SKBH77

Problem 1

Results




i) Testing

I tested the image with multiple parameters to find the best-looking image as well as to test that the parameters alter the image as expected.

Below are images where I have altered the darkening coefficient, note from left to right the initial image is darkened.




1.0 1.5 2.0 2.5

Below are images where I have altered the blending coefficient, note how the mask is more prevalent from left to right.




0.1 0.2 0.3 0.4

And now for testing the two modes, light leak and rainbow light leak.


ii) Reasoning

I have chosen to create the first light leak filter by creating a mask of an increased brightness version of the original image. The reason for this is because the light then looks more natural than using a white mask for example. For the rainbow mask, I chose to create this in an external image editor so that I could get the colours precisely. For blending the mask, I decided to alter the blending coefficient at the top, bottom, left and right edges so that there is less of a hard line between the mask and the rest of the image. Instead, the mask's blending coefficient increases towards the middle of the image, up to a maximum of the passed in blending coefficient.

iii) Computational Complexity

For the light leak filter, creating the mask takes $O(height * width * channels)$ time, as it must go through each pixel of the mask (which is the same size as the input image) and set the corresponding value if its black. When the picture is square this makes it it $O(n^2)$. The rainbow light leak is faster, as the mask is read in from file. This means that it takes O(1) time. For blending the image, which is done regardless of which mode is chosen, that takes O(height*width) time, as it processes the output image one pixel at a time. Given the sample images are square, this is $O(n^2)$

Overall, the light leak takes $O(2 * height * 2 * width * channels)$ time, or for the sample images (given they're square) $O(n^2)$. The rainbow light leak takes O(height*width) to complete, for the sample images $O(n^2)$.

Problem 2

Results

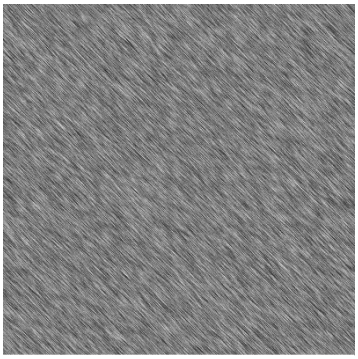
i) Testing

Below are images with differing blending coefficient to test that the parameter works as intended.

0.25

0.5

0.75



1

ii) Reasoning

For creating the noise masks, I decided to create a mask of 0s and then randomly assign values in the range of 0,0.6 to all the pixels in the mask. This is because I wanted the brush strokes to seem less uniform and more random. For the motion blur, the direction is different for both custom noise textures so they differ from each other when applied to the picture, with one being diagonal and the other horizontal.

iii) Computational Complexity

To create the custom noise textures I set the value pixel by pixel so $O(height * width)$, or for the square input images $O(n^2)$. Generating the noise textures is done once for the monochrome effect, and twice for the coloured pencil effect. The remaining steps, creating the motion blur and then merging the images is done in O(1) time. As a result the monochrome takes O(height*width) or $O(n^2)$, and the coloured pencil takes $O(2height * 2width)$ or $O(n^2)$.

Problem 3

Results

i) Testing

Once again, I tested out different parameters to make sure they function as intended.
Below are images where the size of the neighbourhood and sigma used in the gaussian blur are altered. The images are altered but its hard to demonstrate due to the blurring being solved when the image is shrunk.

Size = 5, Sigma = 0.8

Size = 15, Sigma = 0.8

Size = 5, Sigma = 20

Size = 5, Sigma = 40

Below are images where the brightening (Br) and darkening (Dr) coefficients have been changed. These alter the LUTs that are used to beautify the image.

Br = 1, Dr = 0.9

Br = 5, Dr = 0.9

Br = 1.5, Dr = 0.5

Br = 1.5, Dr = 0.9

ii) Reasoning

To apply the Gaussian blur to colour input I process it on each of the 3 channels. Then to beautify the image I decided to convert the image to HSV. This is because the HSV colour space would allow me to alter the saturation and vibrance of the image to make the image look better without changing the overall colour of the image, unlike altering the RGB space. To beautify the image I decided to increase the saturation of the image and then darken the vibrance in order to balance it out. The image is then converted back to BGR before being displayed and saved.

