Assignment 4

ECE 736: 3D Image Processing and Computer Vision

Pranav Rawal

Department of Electrical & Computer Engineering, McMaster University

Question 1: Image Resizing using Seam Carving

To run the python script for the four experiments in this question, run the **Q1_python/main.py** in terminal. Follow the prompts to run experiments 1 through 4, the experiments take a fair bit of time to run due to the fact that I am doing both median filtering and bilateral filtering prior to calculating the image gradients for the energy function. The preprocessing improves the seam selection by denoising the image slightly. Outputs will be saved in the Q1_pyhton folder upon completion, the terminal will inform the user once output images are saved.

Experiment 1:

Below is the original union-terrace.jpg image that the seam carving algorithm will be run on.

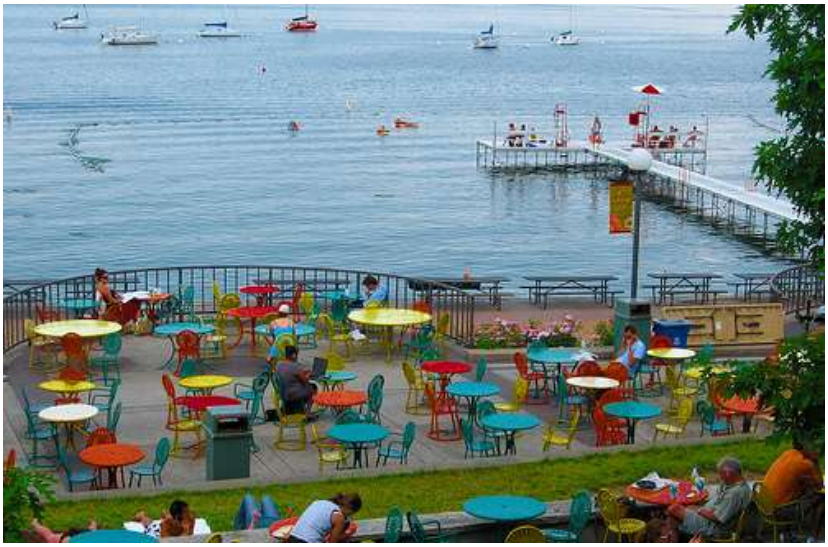

Figure 1: Union-terrace.jpg (original)

On the following page the results from removing 100 columns, 100 rows, and finally both 100 rows and column removed. The implementation does a relatively good job of retaining the most interesting parts of the image (patio area, dock, etc.)

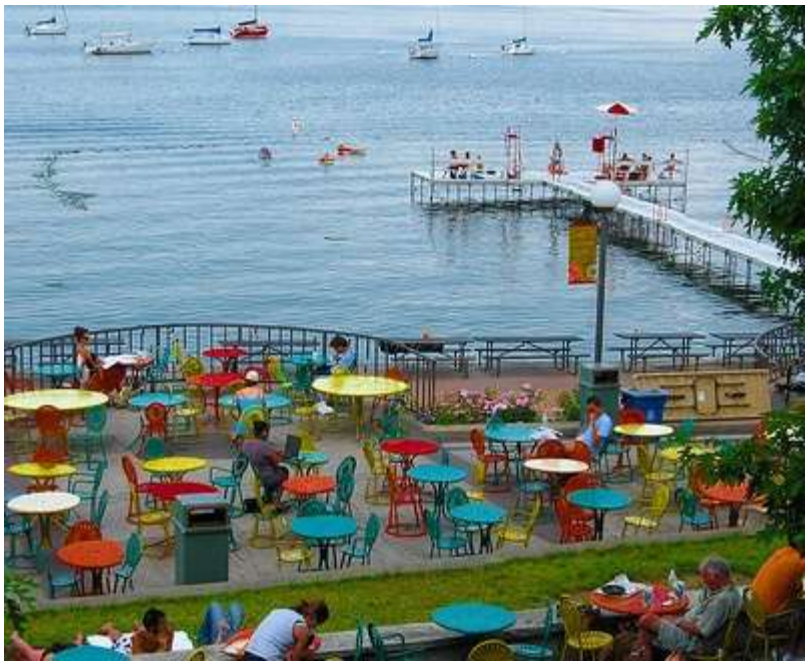

Figure 2: E1a.jpg (100 columns removed)

