Caroline Hermans

15-112

April 29, 2015

**FourEyes Design Documents**

Anyone who wears glasses knows how overwhelming it can be picking out a pair. There are so many styles, and each style falls very differently on different faces. There is literature out there online that tells people what glasses are flattering on which types of faces, but how do you know what type of face you have? This dilemma is exactly the one I’m solving with my term project: users come in with nothing but their face and leave with information and an idea about what type of glasses will flatter their face the most.

My program is divided into several phases. The first phase involves getting the face shape of the user. The next phase supplies the user with general information about their face shape and the glasses they should wear. From there, the third phase is the try-on phase where users can overlay the glasses on their face. The final phase presents the user with a gallery of commercially available recommended frames to search through.

**PROGRAM DESIGN**

The structure of my program is fundamentally built on Booleans that represent each phase of the project. Initially, all of these Booleans are set to “False”. For example, initially, the variable “tryThemOn” is set to False. However, when the “Try Them On” button is clicked, my program sets “tryThemOn” to True and all of the previous phase Booleans to False. This is the underlying format of my term project. My draw functions then check these phase Booleans to see what phase the project is in, and draw the corresponding screen.

**Object Oriented Programming**

I use a significant amount of OOP in my project. Most notably, this can be seen in the “Browse These Frames” screen, where each individual glasses display is an object. The glasses are assigned a name, price, brand, link, and image, as well as a page number and a position on the screen based on being sorted by price. The position is numbered 1-4 for the 4 possible locations in the grid. This information is used for drawing the glasses box and for click checks. The glasses objects have methods such as linkToWebsite(), isClicked(), and huge fancy draw methods that rely on both their position and page number.

I actually use OOP for two other purposes in my project. My buttons are all objects that I created (not default Tkinter buttons and not regular static rectangles), and I have a class DarkerButton that extends Button for some extra design flair. The dots in the “Drag The Dots” screen are also all objects.

**OpenCV Meets Tkinter**

My program also converts every webcam frame from OpenCV continuously into the Tkinter Photoimage type. This allows me to mix and match desireable traits of both Tkinter images and OpenCV images. If I want to use Tkinter functionality on an OpenCV image, I’ll convert the image and then draw on top of it in Tkinter. If I want to use OpenCV functionality, I’ll mess around with the image before converting it.

The “Drag The Dots” screen is an example of converting OpenCV to Tkinter and drawing in Tkinter on top of it. On the other hand, the “Try Them On” webcam feed is entirely done in OpenCV, which reads in the glasses image files, makes them transparent, and overlays them on the user’s face with continuous face tracking. That entire image is then converted to Tkinter. The fact that the entire frame is an OpenCV frame originally allows me to call OpenCV functions such as cv2.imwrite() on the image, which will save the picture to my desktop.

**Calculating Face Shape**

My face shape algorithm is based on extensive research, sources included in the sources.txt file. Essentially, it cycles through the four possible face shapes and sees which one matches up with the user’s facial proportions.

An oval face corresponds to a face length that is 1.5x greater than face width. A heart face corresponds to a forehead width that is more than marginally greater than the chin width. A square face corresponds to a face where the forehead width and chin width are approximately the same, and a round face corresponds to a chin width slightly less than the forehead width. FourEyes turns the two measured facial distances into ratios and goes through several checks of how similar the ratios are to the various target ratios. My program categorizes the user’s face based on the ratios and their similarity to ideal oval, square, heart, and round shapes.

**Recommending Glasses**

Again, this methodology is based on research. Luckily, this part is logically straightforward. All of the sources I looked at recommended a very clear relationship between face shape and glasses frames. Oval faces correspond to square frames, round goes to rectangular frames, heart goes to thin frames, and square goes to round frames. More information about why this is the case can be found in the write-ups I’ve included my source files.

**Sizing the Glasses in Try-On**

This was actually a fun problem to solve. I first chose a standard glasses size and loaded that image in. I already knew how to get face width using OpenCV, so I had my program continually printing out the width of my face as I moved closer to and further from the webcam. I moved my face to where the glasses looked good without being scaled, and recorded that face width. Then I moved my face to where I wanted the glasses to be twice as large, and recorded that face width. I continued doing this for several points, collected all of the data, and performed a linear regression on it to come up with the following equation:

Scale=0.003\*(faceWidth-0.3)

Critically, the glasses image files are all initially the same sizes

**Tracking the Glasses in Try-On**

I use something called Cascades to detect the center of the user’s eyes, and continually update the x and y coordinates of the glasses according to where the users eyes are. I average out the x and y coordinates of the two largest eyes in the frame, and place the glasses in the middle.

**Multiple Faces at a Time**

Multiple faces in the frame was interesting to handle, and still is sometimes a bit buggy. My fix for this problem is to detect the largest face in the frame, and do operations on that. I do something similar with eye tracking; it detects the two largest eyes it finds and returns those, rather than returning the average coordinates all of the eyes it sees. This isn’t a perfect fix, and could definitely use some work.

**USER INTERFACE DESIGN**

This design is intended to present the user with a clean, friendly user experience, while additionally carrying an air of stylishness. The user should be able to use the product easily and have fun doing so, while also trusting what it has to say. For this reason, the “style app” functionality of it gets across with the design. The target audience is anyone who wants to know what type of glasses would look good on them, whether they actually need to wear glasses or not.

**Colors**

I have three colors: a background color, an accent color, and a highlight color. The colors are shades of purple ranging to white allowing for drop shadows and subtle differences in shade.

Background Color:

Hex 423362

Macintosh HD:Users:carolinehermans:Desktop:Screen Shot 2015-04-28 at 9.39.46 PM.png

Accent Color:

Hex C3BAEB

Macintosh HD:Users:carolinehermans:Desktop:Screen Shot 2015-04-28 at 9.40.24 PM.png

Highlight Color:

White

**Fonts**

Helvetica (for titles)

Avenir Next (for body text)

**Clicking**

Nothing is static in my term project, everything changes colors when it’s clicked to give the user feedback. When the buttons are clicked, they become shaded in and set into the screen. When they are released, they pop back up.

**Dots Dragging**

When the dots are clicked in the “dragging dots” screen, they change color, as do all lines that are connected to that dot.

**Instruction Pages**

Through extensive user testing, I finally settled on instruction pages for “Drag the Dots” that got accurate results. First, I tried to list out all of the steps along with the diagram, but people just skipped over it. Over time, I converged on a product that people inherently know how to use without needing to spend very long reading.

**Other Programs**

No other programs I found looked anything like mine, because mine provides a very different service than anything else I’ve seen. It’s not quite a quiz, in the way that the LensCrafters face shape glasses feature is. However, the design of online glasses stores such as FramesDirect did influence my “browse frames” screen, and I formatted the information in a way that was consistent with various other “shopping” technologies.

**Design Justification**

The design of my project achieves what it needs to: a clean, friendly user interface that people will trust to give them information. It is also purple, which doesn’t exclude men but also contributes an air of style to the project. Importantly, it is also fun to use, and users will enjoy their experience! This is largely due to the clicking feedback, and the way things respond in my program. It’s easy to use as well, and users don’t have to invest much time into reading instructions.