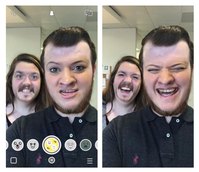
Face Swap: Automatic Face Detection, Alignment, and Compositing

Arnav Das, Varun Govind, Nikhil Mehta

Department of Computer Science, College of Engineering, University of Illinois at Urbana-Champaign

INTRODUCTION

Our final project is implementing a filter, popularized by social media apps such as Snapchat and Instagram, called Face Swap. This tool allows you to switch your face with a friend automatically and in real time, using a series of image processing methods. The result is a fun and interesting application of computational photography techniques.



Face Swap on Snapchat

In order to accomplish this goal, we first tested our process step by step on static images, and then moved onto developing the real-time version of the app.

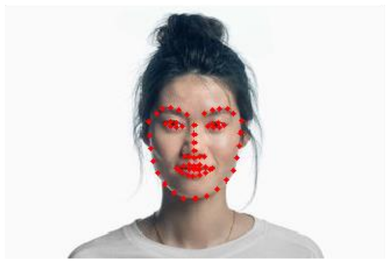
The steps of this process include:

- Face Detection
- Landmark Detection/Feature Mapping
- Computing Homographies to Warp Image
- Cutting around facial region
- Generating Mask around facial cutouts
- Compositing and Blending final Image

DETECTION

Detection and Feature Mapping

To detect faces in an image, we use the *dlib* library's API. The function we use utilizes a neural net to output a bounding box for the faces detected in an image. We use the same API to find the landmarks of the face, such as the eyes, nose, mouth, and jawline. The model we use outputs 68 key features of a face. The landmarks we use are shown in the image below.



WARPING

Features

The landmarks we use to compute the transform are the 68 points that we retrieve from the function shown in the previous section. Since the landmarks are consistent and relatively accurate, we can compute a homography directly, rather than use an algorithm like RANSAC.

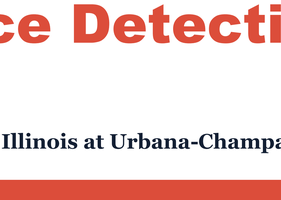
Computing Transform

To compute the transform matrix H, we use the following equations

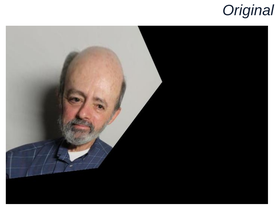
$$\begin{bmatrix} -u_1 & -v_1 & -1 & 0 & 0 & 0 & u_1u_1' & v_1u_1' & u_1' \\ 0 & 0 & 0 & -u_1 & -v_1 & -1 & u_1v_1' & v_1v_1' & v_1' \\ & & & & \vdots & & & & \\ 0 & 0 & 0 & -u_n & -v_n & -1 & u_nv_n' & v_nv_n' & v_n' \end{bmatrix} \mathbf{h} = \mathbf{0} \Rightarrow \mathbf{A}\mathbf{h} = \mathbf{0}$$

$$\mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

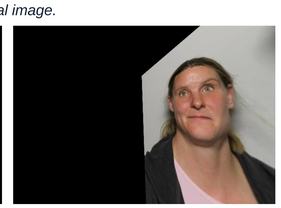
To swap faces, we compute two homographies: one that maps the face of person A to the face of person B, and one that creates the opposite mapping. An example of this algorithm at play is shown below:



Original image.

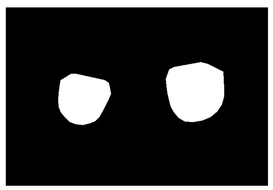


Computing a homography to map the right person to the left.



Computing a homography to map the left person to the right.

We then compute a mask to isolate the faces, by finding the convex hull of the warped points. The process of applying the mask is shown below:



The filled in convex hulls of the two detected faces

The warped images after being masked and added

Now we just need to find a method to composite masked and warped faces to the original image!

COMPOSITING

Once we have computed the homographies and appropriate transforms, we have to composite our images. To accomplish this, we attempted to use three different methods: basic copying and pasting, feathering, and gradient domain fusion.

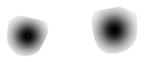
Basic

For a baseline, we show the results of the combined images where we apply no techniques to composite them.

This does surprisingly well, though there are clearly some visible seams.

Feathering

To apply feathering, we multiply the regions of interest of our foreground by K and the corresponding regions in the background by 1 – K, where K is a Gaussian kernel of the same size as an axis-aligned minimum bounding box around each set of landmarks. The kernel masks that we apply on the image from the previous section are shown below:



Background kernel mask

Foreground kernel mask

The final composited image is shown below:

Gradient Domain Fusion

Poisson blending applies a least squares solver for pixels within a certain area to find the best color of the image while preserving the gradients. The result for this method is shown below:

Poisson blending doesn't work particularly well in this setting as the color shifts seem unnatural and the seams are not fully removed.

EXAMPLES

Below, we show some of our favorite examples.

Original Image

Original Image

Alpha Compositing

Alpha Compositing

Poisson Blending

Poisson Blending

Be sure to check out our webcam demo as well!

CONCLUSIONS

In conclusion, we accomplished our goal to make a real-time face swap app. Along the way, we encountered a few difficulties, mainly related to getting the facial composite to look as smooth and natural as possible. Some of the main difficulties were getting the Convex Hull cutout around each face to work properly, and figuring out the best way to blend the warped faces into the final resulting image. We wrote the bulk of the program ourselves, but used some pre-existing models to detect faces and facial landmarks. Overall, this project was a success and we are very happy with our results.

Thank you for reading!

ILLINOIS