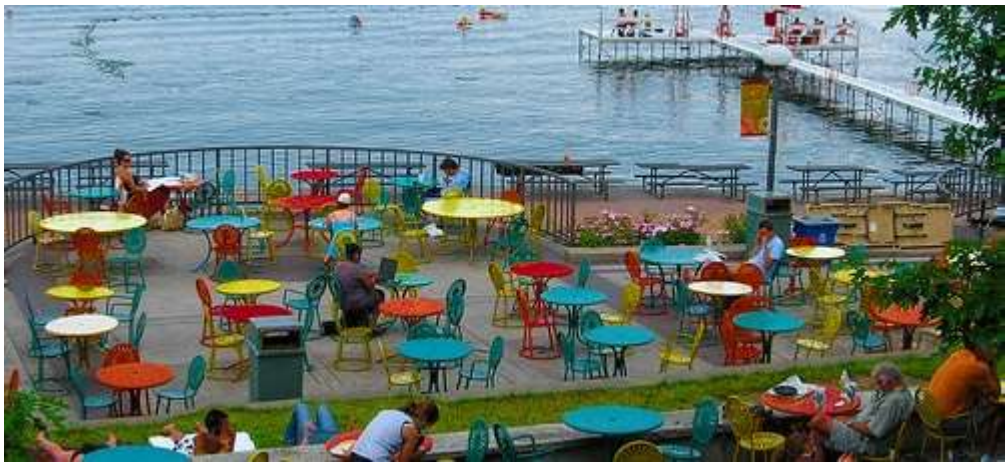

Figure 3: E1b.jpg (100 rows removed)

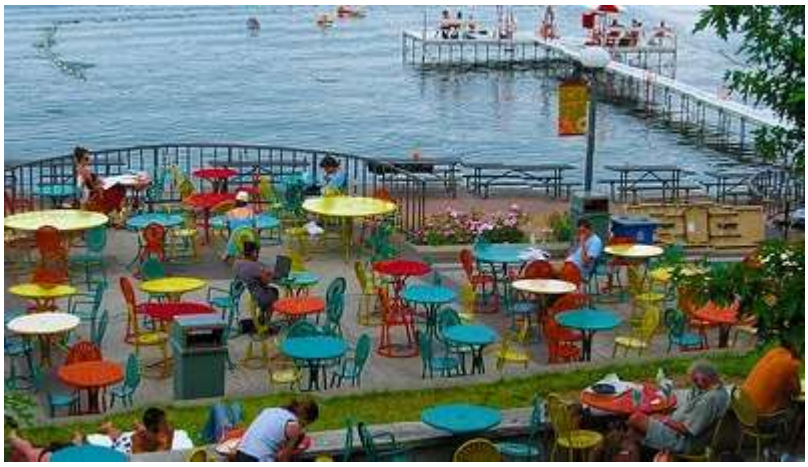

Figure 4: E1c.jpg (100 rows and columns removed)

Experiment 2:

Below are the first horizontal and vertical seam selected by the algorithm. These are reasonable choices since the path of the selected seams are relatively flat in terms of texture and color. For example, the first horizontal seam cuts across the top of the image avoiding the docks and the boats in the distance. And as for the vertical seam its passing through the left side of the image where if compared to the rest of the image there really isn't much going on.

