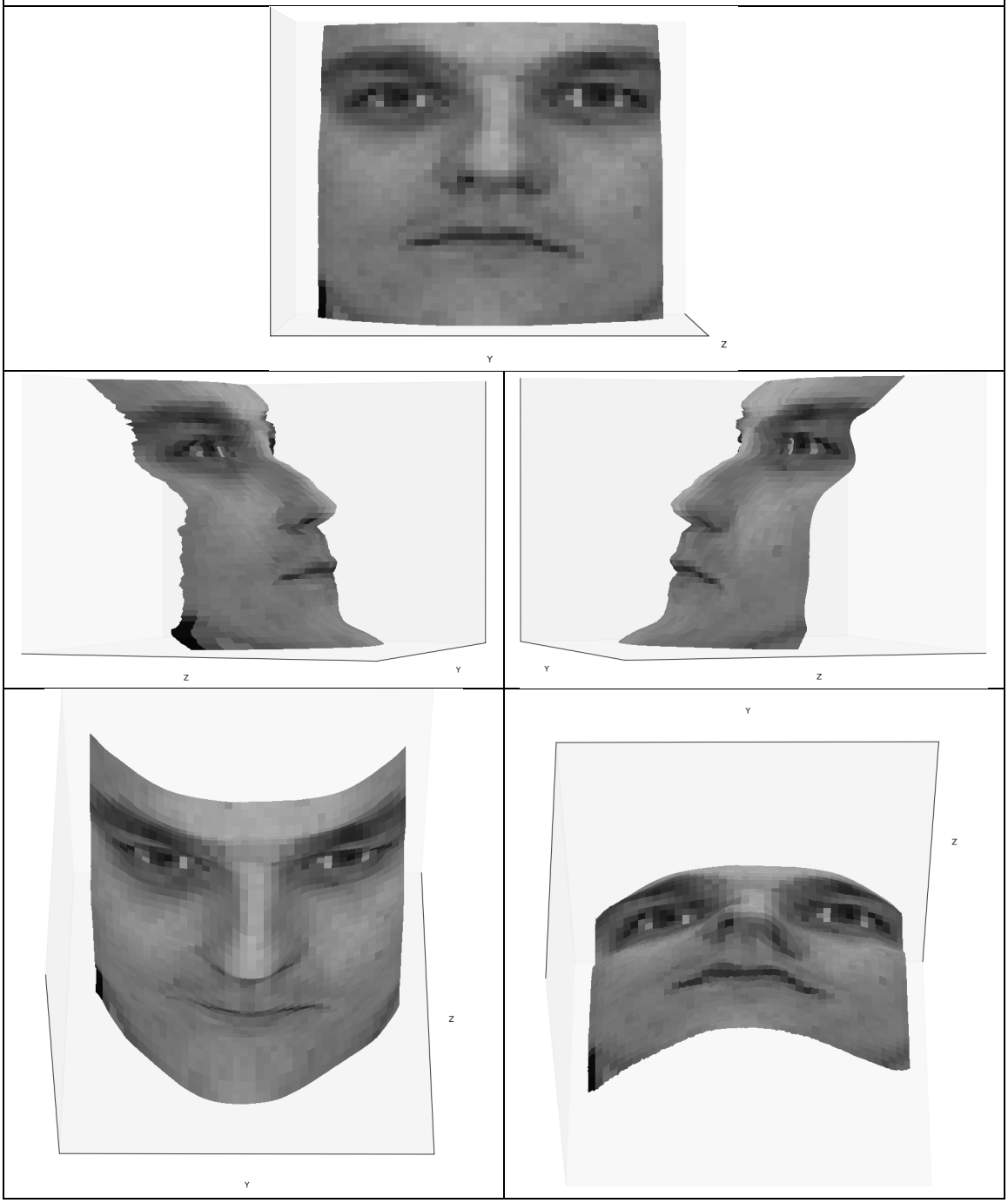
yaleB07 - average



yaleB07 - row



yaleB07 - column



6) Which integration method produces the best result and why?
   For all the subjects above, random method produces the best result because the results generated with random integration method do not have too many flaws on the person's eyes, nose, and mouth. The reason why random method produces the best results is probably because it generates several random paths (35 in total) to destination and average their cumulative sum. This ensures a better and more comprehensive calculations of the surface height since it explores different paths. Compared to row method, it only includes the height information in the first row, similarly, the column method only includes the height information in the first column. Therefore, the average method which averages the row method and column method is still worse than the random method.

