Duck Efface:
Decorating faces with duck bills aka “Duckification”

Marlee Gotlieb, Scott Hendrickson, and Martin Wickham
University of Wisconsin, Madison
CS 534, Fall 2015

http://pages.cs.wisc.edu/~hendrick/cs534/Duckefface-web/index.html

Abstract
Facial recognition in photographs has been utilized in a variety of forms. At first, it was used to simply detect where a face existed, but now it goes beyond this by being able to recognize particular features on the face and even recognize a specific person. Lately, a major use of facial detection and recognition has been creating “filters” to go over the recognized faces. These filters use facial recognition to distort or change a person’s face. For example, the mobile application Snapchat has the capability of recognizing a person’s face and putting a “filter” over it such as placing sunglasses over eyes, dog ears on a person’s head, or turning someone into a ghost, robot, clown, and more. This project aims to use facial recognition to accurately create a filter of our own: placing a duck bill onto a person’s face, or duckification.

Introduction, Motivation, and Goals
While there are many picture editors and filters available, we noticed there was one common “pose” in photographs that was not specifically addressed by any of these editing tools. The “duck face” is commonly made by people in “selfies” and pictures in general. Done by pursing the lips together outwards, this pose resembles a duck’s bill, hence the name “duck face”. The following celebrities’ pictures exhibit examples of the duck face:

We wanted to create a filter that could detect a person’s face, recognize the “duck face” being made by the mouth, and place a duckbill over the mouth. We planned to identify any and all faces on a chosen picture, use facial recognition to find facial landmarks, calculate the necessary transformations in order to place the duckbill onto the different faces, and render the duckbill onto the recognized faces. In the end, this project

automatically places duckbills onto *all* faces in an image, and not just ones making a “duck face”.

Background and Related Work

Most facial recognition uses a training dataset to learn where certain features on a face are located. For this project, we used the dlib library and Helen Facial Feature Dataset to recognize faces in an image.

The Helen Facial Feature Dataset, according to a paper by the creators of the dataset, builds a “facial feature localization algorithm that can operate reliably and accurately under a broad range of appearance variation, including pose, lighting, expression, occlusion, and individual differences.” It used 2000 training and 300 test images in order to construct its dataset, and it generates accurate results even on high-resolution images.

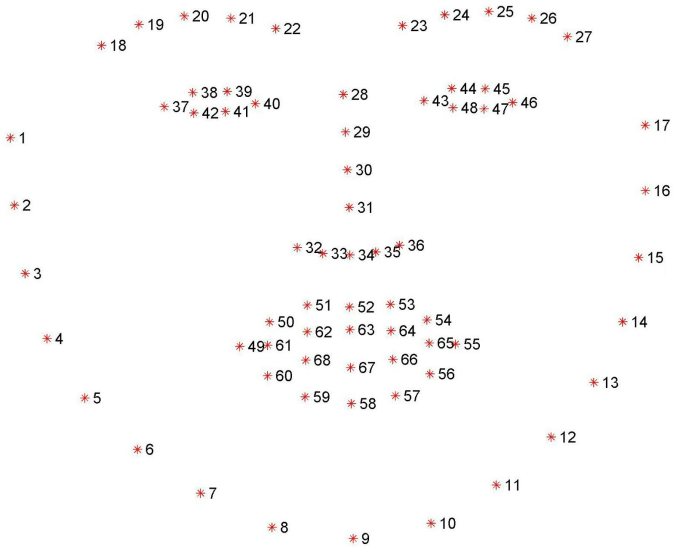
Dlib [1] had a pre-existing implementation of facial landmarking, trained with the Helen dataset. We leveraged these capabilities in our approach to duckification.

The next problem to solve was a version of the Perspective-n-Point problem. We have n points, outputted from the dlib facial detector, and we would like to calculate a transformation matrix to rotate those points, in 3d space, back to their original configuration measured from a model face. We chose a simple iterative approach named POSIT [4]. POSIT allows us to calculate a transformation matrix quickly, in 4-5 iterations of matrix algebra. There are more optimal solutions, but they seemed like overkill for our needs. We ported POSIT from matlab source freely available, to C++, using dlib’s linear algebra libraries.

Method and Implementation

Duckification is implemented in 3 basic steps: facial detection and landmarking, transform matrix calculation, and rendering.

Facial detection and landmarking is implemented with dlib[1]. Dlib exposed a simple to use API, which made setup very simple. The facial landmarker detects 68 unique points on the face, corresponding to the 68 unique points in the Helen Facial Feature Dataset [2].

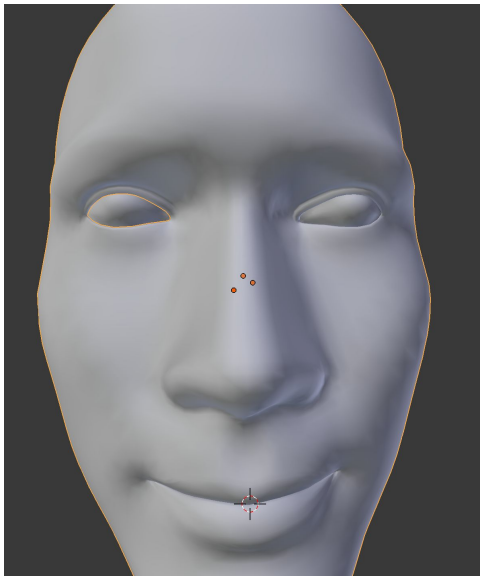


68 point markup from Helen dataset [3]

For our transformation, we use a subset of these points, to allow for quicker POSIT convergence, and to avoid excessive errors. We chose the following subset of points:

37 (outer-left eye)
40 (inner-left eye)
43 (inner-right eye)
46 (outer-right eye)
34 (tip of nose)
49 (left mouth corner)
55 (right mouth corner)

We manually measured the coordinates of each these points on a 3D model of a face found online.