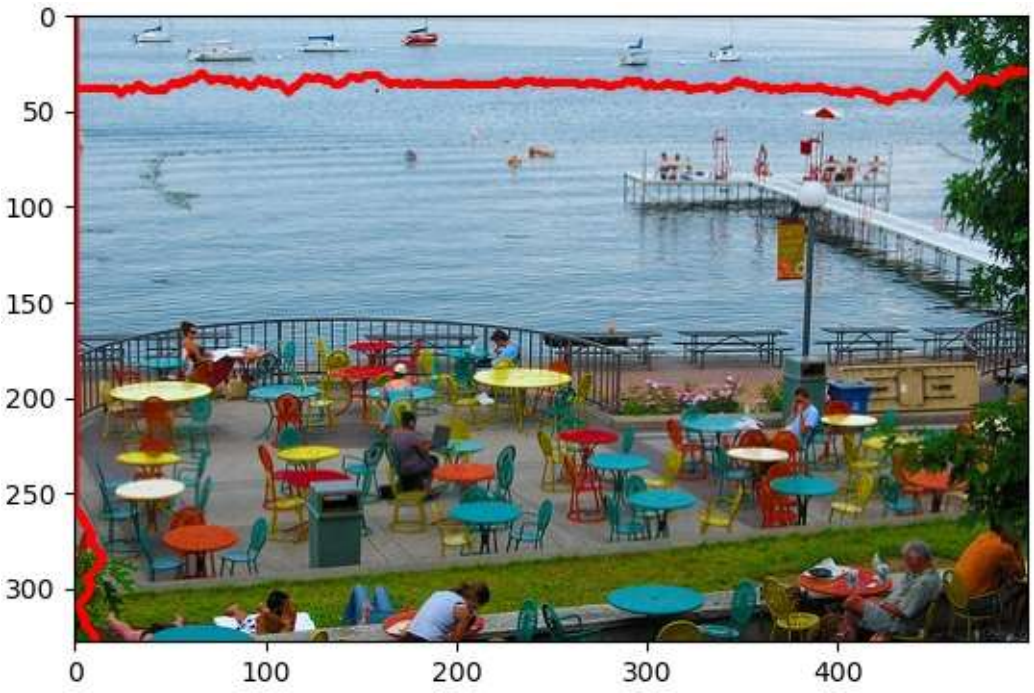

Figure 5: E2.jpg (first horizontal and vertial seams)

Experiment 3:

The input image for the interesting successful result I chose daft_punk.jpg, which can be found in the same directory as **main.py**, however, there were some changes I made to the original image. Firstly, I added a red circle near the top left corner of the image, and secondly, I flipped the image horizontally.

The addition of the red circle was intended to create a bit of a challenge for the algorithm, since the original image has a relatively flat surface at the top in terms of texture and color. I would consider the E3.jpg a successful result due to the fact that the red circle and the two Robots from Daft Punk are is retained in the output image after the removal of 60 rows and 150 columns from the input image in, shown in Figure 7.

The reason I flipped the image horizontally was because the algorithm fails and completely removes the red circle when doing the seam carving. I included the failures in the section further below, under Experiment 4 (interesting poor result).

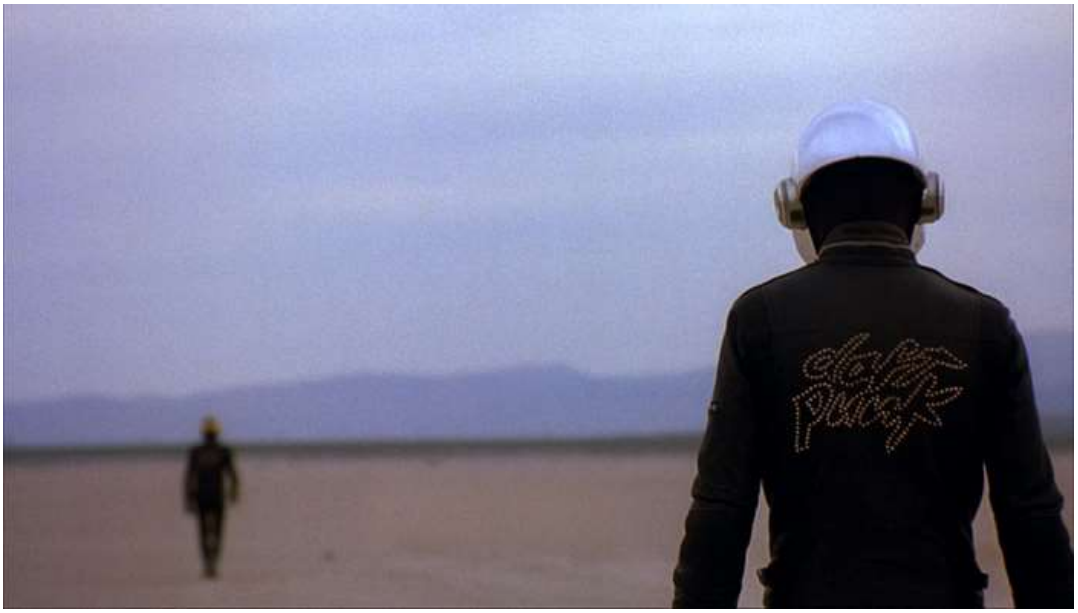

Figure 6:daft_punk.jpg (original image)

