Rajat Bhageria, Rajiv Patel-O'Connor, Kashish Gupta (Group 17)
CIS 581
16 Saturday 2017
Final Project

Face Swapping Across Videos

1. Introduction / Abstract:

Though it seems trivial and an easy "Photoshop job," making face swapping scalable for videos using computer vision has a long pipeline of steps. In this paper, we demonstrate a faceReplacement methodology for two images and then abstract it be able to swap the faces in two videos. In particular our face replacement method finds the facial landmarks in a face, creates a convex hull of the face from those landmarks, find the Delaunay triangulations based on the hull, does an affine transformation from a triangle in one image to the corresponding triangle in the next image, does seamless blending, and then returns the blended image. Through this method, we were able to successfully swap faces between two single-faced videos of unequal length.

2. Methodology:

1) Write a faceReplacement function (img1, img2):
    a) Read in the two images and make a copy of the second one as an output image; call it img2Warped
    b) Find a nx2 array of the xy positions of the facialLandmarks in both of the images
        i) Using dlib, use the frontal face detector with the *shape_predictor_68_face_landmarks* predictor
        ii) Convert the images to grayscale
        iii) Improve contrast of grayscale image using CLAHE
        iv) Run the detector on the grayscale images (post-CLAHE) and return the xy positions of the facial landmarks
    c) Using OpenCV's convexHull function, pass in the nx2 matrix of facialLandmarks of the second image and get indicies for the convexHull on the second image
        i) The idea here is that all the features allow us to find the positions of the outline of the face. This is useful when we calculate the Delaunay triangulations.
    d) Calculate the hull positions of img1 and img2 (the indices part of the entire land mark function).
    e) Get the Delaunay triangulation for the hull2 simplicies using the Scipy.spatial package
    f) For each of the k triangles formed
        i) Get the xy positions of the corners of the triangle in img1 and img2
        ii) Warp triangle1 from img1 to img2Warped to trinagle2
            (1) Create a bounding rectangle around both of the triangles

            (2) Create a mask of the triangle and use fillConvexPoly to make the mask only the dimensions of the second rectangle for the second triangle.
            (3) Calculate an affine transform from the first rectangle to the second
            (4) Apply the affine transform on the first rectangle to get the second rectangle
            (5) Copy the new rectangle to the img2Warped
    g) Do blending
        i) Get the mask of only the hull and use fillConvexPoly to fill it
        ii) Get the center of the face
        iii) Do blending with seamlessClone on cv2
    h) Return the final image

2) Write a faceReplacementVideo function for two videos that calls faceReplacement
    a) We first extract all of the frames into a 4 dimensional numpy array (shape = (frames, h, w, 3))
    b) Create two empty 4-d NumPy arrays that will be populated with frames in which the face is swapped
    c) Build "swapped" videos one at a time, frame by frame
        i) We use corresponding frames between videos
            (1) If the video of interest is longer than the other video (one from which we are taking the face), we use the last frame of the other ideo to populate the remaining frames

3. Results:

We can successfully swap faces between two single-faced videos of unequal length. We use seamless poisson cloning to ensure to make face swap appear more natural

We have engineered a system that given two single-faced input videos, can generate two corresponding output videos in which the faces are swapped. Our system successfully handles input videos of different lengths, and utilizes gradient domain blending to make the face swap appear natural. These results can be visualized in the appended files.