7) Compare the average execution time (only on your selected subject, "average" here means you should repeat the execution for several times to reduce random error) with each integration method, and analyze the cause of what you've observed:

yaleB07	
Integration Method	Execution Time (s)
random	88.79478096961975
average	0.0009612358093261719
row	0.0009146415710449219
column	0.0009180835723876953

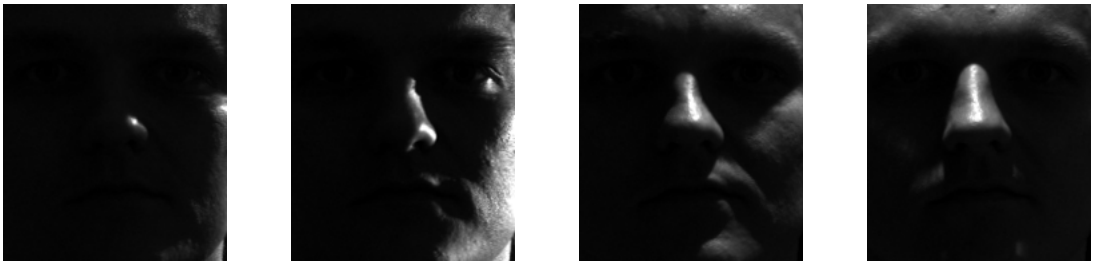
   The row method and column method have roughly the same runtime. The average method has slightly longer runtime than row or column method probably because it has an extra step which averages the results acquired from row and column methods. The random method has the longest runtime, approximately 1e5 times slower than row, column, average methods since it has nested for loop, and it explores random path 35 times to each destination.

C: Violation of the assumptions

8) Discuss how the Yale Face data violate the assumptions of the shape-from-shading method covered in the slides.

1. The input images are not Lambertian surface which appears uniformly bright from all directions of view and reflects the entire incident light. The Lambertian model is a perfect diffuse surface that scatters incident illumination equally in all directions.
2. The input images are not a local shading model where each point on a surface receives light only from sources visible at that point.
3. The set of input images of a subject may not be obtained in exactly the same camera/object configuration. The person may have some slightly changes of facial expressions which may further cause misalignment problems.
4. Some images exhibit excessive shadows, particularly affecting certain areas of the face, which results in these sections being overly dark. This shadowing can introduce noise that may potentially affect the accuracy of computations involving albedo, surface normals, and height maps. Moreover, the presence of a moustache causes some inconsistencies in the skin's smoothness.