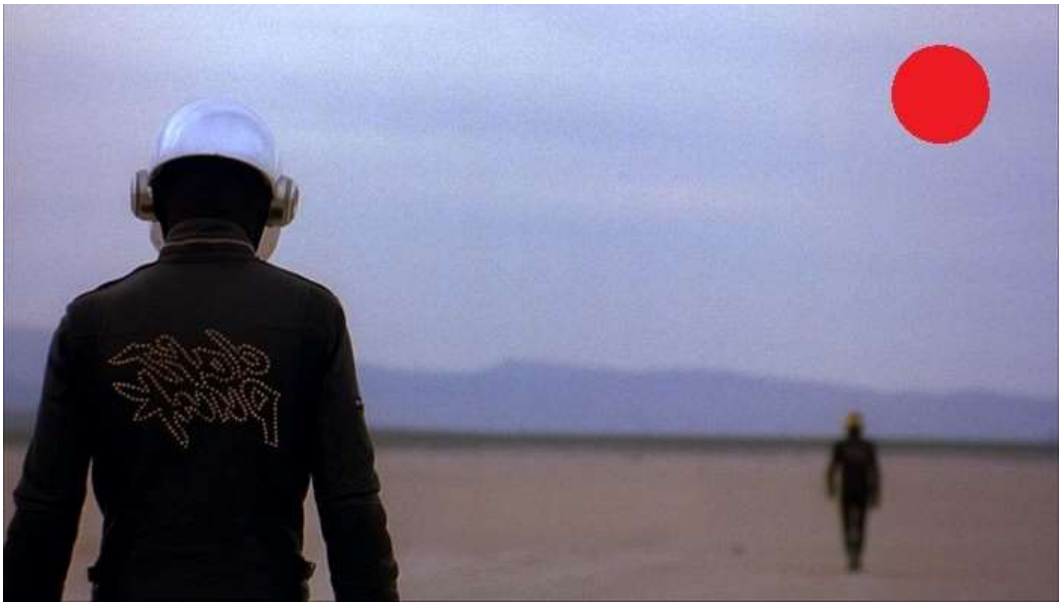

Figure 7:daft_punk1.jpg (input image for experiment 3)