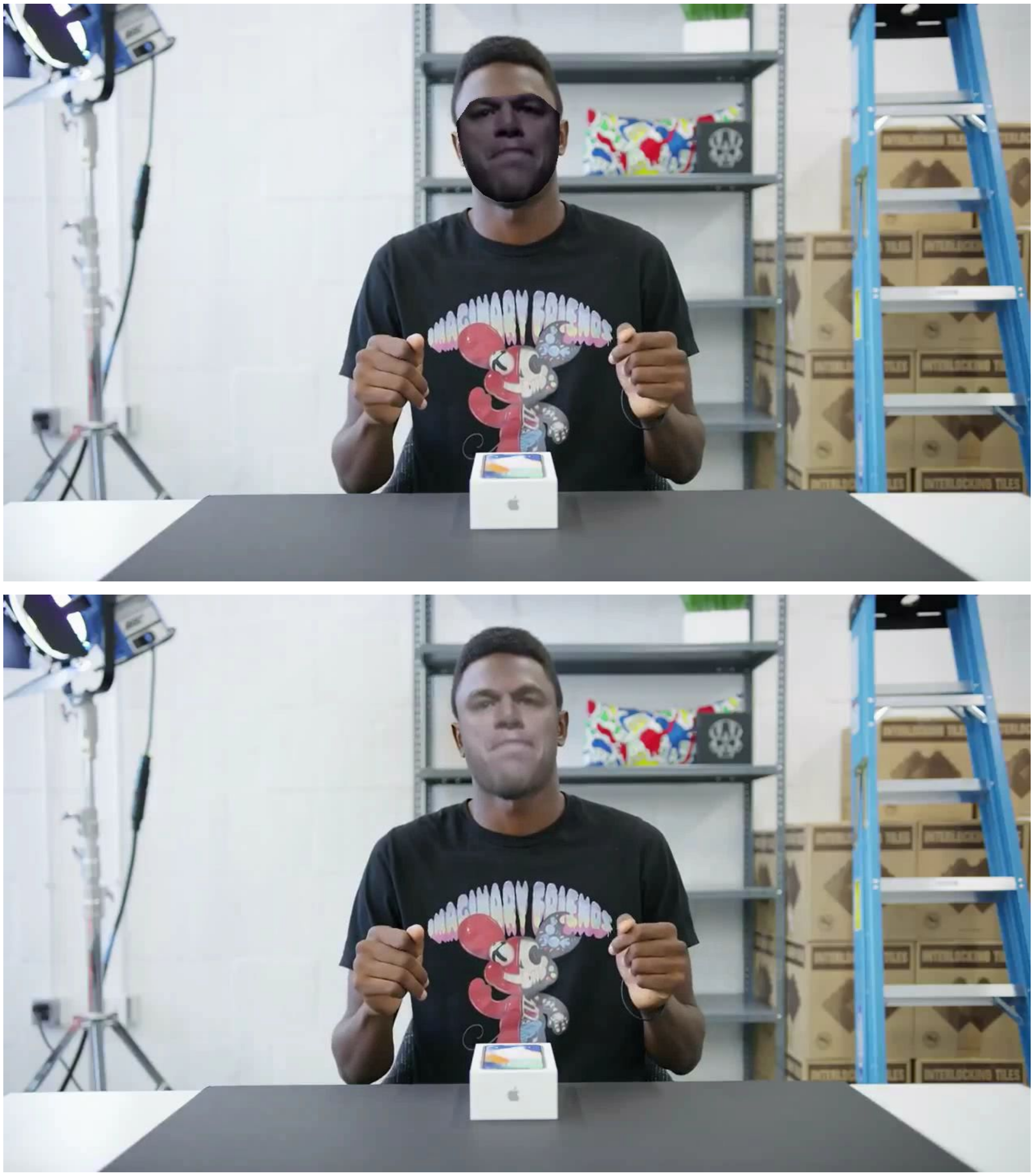

Figure 1: *Before Blending (top) and After Blending (bottom)*

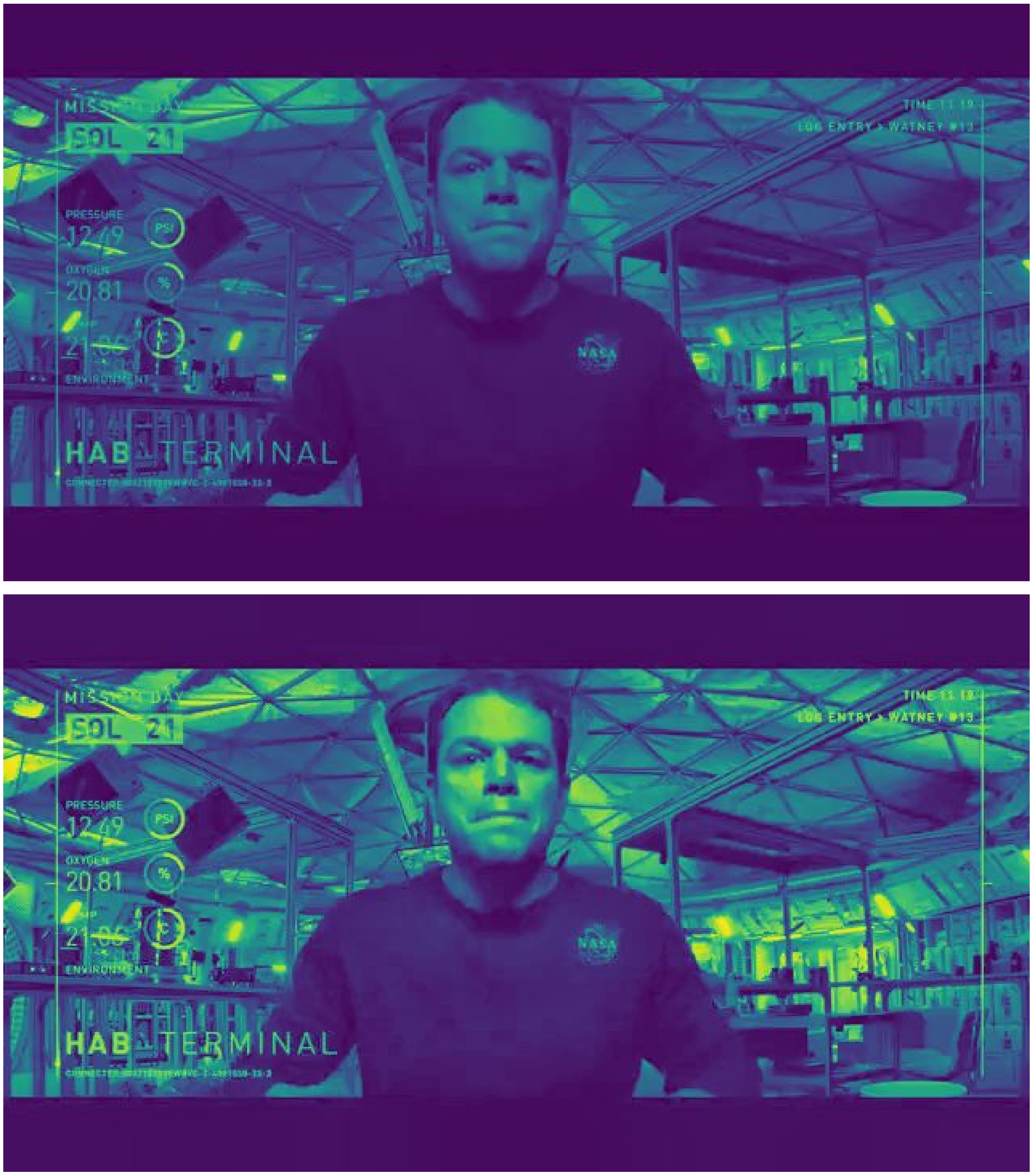

Figure 2: *Before CLAHE (top) and After CLAHE (bottom). This is in the context of the input which will be used for facial extraction.*

4. Analysis / Future Work:

Our system is mostly vectorized and optimized from a memory standpoint such that it runs in a reasonably quickly without absurd memory consumption.

While our system is robust when it comes to simple single-face swapping, it is limited in its true application potential. The system is reliant on finding facial landmarks from a fully visible frontal face, and will break if facial landmarks are not detected in the first frame. Additionally, to account for short periods (on the order of a few frames) in which our system cannot detect facial landmarks, we utilize the facial landmarks from the previous frame. Our system will perform poorly for videos in which a full frontal face is not visible for extended periods of time. This could be overcome by detecting features less frequently and tracking them using optical flow.

Another limitation of our system is that it has been engineered to only consider single-face swapping between two videos. Future work could be done to handle multiple (> 2) video inputs, videos that have more than one face each, and videos that have unequal numbers of faces. Handling multiple faces would require some extra looping and adding additional dimensions to several NumPy arrays, namely in our *facialLandmarks()* function and in our *faceReplacement()* function. To handle videos that have unequal numbers of faces, we propose the following:

