Took a quick look at a Shiseido web page. Trying to extract colors from the HTML would be hopeless. The HTML is very complicated and would differ from web site to web site. The backgrounds are white. The patches are solid.

Looked at the Sephora web site. Looked at Gorgio Armani Luminous Silk Foundation (Armani.png). The backgrounds are white. The patches show brush strokes. Watched a video explaining all the ways of applying the foundation, along with a bunch of other cosmetics. Choosing the right thickness is very important.

Also looked at Estee Lauder Double Wear Stay-in-Place Makeup. Once again the patches show brush strokes. None of the patches look very dark. However there are also examples of arms of various colors with 11 patches of foundation on all of them. I assume they all have the same 11 patches. Some look very dark. I assume the patches look lighter because the impression is dominated by the places where the makeup is put on thinly. You specify the shade by clicking on a patch. The shade names appear next to the word “color”.

Also looked at Make up for ever pro Finish Multi-use Powder foundation (not a cream). In this case, the patches do not show brush strokes, but do show some graininess.

Dan is leaving. How do we get enough information to continue? I should install the development environment. I should also learn how to run the application. I should make an architecture diagram so Dan can tell us how to fit my proposed code into the application.

Installed Anaconda and PyCharm and sent them a note with the architecture diagram. Looked at some more cosmetics web sites. I think I’ll just use the darkest color from the Sephora web site and the solid color from the others (that is all they show) and call that R∞, the color of a very thick layer of the cosmetic. I can then calculate color of the skin with varying amounts of this cosmetic using Kubelka-Munk theory. It turns out that when the cosmetic matches the average skin color, which is the normal usage, the Kubelka-Munk formulas get very simple.

Figured out how to get a command tool using Navigator. Go to the environments section, make a new one, and click on a triangle to get a pull down menu for making a command tool that will accept conda commands:

conda create -n glam python=2.7 anaconda

source activate glam

conda install -c menpo dlib

conda install -c menpo opencv3

pip install imutils

Made a new PyCharm project and hooked it to the Glamtech environment. Made and ran a ‘hello world’ python script. Then tried to import OpenCV. No dice, though it is clearly in the Glamtech environment. However PyCharm was only showing me the root environment, not the one in which I had run the conda commands. When I added it to the list, it worked.

But when I tried to run one of Dan’s scripts, it balked at the import dlib command. It might be that the system was getting confused by all my previous incorrect attempts, so I uninstalled all the software and reloaded it. This didn’t work so Dan got rid of the dlib command in his XML analysis code and helped me import the additional libraries I needed to run it. He sent one more note explaining how to get useful outputs from his script, but I haven’t implemented it because I don’t think I’ll be going further with the face segmentation analysis.

Also listed some useful sites.

Anaconda:

<https://www.continuum.io/downloads>

<https://docs.continuum.io/anaconda/install/windows>

<https://docs.continuum.io/anaconda/navigator/tutorial>

Face segmentation:

<https://pdfs.semanticscholar.org/d78b/6a5b0dcaa81b1faea5fb0000045a62513567.pdf>

<https://web.stanford.edu/~hastie/Papers/ESLII.pdf>

Face segmentation model:

<http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2>

PyCharm:

<https://www.jetbrains.com/pycharm/download>

Helen dataset

<https://www.dropbox.com/s/rxscw8jo64hiiec/helen_model.zip?dl=1>

Implemented the scripts needed to extract the darkest color from a patch image pulled off the web.

Started making the images I’ll need to test the code that shows a face with the cosmetic applied. Extracted some face regions. Next I’ll have to split the variation in face color into a variation in face illumination and a variation in skin reflectance. I really do not have the information I will need to do this exactly, but I’m just trying to get a reasonable looking simulation. To do this I’ll assume that the skin reflectance is constant, so the illumination variation will give us shape from shading. Then I can add some blemishes to the skin reflectance image and cover it up with cosmetic.

The first step is to be able to calculate a reasonable looking bare face image with:

RGBface = Yshad \* RGBaveref

Where Yshad is a gray image giving us shape from shading. In fact, the illuminant might not be neutral, but we can get a reasonable looking simulation assuming it is. All we know is RGBface so we can get this result with a big Yshad and a small RGBaveref or a big RGBaveref and a small Yshad. To get reasonable values for Yshad note that:

Yface = Yshad \* Yskin

Where Yskin is the neutral reflectance of the average skin color. Furthermore, we could determine Yskin if there was an object of known reflectance illuminated the same way as the skin. One possibility would be the sclera of the eye and the surrounding skin illuminated without obvious shadows. This is clearly future work. For now we will just pick a reasonable value for Yskin. Then:

Yshad = Yface / Yskin

We now have to pick a RGBaveref consistent with the assumed Y. To do this we can calculate:

RGBref = RGBface / Yshad

Hopefully RGBref will be reasonably constant, but in any case we should calculate its weighted average, since the bright areas of the face will give more accurate values than the darker ones.

RGBaveref = ∑( RGBref \* Yshad) / ∑ Yshad = ∑ RGBface / ∑ Yshad

I need to go from sRGB to RGB to do these calculations, but there is no sRGB to RGB color transform in the OpenCV library. Just used ‘for’ loops and sRGB to RGB subroutines, and it goes fast enough.

I next add a pimple to the face reflectance, and combine with the shading to get the appearance of the bare skin.

Finally I use Kubelka-Munk to calculate the reflectance of the skin with a layer of cosmetic with the desired color and the desired thickness (opacity), and combine with shading to get the RGB color. I’ve chosen the cosmetic color to be equal to the average color of the skin, which would be the usual usage. The final step is to convert to sRGB so it will look right on a monitor.

Note that it is possible for the RGB color to be greater than 1.0, and this does strange things on the conversion to sRGB. In places where this might happen, we slightly scale down the offending separation so it never exceeds 1.0.

I’ve enclosed a paper on the Kubelka-Munk theory. I’ve played with the algebra to make it easier to implement. The final equations are:

F = exp(-opacity)

G = R∞ \* R∞ + Δ \* R∞ - 1

R = (Δ \* F – G \* R∞) / (Δ \* F \* R∞ - G)

These equations are applied for each separation. Δ is the difference between the skin reflectance and R∞. R∞ is the R, G, and B reflectance of a very thick layer of the cosmetic. R is the R, G, and B reflectance of the skin with the cosmetic applied.

The key functions are in colorLib.py. The colorDriver.py code shows how to use them. Most of the code makes and displays the simulated images. The key functions are getPatchColor() and showFace().