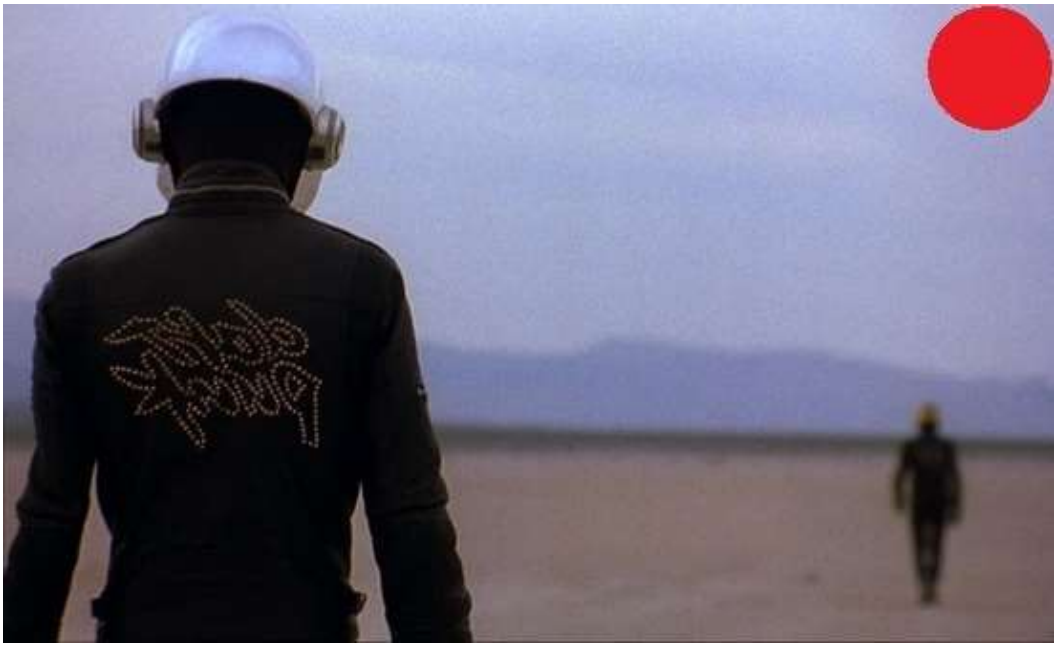

Figure 8:E3.jpg (output image from experiment 3)

Experiment 4:

As discussed above, the original image from Figure 6 needs to be flipped horizontally in order to get the successful result in Figure 8. However, if not flipped horizontally and running main.py we get the result in Figure 10, from the input image in Figure 9.

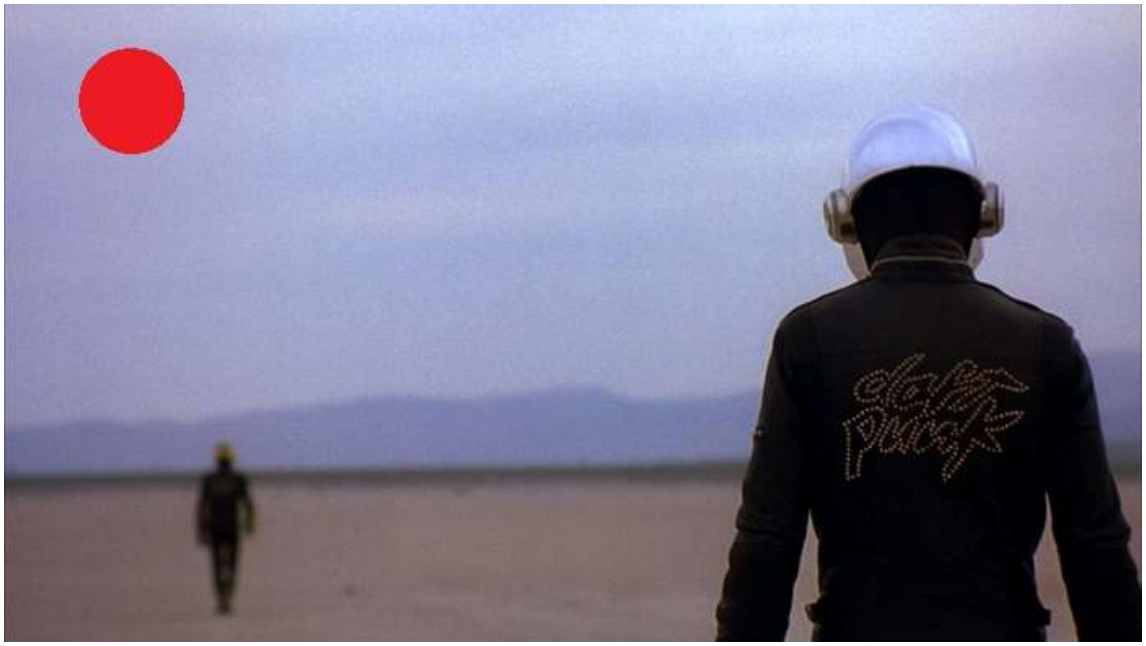

Figure 10: daft_punk.jpg (input image for experiment 4)

