ArtMe: 15-463 Final Project

By Caroline Hermans

Overview

For my final project I created ArtMe, an iOS App that lets people insert their faces into famous paintings. The app uses the AVFoundation and CoreImage frameworks to manipulate and display camera feed in real time. No matter how far away the person's face is or what location the face is, the app will find, resize, crop, and shape the face according to how the painting is arranged. Also, the camera feed is filtered in real time to make it fit in with the skin tone of the paintings. Finally, the person's edited face is displayed under a semi-transparent section of the painting with the face removed, which allows for a smooth edge transition from the painting to the face of the user.

Project Context

I made ArtMe because I wanted to use computational photography to do something both challenging and fun. The live photo booth aspect uses computer vision and image manipulation methods to let someone have a good time trying out different paintings and making funny faces. I'm a firm believer that the benefit of computer science is that it allows people to do things they couldn't do without the magic of computer vision. With a fun, inviting UI, computational photography can bring out a fun and enjoyable experience.

I was inspired to build ArtMe by my experience with Snapchat, a company that I really admire for using technology and computational photography to build something fun that people love. Last summer, I interned on the camera team at Apple, and there I learned that it's more challenging than it seems to do filtering on live camera feed in real time without the feed lagging. I built on the knowledge I gained during my internship to build an app that turned out to be more complex than I initially anticipated. Everyone I've shown ArtMe to has had a fun time. Ultimately, I want to polish up the app and put it on the App Store.

Algorithms

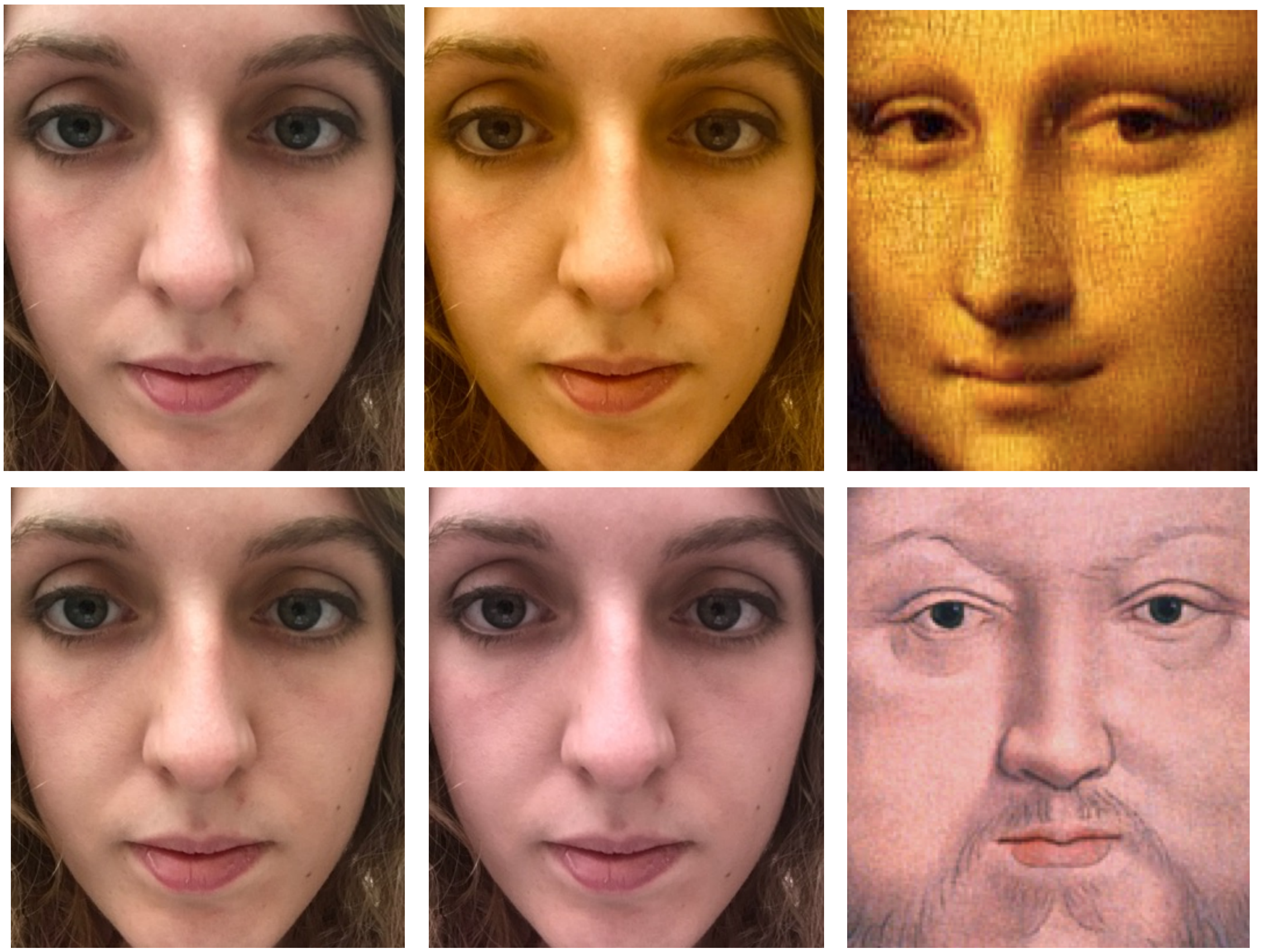
There are three key aspects to the functionality behind ArtMe: filtering the live camera feed, blending the face with the painting, and doing the actual face detection and resizing of the live camera feed.

