6.8371 Pset 9
David Xiong

During this assignment I worked on a coherent line drawing algorithm. This algorithm uses any input image and outputs the same image drawn with black lines on a white background. I chose to implement this because I am interested in non-photorealistic styles of rendering and animation. Using a filter like this, it is possible to change pictures or videos into a unique style.

There is a history of non-photorealistic rendering relating to line drawings. While some methods produce line drawings on 3D models, this assignment focused on 2D pictures. In Salisbury et al. [1994] (https://www.cs.ucdavis.edu/~ma/SIGGRAPH02/course23/notes/papers/Salisbury94.pdf), a method is developed to draw images in an ink pen style, using different textures consisting of strokes to fill in the interiors of the image. This method also relies on an edge-detecting algorithm to find the outline strokes for the image. Gooch et al. [2004] (https://dl.acm.org/doi/pdf/10.1145/966131.966133) demonstrates a method to produce illustrations of faces given photographs. Instead of using Canny edge detection, a difference-of-Gaussian filter is used for edge detection. The paper this assignment is based on, Coherent Line Drawing by Kang et al. [2007] (http://www.umsl.edu/cmpsci/faculty-sites/kang/publications/2007/npar07/kang_npar07_hi.pdf), extends on the difference-of-Gaussian method to produce a more coherent line detection algorithm. This paper introduces an NPR algorithm to generate line drawings from photographs while keeping smooth and stylistic lines.

The algorithm of Kang et al. is not too complex. It involves first creating an edge flow field, which represents the directions of the edges. The method uses a kernel applied to every pixel of the image, to calculate a smooth vector field which preserves stronger edge directions, while weaker edges are covered to align more with the strong edges around it. The algorithm first starts with a tangent vector field, which is the gradient field rotated 90 degrees counterclockwise at every point. Then, the following kernel is applied to the tangent field:

$$\mathbf{t}^{new}(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{y} \in \Omega(\mathbf{x})} \phi(\mathbf{x},\mathbf{y}) \mathbf{t}^{cur}(\mathbf{y}) w_s(\mathbf{x},\mathbf{y}) w_m(\mathbf{x},\mathbf{y}) w_d(\mathbf{x},\mathbf{y})$$

The tangent field is updated by looking at the neighboring pixels, and calculating a weighted average. The w_s specifies the neighborhood by setting a weight of 1 for every pixel within a radius. The w_m term gives bigger weights to pixels who have higher gradient magnitudes. These correspond to those with more dominant edge directions. W_d represents how closely aligned the angles of the tangent vectors are. In all, using this filter smooths the tangent lines to align closer together, putting more weight on stronger edges. This kernel could be applied multiple times if desired.

Next, a flow-based Difference-of-Gaussians filter is applied to the edge flow field, in order to produce the coherent lines. For each pixel, and area surrounding the pixel aligned with the flow field is used. The difference of Gaussian is applied in the gradient direction, and integrated. By applying the filter this way, the output can be exaggerated along stronger edges, while not showing on weakedges. This helps produce the coherent lines. Finally, a binary threshold is used to color each pixel as white or black. The final result is coherent, smooth lines which capture the main features of the image.

For implementation, the assignment was coded in C++ with the Image class used in previous psets. The equations corresponding to edge flow and difference of gaussian filter were implemented. For the integrals in the DoG part, the area was discretized and summed. The article contained all the equations needed, which made it easier to implement. However, there were some parts where details were sometimes left out, which was more confusing. In total, the implementation runs well, however the results do not produce lines that are as coherent as seen in the article. This could be due to an error in the code, which would be the next step given more time.

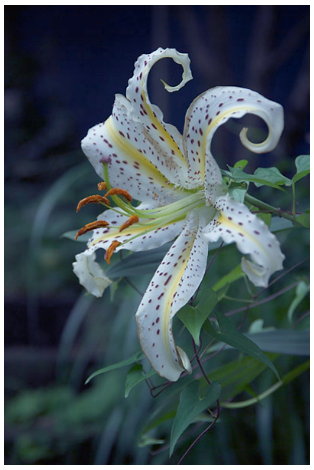
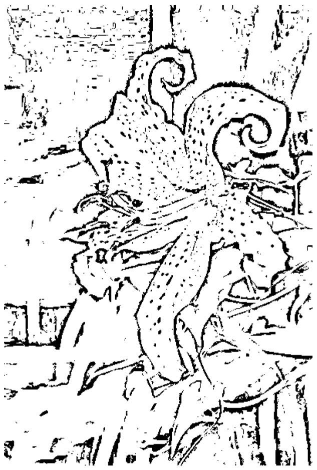
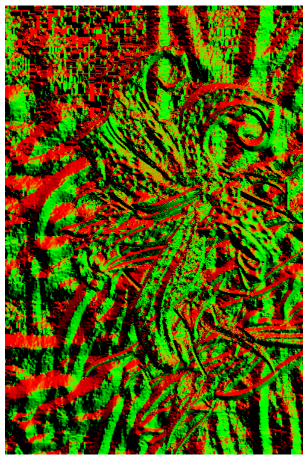
The edge flow field was an intermediate test case, with the horizontal values in the red channel, and vertical values in the green channel.