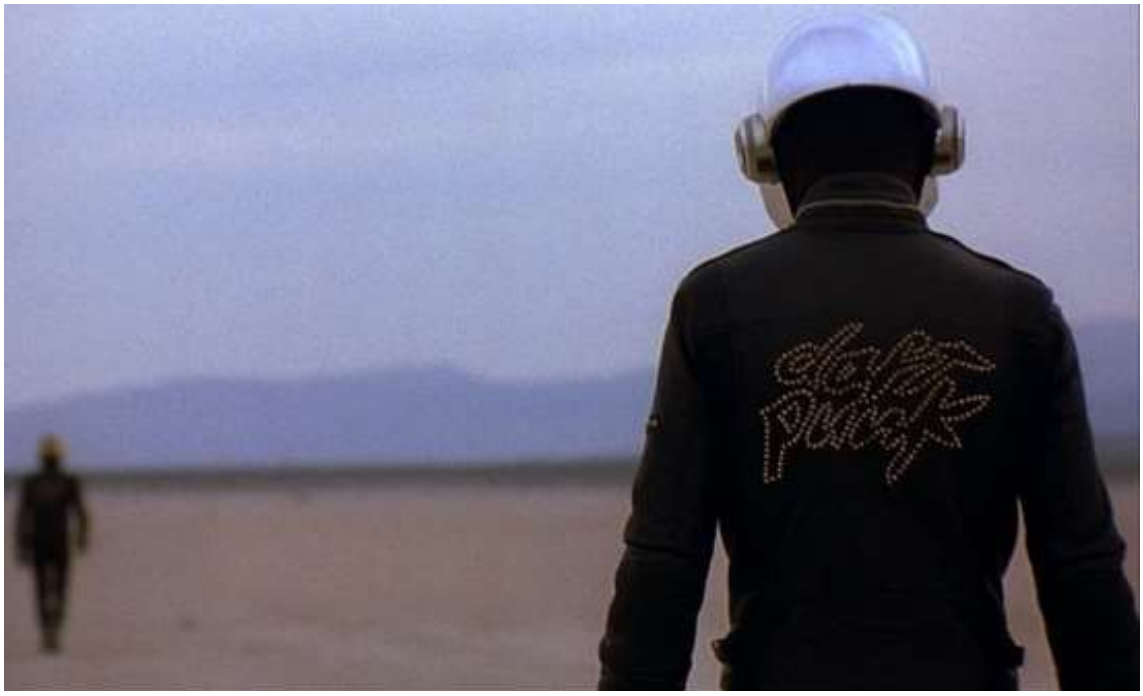

Figure 9: E4.jpg (output image for experiment 4)

Question 2: Bilateral Filtering

The code for Bilateral filtering can be found in **Q2_python/Bilateral_Filt.py** and it is an adaptation of the implementation by the github user: **anlcnydn**. The link to their implementation can be found here: https://github.com/anlcnydn/bilateral/blob/master/bilateral_filter.py.

I modified the code to be compatible for colored images and I also removed the need to input the widow width and instead determining the extent of the width to be two standard deviations from the mean and odd numbered.

I am computing 4 outputs based on the following combination of sigmaS and sigmaI, respectively: (2.5, 5), (2.5, 10), (5, 5), and (5, 10). The outputs can be found in the same directory as **Bilateral_Filt.py.** For comparison, I am also computing the bilateral filter implementation by OpenCV for the same sigmaS and sigmaI values. Figures 11 – 14 are output images from my implementation and Figures 15 – 18 are output images from OpenCV's bilateralFilter function.

