DIGITAL IMAGE FORMATION AND ENHANCEMENT

A presentation by **Hans Emmanuel Hernandez** (2020-11387)

Objectives

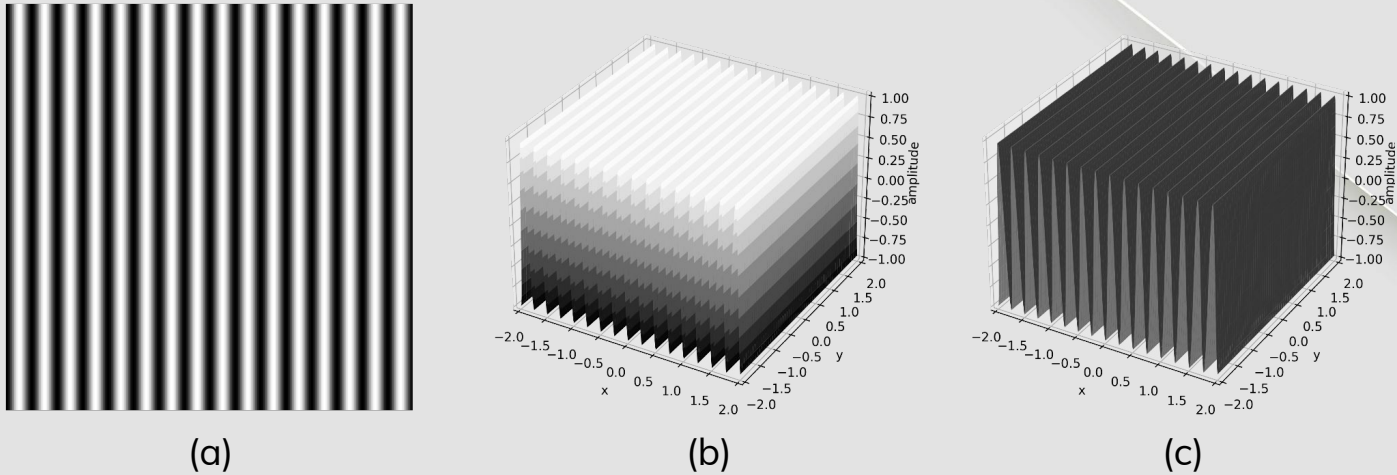
In this activity, we implement multiple image processing techniques that aid analysis and enhancement. We have the following objectives:

- Simulate 2D and 3D shapes, objects, and various images mathematically using python
- Observe the differences between image file formats such as JPG, PNG, BMP, and TIFF, and their corresponding uses in image processing after compression
- Enhance a low-light image through histogram backprojection using different CDFs such as linear and sigmoid
- Apply different enhancement techniques such as contrast stretching, gray world algorithm, and white patch algorithm to faded, colored photographs

Results and Analysis

In this part, we present the results, discuss their significance in line with the objectives, and analyze accordingly.

Sinusoid along the x-direction

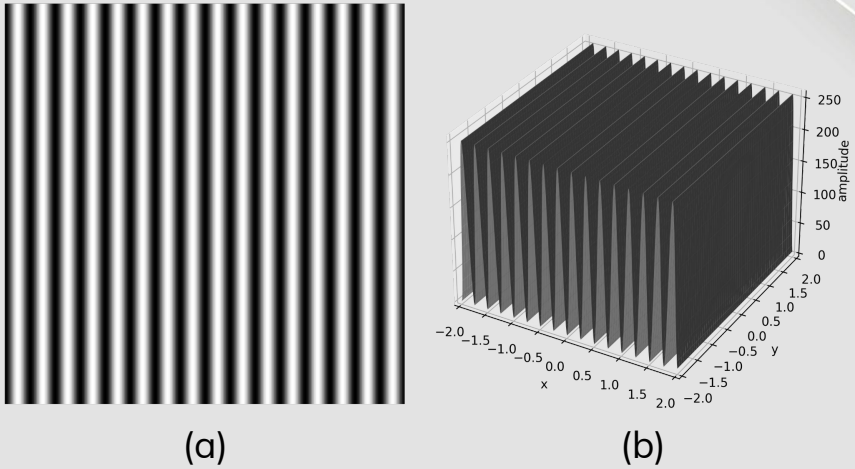


(a) (b) (c)

Fig. 1 Mathematical spatial simulation of a sinusoidal wave given by $A\sin(4\pi fx)$ propagating 4 cycles per *cm* along the x-axis. (a) shows its *2D* grayscale projection, (b) shows its *3D* grayscale projection, and (c) shows its *3D* spatial profile.