Images from yaleB07



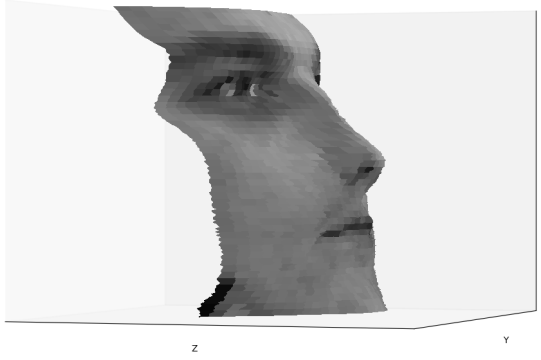
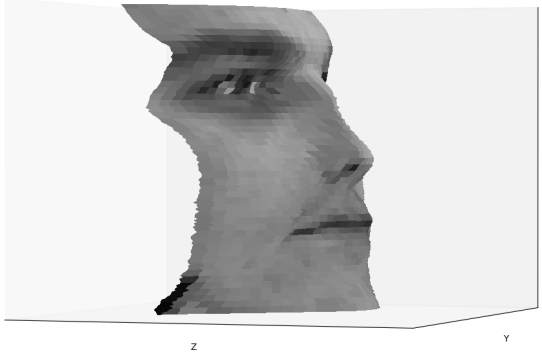
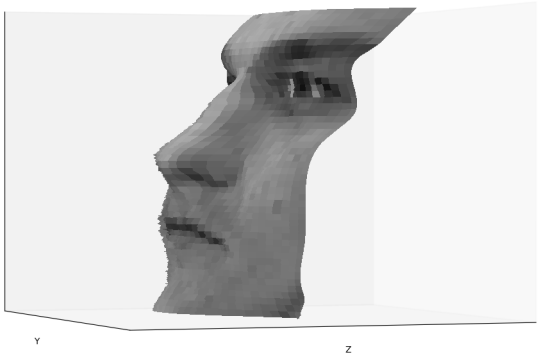
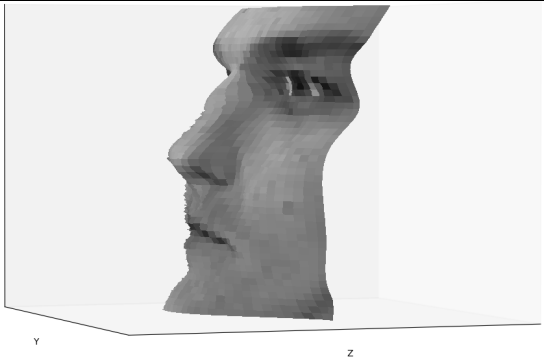
9) Choose one subject and attempt to select a subset of all viewpoints that better match the assumptions of the method. Show your results for that subset.

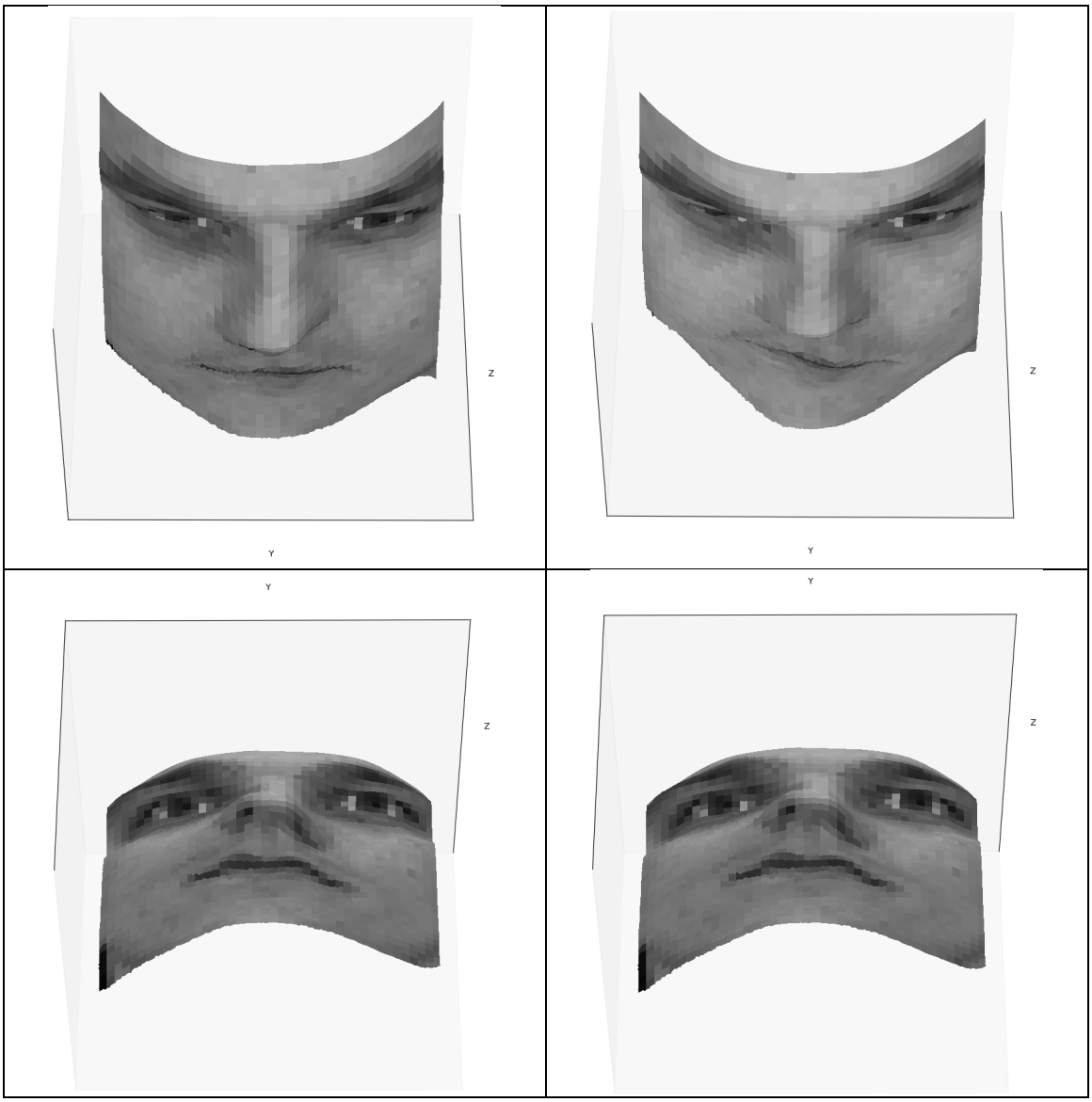
```
im_list_filtered = []
for fname in im_list:
  mat = load_image(fname=fname)
  num_dark_pixels = numpy.where(mat <= config.pixel_val_thres)[0]
  num_dark_pixels = num_dark_pixels.shape[0]
  dark_pixel_ratio = num_dark_pixels / mat.size

  if dark_pixel_ratio <= config.dark_pixel_ratio_thres:
      im_list_filtered.append(fname)
```