iii) Computational Complexity

Smoothing the image has a high computational complexity, taking $O(height * width * channels * size^2)$ which for the square input images is $O(n^2)$. Generating the LUTs takes O(1) as it only needs to compute the values for 0,255 to both brightening or darkening. Finally beautifying the image takes $O(height * width)$ as it goes through the image pixel by pixel, once again for the square images this is $O(n^2)$. Overall therefore the filter takes $O(n^2)$.

iv) Requested Discussion

I decided to use a Gaussian blur for the smoothing filter because the problem with Gaussian smoothing is that it can cause blurring when removing noise, which is the intended purpose of the filter. I set the size of the neighbourhood evaluated to a 5x5, this is because a higher size would increase computational complexity and make it harder to test combinations of parameters. For the sigma value I chose 0.8 as I only wanted a subtle blurring and a larger sigma would've caused increased blurring

Problem 4

Results

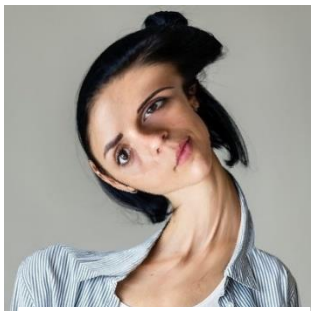
Swirled Image

Blurred Image

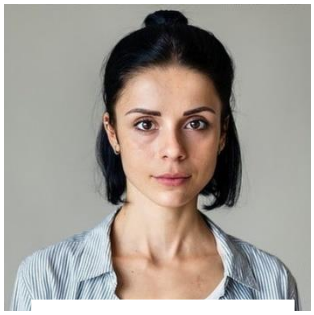
Swirled Blurred

Unswirled Image

Difference

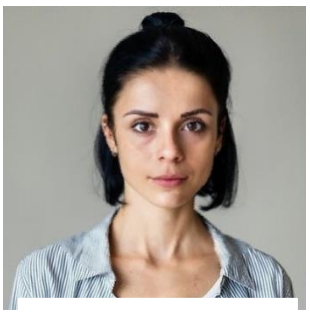


Swirled Image

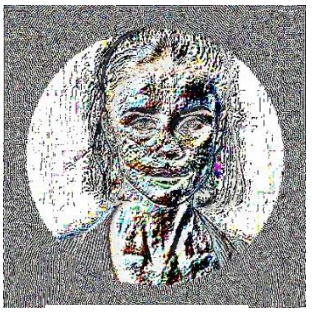


Blurred Image

Swirled Blurred



Unswirled Image



Difference

i) Testing

Below are images with differing parameters to check that the parameters change the image as expected.

R = 150, Theta = pi/2

R = 150, Theta = 0.5

R = 50, Theta = pi/2

Size = 60

Size = 20

ii) Reasoning

For the low pass filter, I decided to convert the image into the fourier space and the use a circular ideal low pass filter. This is because upon testing the better-looking images didn't include any ringing, so I didn't have to account for them using Butterworth or Gaussian low pass filtering. For the geometric filter I decided to use a variable swirl amount, this is so the swirl gets more intense towards the centre rather than being uniform throughout. To inverse the swirl I decided to do the same transform but with a negative theta. Both interpolation strategies are present in the code and can be altered using the mode parameter.

iii) Computational Complexity

The low pass filter takes $O(height * width)$ as it passes over the entire mask (which is the same size as the input image) and changes the values in the correct region to 1. For the square images this is $O(n^2)$. Transforming the image takes $O(height * width * channels)$ as the transform must occur on every pixel in every channel. This is $O(n^2)$ if the image is square (like the sample images). The complexity to get the inverse of a swirl is identical, as the same function is run just with negative theta. Overall, to blur, swirl then unswirl the images it takes $O(3 * height * 3 * width * 2 * channels)$ or more precisely for the input images $O(n^2)$.

iv) Requested Discussion

The blurring of the image before swirling helps to remove certain aliasing artifacts. This is most notable example of this occuring is in the hair. Looking at the two images below, you can see the blurred image on the right has less aliasing effects and makes the hair look more smooth and artificial.

The difference between the source image and the unswirled image is evident when subtracting the images. The difference outside of the swirl radius are as a result of the blurring done on the image before swirling. The main difference is within the swirl radius where because of interpolation the pixels have not necessarily returned to the same position as the source image.