As we can see from the figure on the left, the 2D projection of the sinusoidal wave shows **a blurry alternating pattern of black and white stripes.** We can visualize this further by examining its 3D grayscale projection at the center image. If we view it from above, we would see that **the global maximum amplitude of the wave represents the white stripes while the global minimum is represented by the black stripes.** This is accurate for grayscale values since **the highest value (255) is white while the lowest value (0) is black.** The rest of **the values in between represent a range for the shades of gray which can be observed by the 'blurring' of the stripes** from the center outwards.

Converting the intensity of the sinusoid to uint8



(a) (b)

Fig. 2 Mathematical spatial simulation of a sinusoidal wave containing uint8 data type after normalization and mapping amplitude intensities to [0, 255].

As we have shown in the previous slide, **the amplitude intensity consisted of negative values ranging from [-1.00, 1.00].** This is problematic in terms of physical images so we have to convert them into their corresponding grayscale values. **Since our data type is float64, we simply convert it to uint8 which accommodates integers from [0, 255].** This can be done by first normalizing the amplitude array [-1.00, 1.00] by a normalization factor of 255 then mapping the scaled values to [0, 255]. Finally, we convert the data type to uint8 since our new values are now appropriate. Again, this is accurate with the grayscale values as discussed in the previous slide. **The lowest value (0) represents black while the highest value (255) represents white.**

Grating with square waves

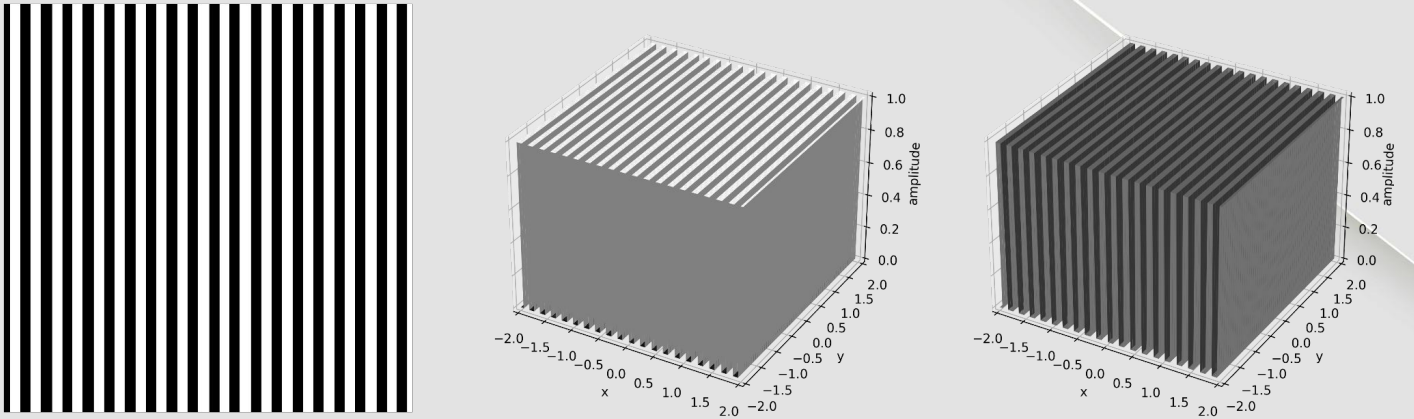
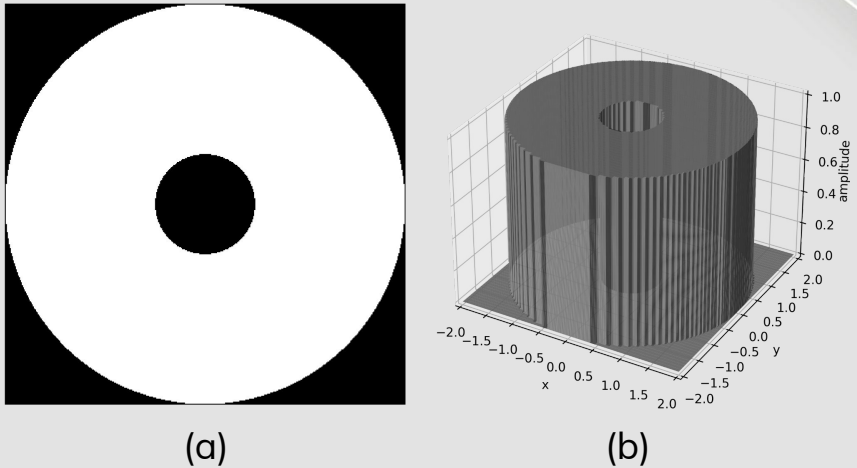