The above code basically filters out all the images that are way too dark to capture any information in it. Specifically, images having 75% of pixels that have the value less than 50 are filtered out because it’s very difficult to capture features from those images.

10) Discuss whether you were able to get any improvement over a reconstruction computed from all the viewpoints.

yaleB07	
Original Input	Filtered Input
	
	
	



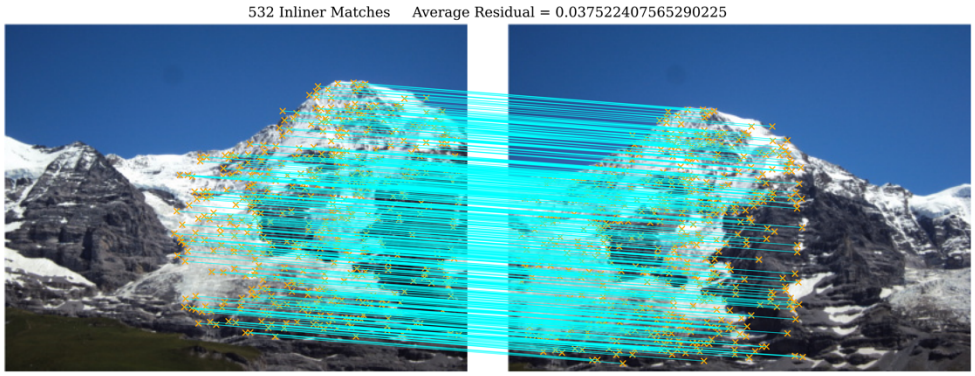
As the results shown above, the height map generated by the filtered input images is clearer, sharper, and brighter. Specifically, the shapes of the nose, mouth, and chin are more realistic.