Filtering and Skin Matching

In order to filter the camera feed, I used GLKit within Xcode to render the camera feed with OpenGL, allowing me to perform more computationally expensive image manipulations without losing the real-time aspect of the app. Every frame, I have the camera feed outputting the current image as video data. In my capture event handler, I handle each frame as a separate CIImage.

Based on which painting has been selected by the user, a different set of filter values are used. For example, *The Mona Lisa* has a much warmer image temperature to create a somewhat yellower look, whereas *Girl with a Pearl Earring* has a higher exposure to match the pale skin on the girl in the painting. In order to select the filter for each image, I used my knowledge of photo editing to play with values that could adjust the temperature, brightness, contrast, and tint in order to match a different skin tone.

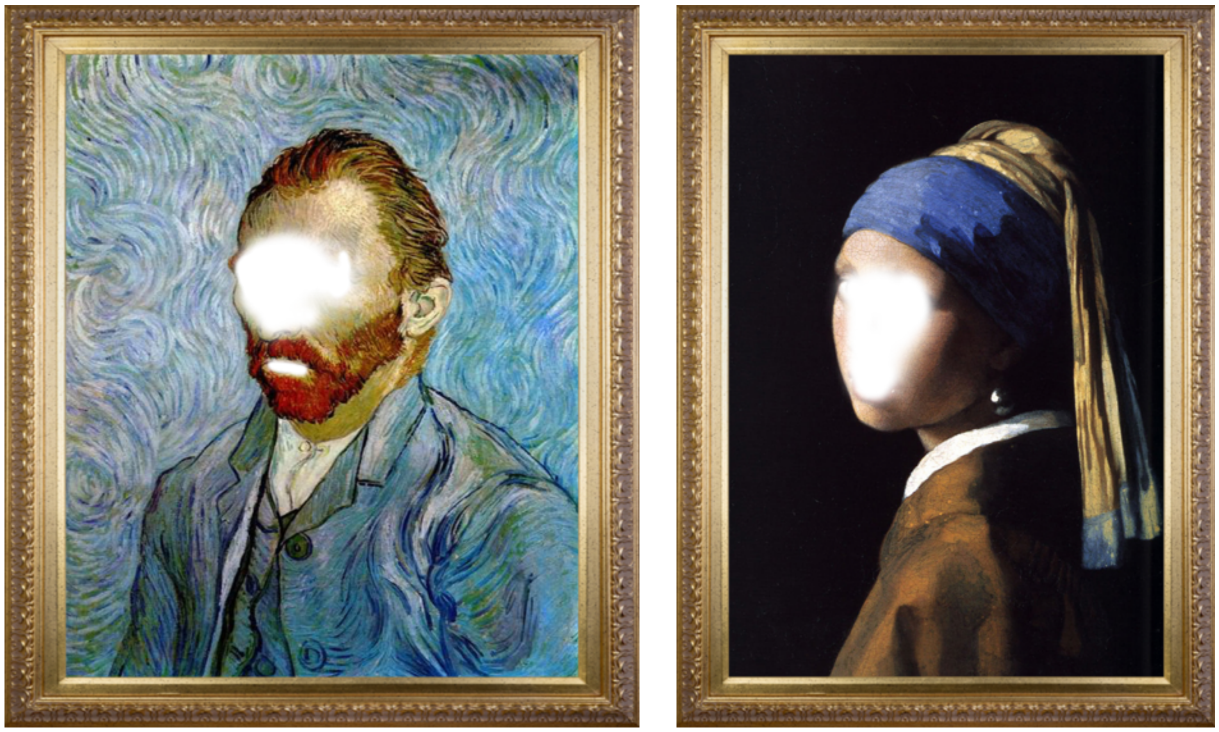
Once the filter values are selected based on the painting that's being used, I apply the CIFilters to the current CIImage and update the current video data output to use the new image.