Let us define the video with more faces as video1 and the video with fewer faces as video2.

1. For the "swapped" version of video2, randomly select a set of facial landmarks from video1 such that the set size is the number of faces in video2.
2. For the "swapped" version of video1, loop through the set of facial landmarks from video2 until each face to be replaced in video1 has a corresponding set of facial landmarks.

Lastly, this system could be further engineered to increase practicality by giving the user "Snapchat-like" filters. We propose simply ones such full-frame color filters (sepia, black and white, etc.) and video speed filters (2x, .5x).

5. References:

5.1. Libraries Used:

- Dlib for extraction of facial features
- OpenCV for general image processing functions (RGB to grayscale, BGR to RGB, etc.), CLAHE, finding the convex hull, warping, and blending
- Scipy.spatial for Delaunay triangulation
- Skvideo.io for generating MP4 from Numpy ndarray

5.2 Tutorials Used:

Convex Hull — OpenCV 2.4.13.4 documentation. (2017). *Docs.opencv.org*. Retrieved 19 December 2017, from https://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/hull/hull.html

Mallick, S. (2017). *Face Morph Using OpenCV — C++ / Python*. *Learnopencv.com*. Retrieved 19 December 2017, from https://www.learnopencv.com/face-morph-using-opencv-cpp-python/

Mallick, S. (2017). *Face Swap using OpenCV (C++ / Python)*. *Learnopencv.com*. Retrieved 19 December 2017, from https://www.learnopencv.com/face-swap-using-opencv-c-python/

Mallick, S. (2017). *Seamless Cloning*. *Learnopencv.com*. Retrieved 19 December 2017, from https://www.learnopencv.com/tag/seamless-cloning/