Figure 11: bilateraltask0_sigmaS_2.5_sigmaI_5.jpg

Figure 12: bilateraltask1_sigmaS_2.5_sigmaI_10.jpg

Figure 13: bilateraltask2_sigmaS_5_sigmaI_5.jpg

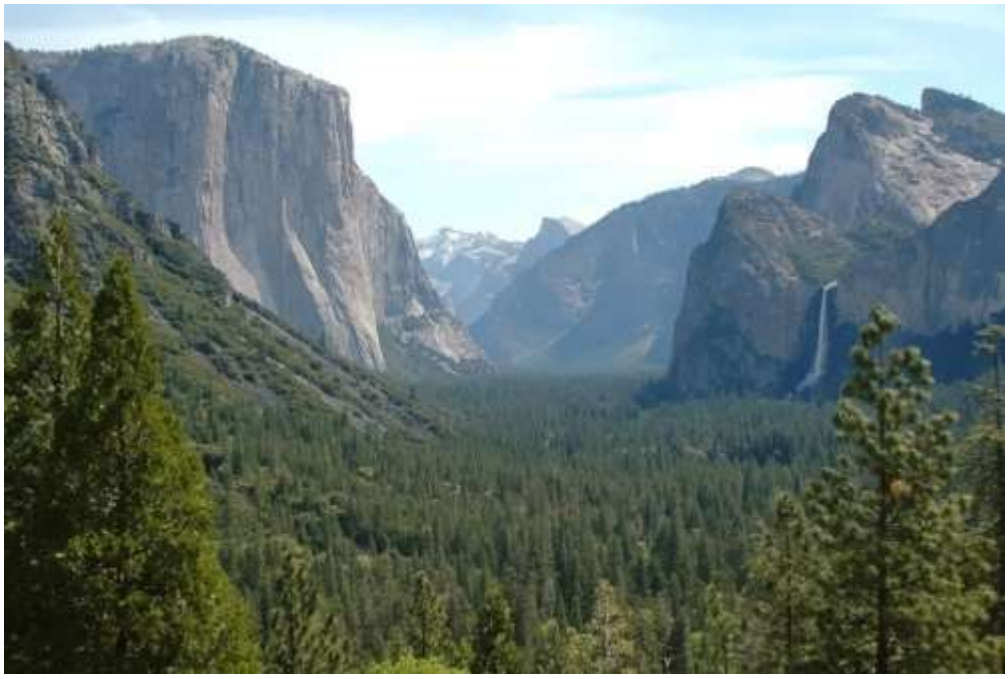

Figure 14: bilateraltask3_sigmaS_5_sigmaI_10.jpg

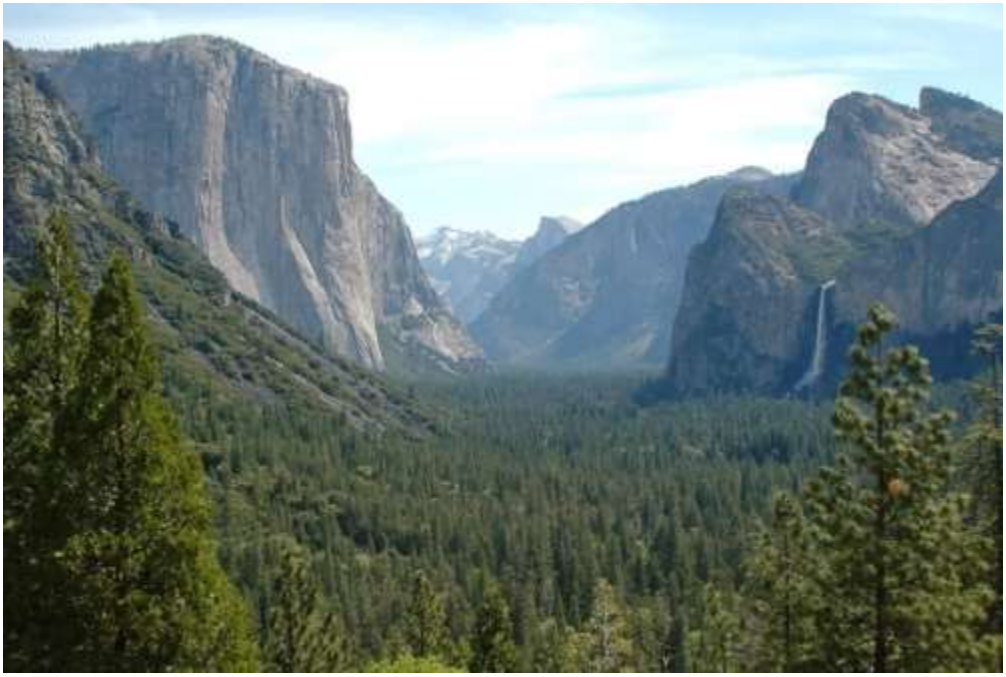