Different painting skin tone filters on real faces

Blending

It would look jarring to copy and paste the face directly on the painting's face, so some sort of blending is required. Initially, I wanted to use linear blending to smooth the transition between the faces and the paintings. However, when I attempted this, it wound up slowing down the app significantly. Since it's important for the app to work in full time without lag, I thought of a way around the time-intensive computations that still allowed the blending effect to take place. Instead of blending the photos in real time, the painting images themselves are edited to have a transparent hole where the face is, with blended edges. That way, I can display the face feed behind the hole, and still achieve the same blended looking effect. As a bonus, this allows me to maintain the facial hair in Van Gogh's *Self Portrait.*



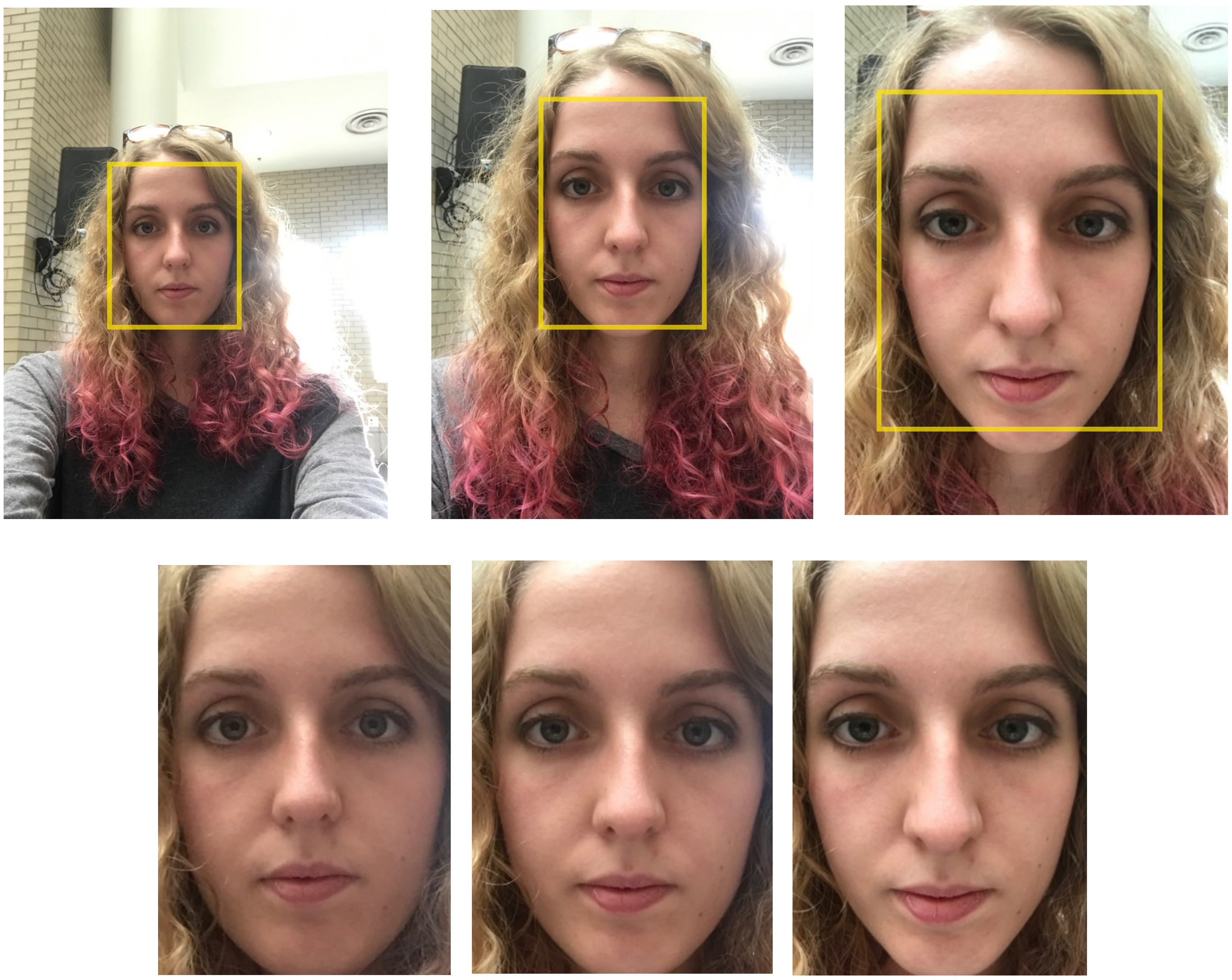
Solution allows for blending-like effect in real time without being computationally expensive