Coordinates of each point, relative to center of mouth, and corrected for symmetry (labeled with the Helen dataset indices).

Helen Dataset Index	x	y	z
37	-44.43	47.56	62.53
40	-19.05	44.89	61.99
43	19.05	44.89	61.99
46	44.43	47.56	62.53
34	0	-5.87	38.28
49	-25.22	13.17	-3.60
55	25.22	13.17	-3.60

T1. Measured points from reference face

Once a facial landmarking is finished, we now have all the data we need to calculate our transformation.

The transformation is an implementation of POSIT [4], ported from Matlab code that is publically available [5]. The port was made easier due to dlib's build in linear algebra library, which made most Matlab operations trivial to implement in C++. Our implementation of POSIT takes the following inputs:

image_points: Our list of 2D points outputted by facial landmarking. These are the measured xy pixel coordinates of the subset of the Helen Facial Feature Dataset feature points discussed from earlier.

object_points: The pre-calculated 3-D coordinates of each feature point, as measured on the reference face from table [t1].

focal_length: An estimated focal length that the picture was taken at.

center: The center coordinates of the photograph (**IS THIS RIGHT?**).

POSIT uses an iterative approach to converging on a transformation matrix, usually converging within 5 iterations.

After converging, out implementation of POSIT returns a combined transformation-rotation matrix.

We then apply this transform matrix to a duck-bill model, also found and modified from an online source. The transformation, and rendering process are applied with OpenGL.

Results

The four images above show the results of our project at the different stages. The top-left image is the original input image, which contains seven different faces to be detected. The top-right image shows the result after detecting the facial landmarks, outlined in green. The bottom-left image includes the transform placed on each of the faces. Finally, the bottom-right image displays the completed product as all seven faces have a duckbill placed onto them.

Above and below, more group images exemplify how our project accurately places duckbills onto several faces, even if some are not directly facing the camera, or if they are at different distances from the camera.

The image below shows another example of duck bills accurately being placed on a group image.

Future Improvements/Conclusion

Our final result successfully implemented the basic ideas of duckification for our project - taking an image with faces and rendering duckbills onto those faces. We were very happy with the results of the project and the implementation we were able to accomplish. Looking ahead, here are some further improvements that could be made to our project:

1. Morphable Face Model: Instead of running POSIT against only one set of 3d points extracted from a single model face, a modified version of the algorithm should be run against multiple face models of differing form, to find the best “fit”. This will allow for significantly better alignment of the duck bill.
2. Fix rotation errors: In many of our tests, the duck bills seem to be pointed too far “down”, and don’t seem to ever point “left”. We have yet to find the source of this error.
3. Only apply to actual “duck faces”: Originally, the goal of our project was to take in an image, find faces that are doing the “duck face”, and position a duckbill onto these faces. As of now, the capabilities of our project are just to find *any* face and place a duckbill onto it. Therefore, one future improvement we had an idea for is to implement this “duck face” detection feature. We would have to use a training set of people making the “duck face” in order to teach facial recognition for duck faces specifically. This improvement would show the true meaning of duckification.
4. Increased accuracy of filters: Although our project is able to resize and transform a duckbill in order to align it with different sized and angled faces, a future improvement one could implement with this project would be to make the duck bill better blend with the face. This could give the

image the appearance that the duckbill is actually part of the face and not only placed on top of the image. One could blend the edges of the duckbill with the colors of the skin. Also, one could use information on the colors of the image and people in it to give the duckbill more accurate coloring. One could even use the contours of the face, extracted from landmarking, to apply 3d morphing and blending techniques.

The previously discussed improvements apply to more accurate duckification. However, our project could be extended beyond the basic duckbill as well by making it possible to place other images or objects onto an input image. For example, you could replace a human's nose with a pig snout or ears with elephant trunks. Different feature points might need to be used to make this possible, but the basic face detection algorithms could be used for a basis of any project similar to this.

All in all, we were very pleased with the results of our project. Our implementation works very smoothly to integrate a duckbill onto an image and is an excellent basis for future projects that utilize facial detection.

References

1: dlib (http://dlib.net)
2: Helen Dataset (http://www.ifp.illinois.edu/~vuongle2/helen/)
3: Helen Dataset Point Indecies (http://ibug.doc.ic.ac.uk/resources/300-W/)
4: POSIT: Model-based object pose in 25 lines of code (http://www.cfar.umd.edu/~daniel/daniel_papersfordownload/Pose25Lines.pdf)
5: POSIT Implementation: (http://www.cfar.umd.edu/~daniel/classicPosit.m)

Code Details

Facial Landmarking: dlib (http://dlib.net)
Matrix Library: dlib
POSIT: A port of code from “Model-based object pose in 25 lines of code” (http://dl.acm.org/citation.cfm?id=204015)
OpenGL: The OBJ loader was based on one written by Martin for a CS506 project 2 years ago, and modified for this project. The rest of the OpenGL code was written from scratch for this project.

Team Member Contributions

POSIT Porting - Scott & Marlee
OpenGL Rendering - Martin
Dlib experimentation - Martin
Helen Dataset investigation - Scott & Marlee
Manual Model Face Measurements - Scott & Marlee
Duck Bill Creation - Scott & Martin
Paper - Scott, Marlee, & Martin
Presentation - Marlee (small additions by Scott & Martin)