Figure 15:OpenCV_0_sigmaS_2.5_sigmaI_5.jpg

Figure 16: OpenCV_1_sigmaS_2.5_sigmaI_10.jpg

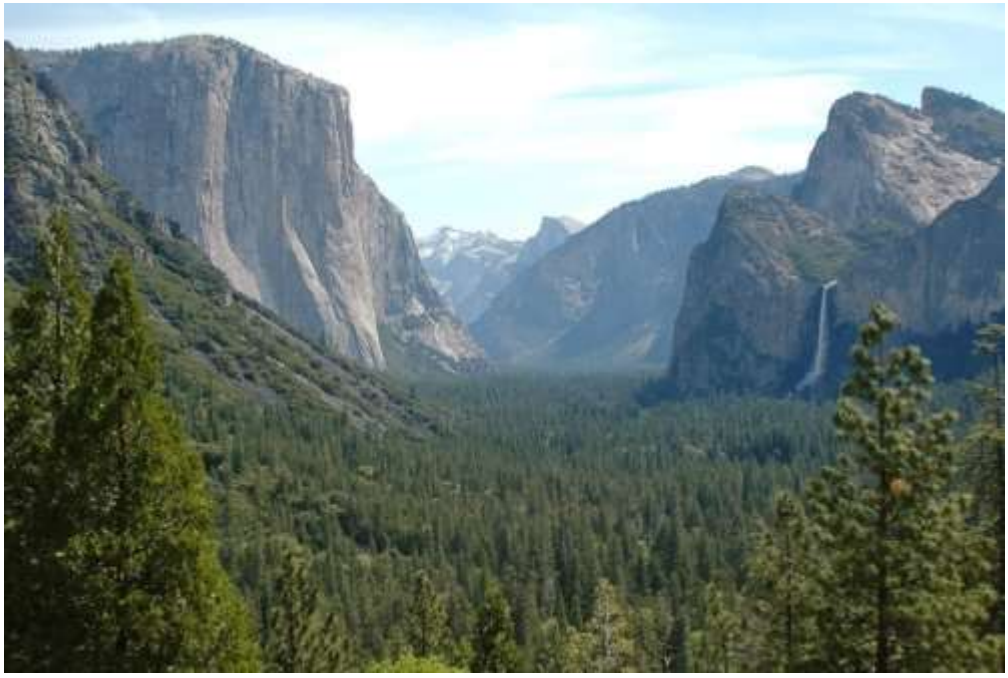

Figure 17: OpenCV_2_sigmaS_5_sigmaI_5.jpg

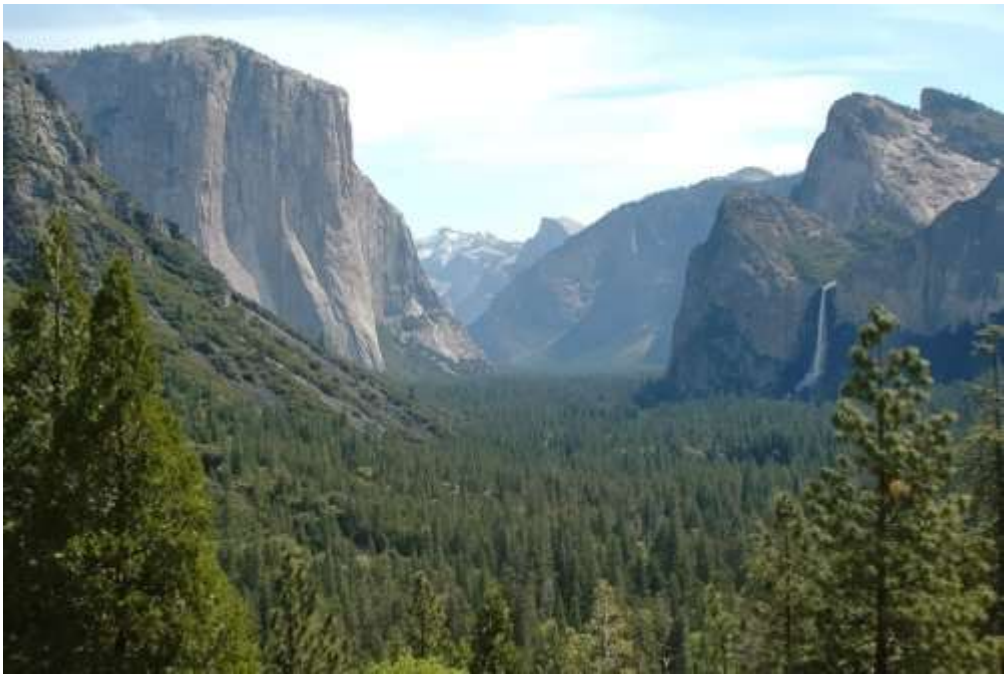

Figure 18: OpenCV_3_sigmaS_5_sigmaI_10.jpg