Face Manipulation

The first step to face manipulation is to run face detection. In order to do face detection in real time, I needed to use a face detection system that was fast and possible to run without causing any lag.

In Objective-C, there is a CIDetector object that can be configured with any sort of classifier to search for faces. I created a CIDetector object using a facial detection classifier that searches the screen for faces every frame. Then, I give the CIDetector the current frame as well as the orientation of the frame (the CIDetector is not orientation-agnostic, it matters which way the camera is rotated) and I check to find the bounding box of the face.

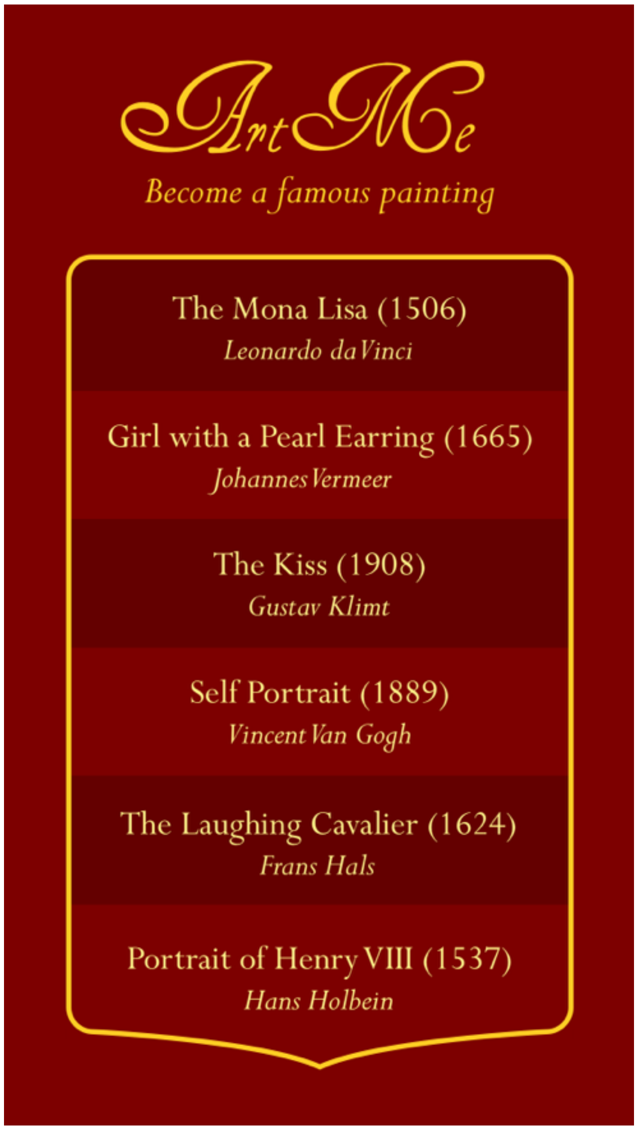
The most fun part of playing with this project is that no matter where you move your face, it stays the same size and in the same place in the live camera feed in front of you. This is because every frame I'm running face detection, and then cropping the camera feed image to include just the face. Once I crop the camera feed to include just the face, I resize the face to match the size of the face in the painting. Finally, I move the face to match the location of the face in the painting. These offsets and sizes are unique to each different painting, and change depending on which painting is selected.