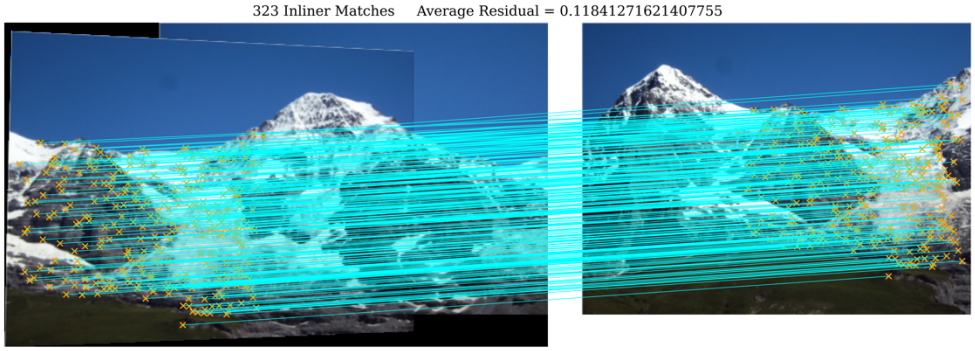
Part 3: Extra Credit

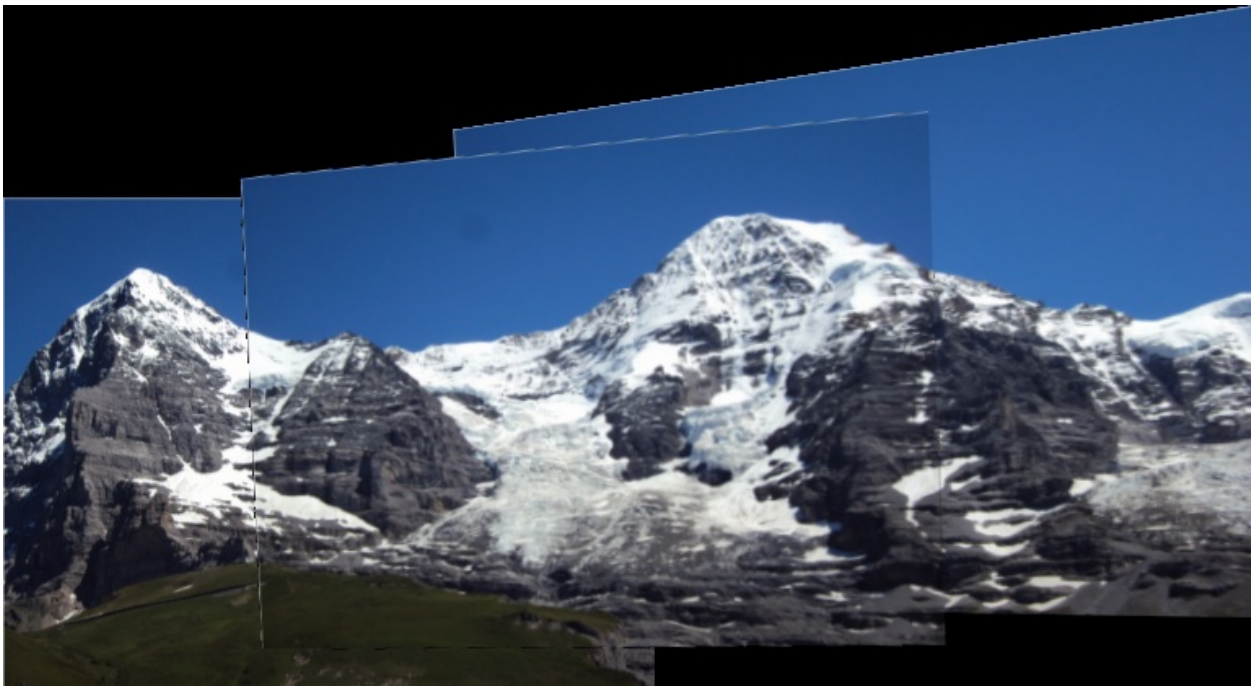
Post any extra credit for parts 1 or 2 here. Don't forget to include references, an explanation, and outputs to receive credit. Refer to the assignment for suggested outputs.