I used the following test cases:

1) Edge flow field image with 0 radius. This resulted in a black picture as expected.
2) Edge flow field image with 3 sigma, n=0. This just calculates t0, which is the gradient field rotated. This performed as expected.
3) Edge flow field image with 3 sigma, n=1.

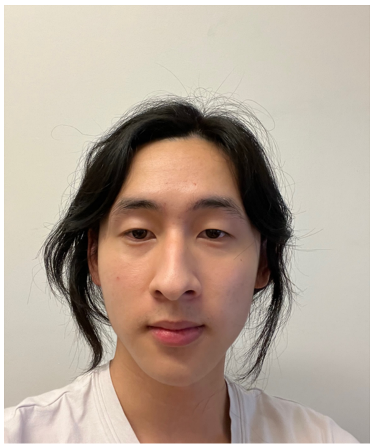
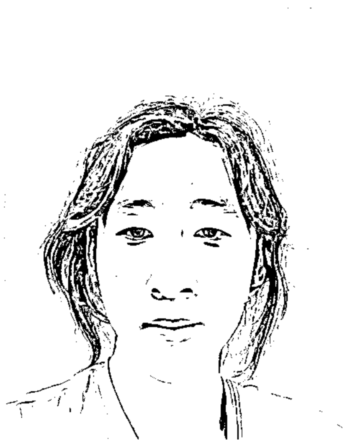
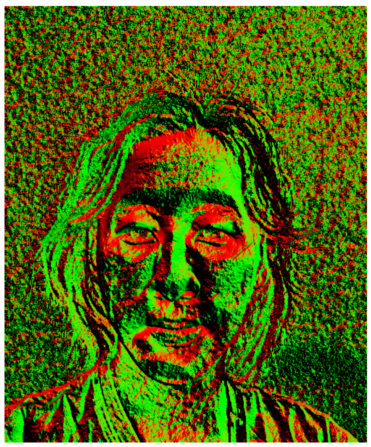
4) Testing the Gauss values function, to see that I did calculate them correctly. This did perform correctly with what I had expected.
5) Full flower test run. This did result in lines around the outlines, but wasn't as coherent as I had hoped.

Here are the results from the algorithm, run on the flower image from an earlier pset:

Original, Line drawing ETF, n=1

Next, I tried using the algorithm on a picture of my face.

Original Line Drawing ETF, n=1

This algorithm is very quick to run, taking about 5 seconds per picture.

Ethics Writeup

Imagine that your supervisor has given you the task to create a system to manipulate images of people to make them look more attractive.

If I was asked to create a system to manipulate images of people to make them more attractive, here are some of the specific changes I would propose: Make the skin smoother, make the face and body more symmetric, slim down some parts of the body if they were not already skinny, make the person's facial expressions more happy. These changes could be construed as unethical or irresponsible because they are misleading viewers on what the person looks like in real life. When given a photo that looks very realistic, the common assumption a viewer would have is that it is a photo taken with a camera, which captures a scene of real life. However, when a photo is altered but still remains realistic, people will think the features added -- such as smooth skin, symmetric face, skinny body -- are part of who the person is, not edited on. Also, because beauty is subjective, the photos would be biased toward the direction of what I find to be attractive. The program should not be pushing any kind of specific look. If it does, there would be problems when the user's vision of what they want is different from the programmer's vision. More broadly, editing pictures and presenting them as real is probably damaging to the younger people in society, who would look at models and place high expectations on themselves.

One way to disconnect from ethical problems would be to generalize the system enough such that the main task is not specifically for editing people to make them more attractive, but instead just to edit photos of people. For example, some people may wish to see themselves with longer or shorter hair. The system could allow for changes like this with many options, so that the user can choose to edit themselves however they feel. This has nothing to do specifically with attractiveness, and does not push the programmer's biases on anyone. Likewise, other parts of the body could also be easily editable, as long as there remains a large variety of options that are relatively structured. For example, the distance between the eyes could just be determined by a parameter, and there would be no option which is pushed to be "more attractive". Some people may find automatic skin lightening to be an ethical problem. If a slider was presented for users to lighten or darken their skin tone, with the user being the one in control, there would be less ethical problems. In all, this also helps widen the scope of the project so that more users could find a use for it. Instead of just those looking to make themselves more attractive in images, people can also use it to edit themselves to funny extremes if they find that funny.