Creating consistent same-sized face images from different-sized photos of faces

Interface

Since I wanted this app to be enjoyable, I spent time making sure the interface was easy and fun to use. I designed the whole app to look like a fancy museum, with golden picture frames and fancy cursive script. I also let the photo booth part of the app take over with the big yellow camera button. I additionally built custom transition classes within Xcode so that it swipes right to left between the different views.



ArtMe
Become a famous painting

The Mona Lisa (1506)
Leonardo da Vinci

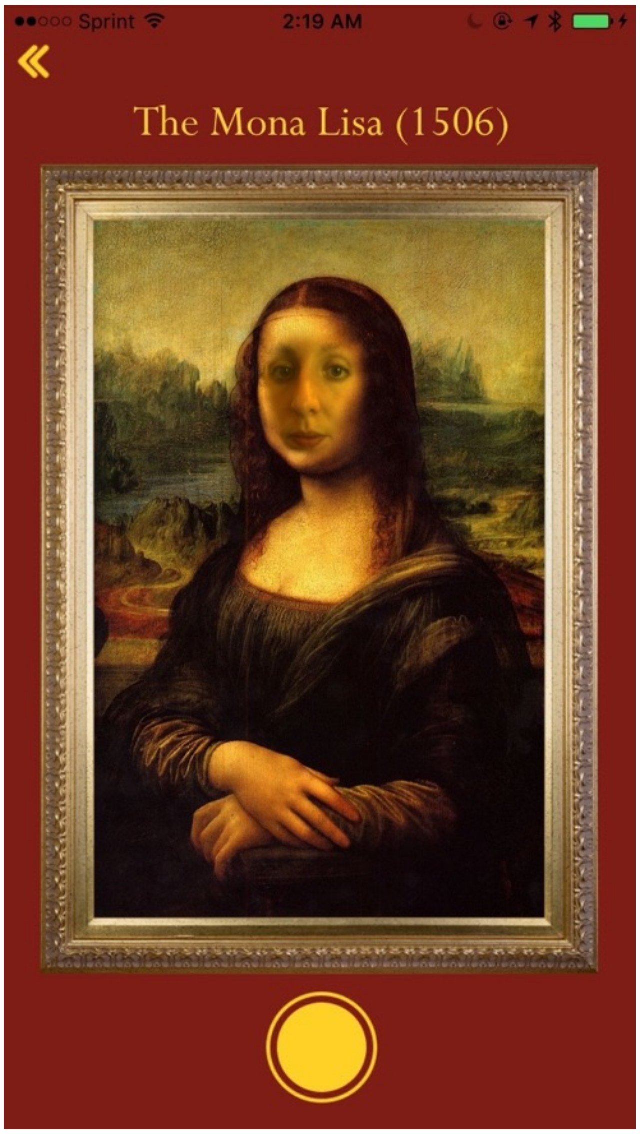
Girl with a Pearl Earring (1665)
Johannes Vermeer