Part 1: Stitching Multiple Images

1. hill

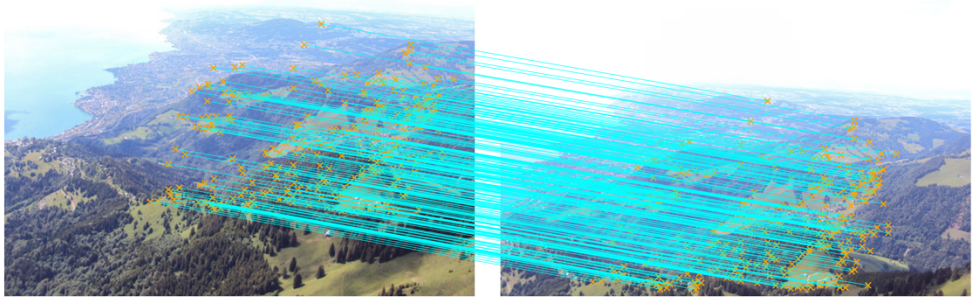




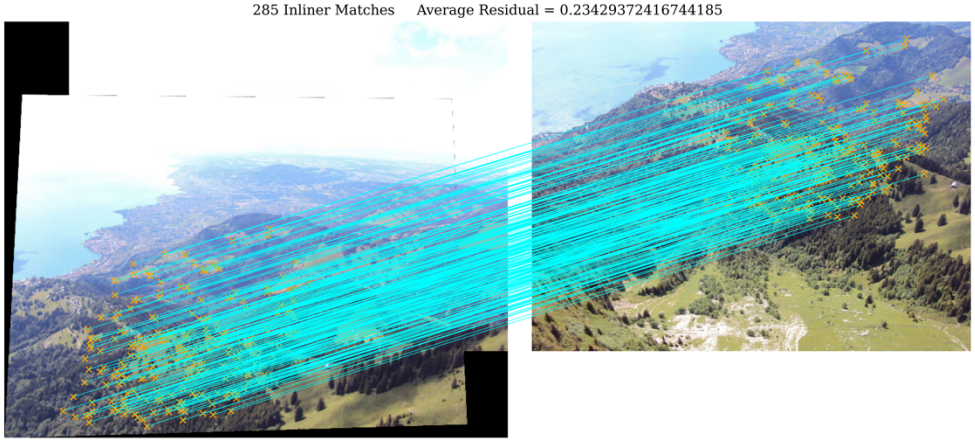


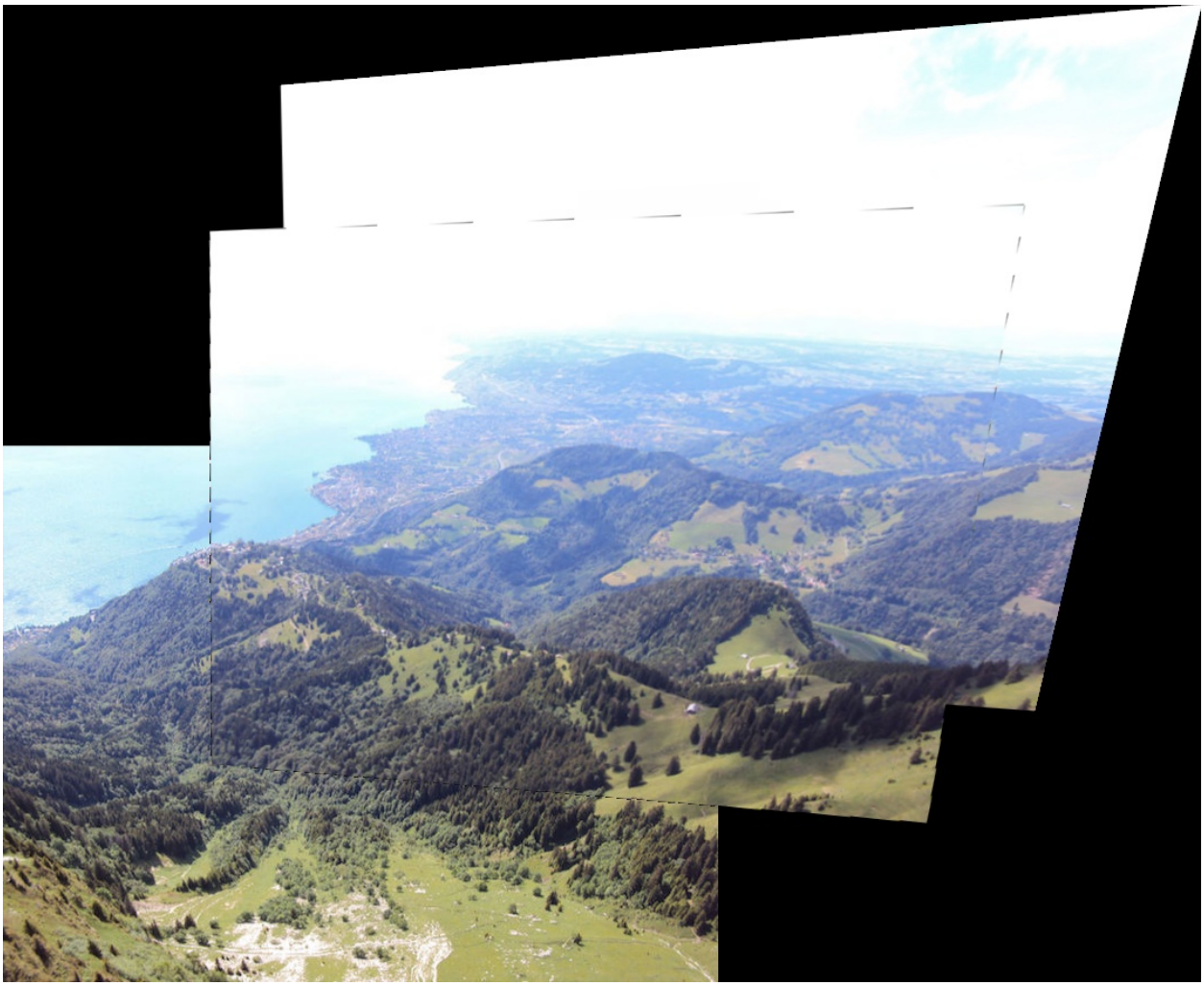
2. ledge

300 Inliner Matches Average Residual = 0.22629246215877244



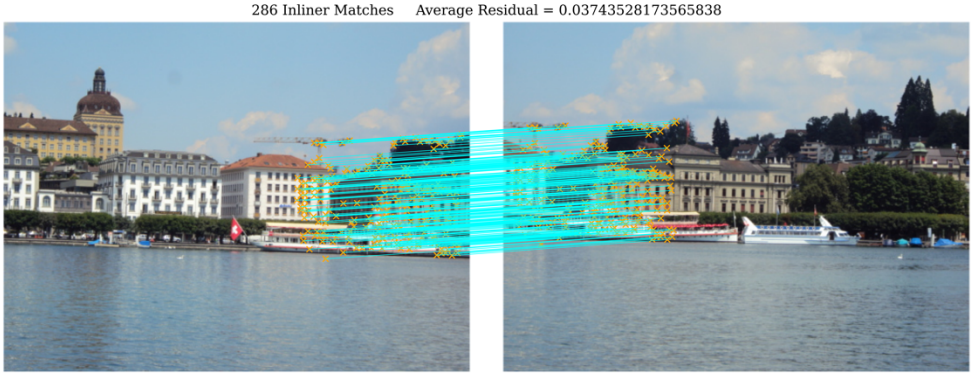
285 Inliner Matches Average Residual = 0.23429372416744185



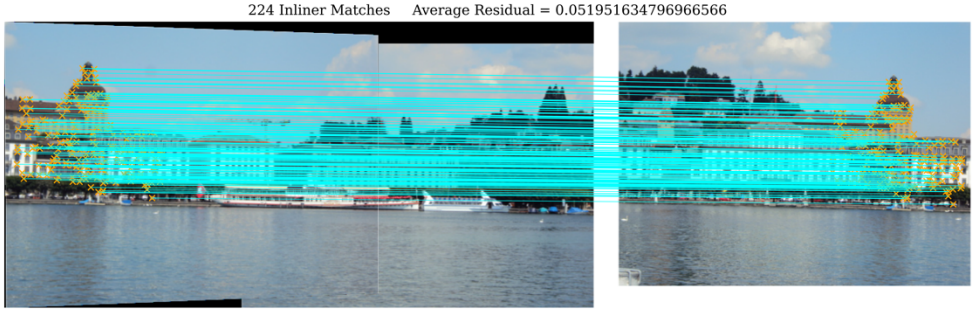


3. pier

286 Inliner Matches Average Residual = 0.03743528173565838



224 Inliner Matches Average Residual = 0.051951634796966566



Reference

https://scikit-image.org/docs/stable/auto_examples/registration/plot_stitching.html#sphx-glr-auto-examples-registration-plot-stitching-py

https://www.sciencedirect.com/topics/computer-science/lambertian-model

https://www.azooptics.com/Article.aspx?ArticleID=790