Image Processing Assignment Report

CIS: qvhf99

# Problem 1

## Implementation

My implementation of the light leak filter works by first darkening the input gamma by a customizable amount. It then uses the mask to perform a bitwise AND operation on the input image to obtain our foreground lighter layer. It then blurs this layer with a constant box blur to remove harsh edges. This is then added onto the base image with a weight of one as to overlay itself onto the darker background.

To implement the rainbow effect; the filter takes the lighter foreground and adjusts the values of the BGR channels according to some simple linear interpolation based on the given pixel’s position in given row. This creates a smooth, prism-like effect where one end of the foreground is shifted towards blue and the other towards red with green being dominant in the middle.

## Testing

I iterated on this filter a number of times. Once it was functional, I tested it on both input images when I noticed the harsh edges when transitioning the bright light leak to the dark background. For this reason, I implemented a simple box blur on the foreground. I also noticed when providing masks that weren’t totally black/white the output was very strange, so I added thresholding to the mask. The main point of testing was getting the rainbow effect right, I eventually used desmos to graph the exact effect I was trying to achieve.

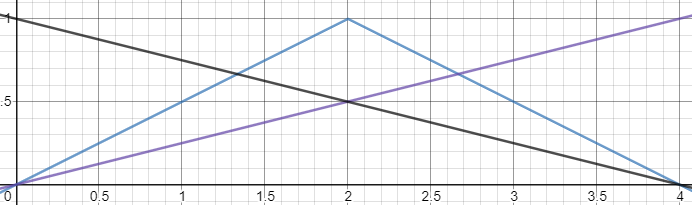


Figure – Graphed rainbow effect

## Computational Complexity

The computational complexity should be where is the number of pixels in the image and is the size of the box blur kernel. This is the dominant term as all the other operations (masking, adding, applying the rainbow effect) have cost or cheaper. This may be lower depending on the exact implementation of OpenCV’s filter2D function, but this should be correct for the simplest implementation method.

# Problem 2

## Implementation

The implementation is very simple. I decided to generate noise following the Gaussian distribution due to its relative regularity, combined with some horizontal motion blur I feel this creates the soft, scratchy lines a pencil would. The noise texture is simply then added to the input image following the blending coefficient given by the user. In the case of the color input, I apply it to the red and blue channels (using separately generated noise textures). I then adjust their gamma to show it’s possible to tint the output color considerably.

## Testing

When first creating this filter, I used a gaussian noise texture from the internet as I wanted to get the core of the filter right before worrying about generating my own noise. When eventually generating my own noise, I encountered a small issue of some pixels coming out white for no reason. This was one case (of many) where I was adding floating point-backed images to integer-backed images.

The main point of testing this filter was finding good values for the Gaussian distributed noise’s variance, a good value for the blending coefficient for colored and grayscale images and how much motion blur to apply to the noise. I eventually settled on some good default parameters (used to generate the below examples) through trial and error. I also tested various types of combinations of blurring, but it made the output too messy and unfocused.

## Computational Complexity

This filter should have a computational complexity of as all operations perform a constant number of actions on each pixel and all operations are performed a constant number of times (i.e. 1-2 noise generations, 1-2 motion blur applications, 0-2 gamma adjusts, 1 image blend).

# Problem 3

## Implementation

My implementation of this filter uses splines of and some given data points for said splines to pass through to build 4 separate lookup tables, one for each color channel and one for the final image. The curves pull up paler red values and lower common blue/green values to create a warmer image, particularly on pale skin/clothes etc. The overall BGR curve of the image is then pushed down to slightly darken the image as a whole creating the filter which I call “Warm Noir”.

## Testing

Writing the code to generate the lookup table was actually very simple once I understood how to use SciPy’s splines. The main challenge was finding curves that improve the quality of the image. I performed most of my testing on the first input file of the woman. I played around in a well-known photo-editing application with color curves until I found something that looked like a filter I would recognize on Instagram (skin tone adjustment is of course always popular).

## Computational Complexity