The Kiss (1908)
Gustav Klimt

Self Portrait (1889)
Vincent Van Gogh

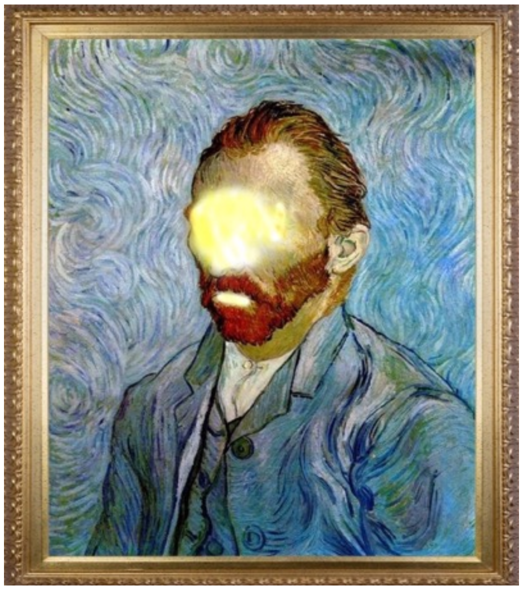
The Laughing Cavalier (1624)
Frans Hals

Portrait of Henry VIII (1537)
Hans Holbein

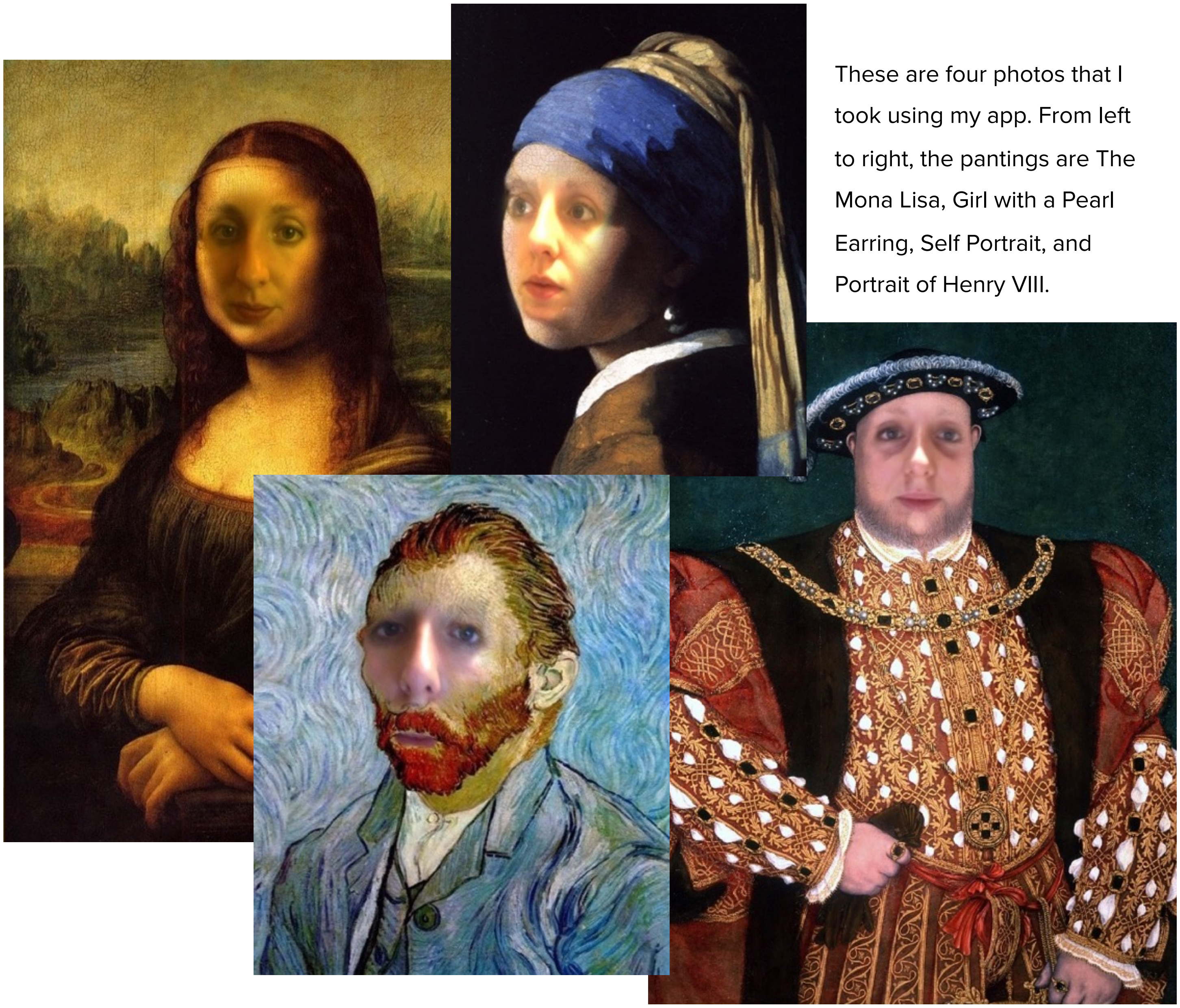


Challenges

Because the app had to be in real time, a major challenge was to make all of the image processing fast enough so that the app didn't lag. I did achieve that by using OpenGL and compromising on the image blending, but it was definitely a challenge with implementing this app, as I'm sure it is with any real-time camera application. Additionally, every once in a while the face detection does fail to find a face. This creates results where unexpected images are put in the faces of paintings.



Results



These are four photos that I took using my app. From left to right, the pantings are The Mona Lisa, Girl with a Pearl Earring, Self Portrait, and Portrait of Henry VIII.