Fig. 3 Mathematical spatial simulation of a square wave given by a rounded sinusoid function propagating 5 line pairs per *cm* along the x-axis. (a) shows its *2D* grayscale projection, (b) shows its *3D* grayscale projection, and (c) shows its *3D* spatial profile.

As we can see from the figure on the left, the 2D projection of the square wave shows **a solid alternating pattern of black and white stripes known as grating.** Unlike in the sinusoidal wave from before, **the luminance of the square waves create sharp edges because its maximum and minimum peaks are flattened.** There is a high contrast between the global maximum amplitude and the global minimum since the values of the wave are rounded, thus, **the wave instantaneously jumps from its lowest to its highest peak without disturbance.** Again, this can be visualized further by observing (b) from above and comparing it with its 2D projection in (a).

Hubble’s primary mirror simulation

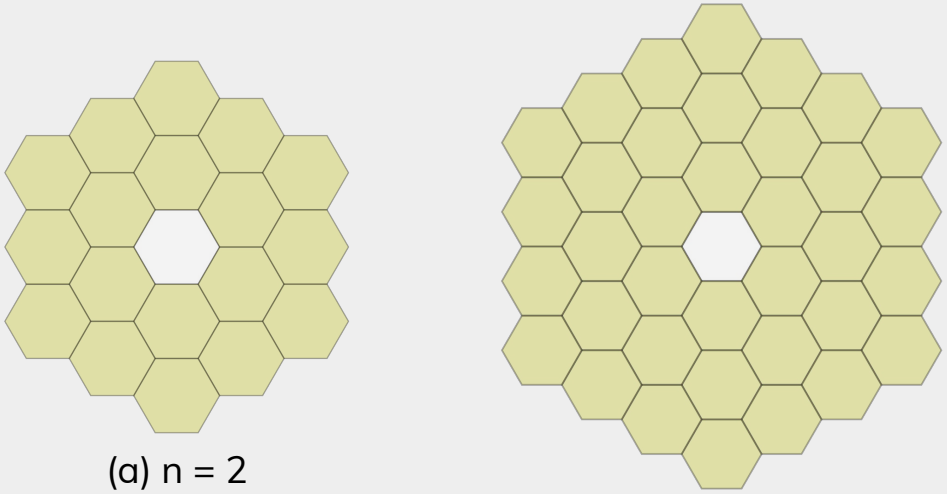


(a) (b)

Fig. 4 Mathematical spatial simulation of Hubble’s primary mirror given by an annulus function. (a) shows its 2D grayscale projection while (b) shows its 3D profile.

As we can see from above, the 2D simulation of Hubble’s primary mirror resembles that of an annulus. This is generated by **a simple circle function wherein the amplitude of the inner radius is set to 0 while the rest of the body is set to 1.** Again, this is accurate with its corresponding grayscale values since the range goes from black to white.

JWST primary mirror simulation



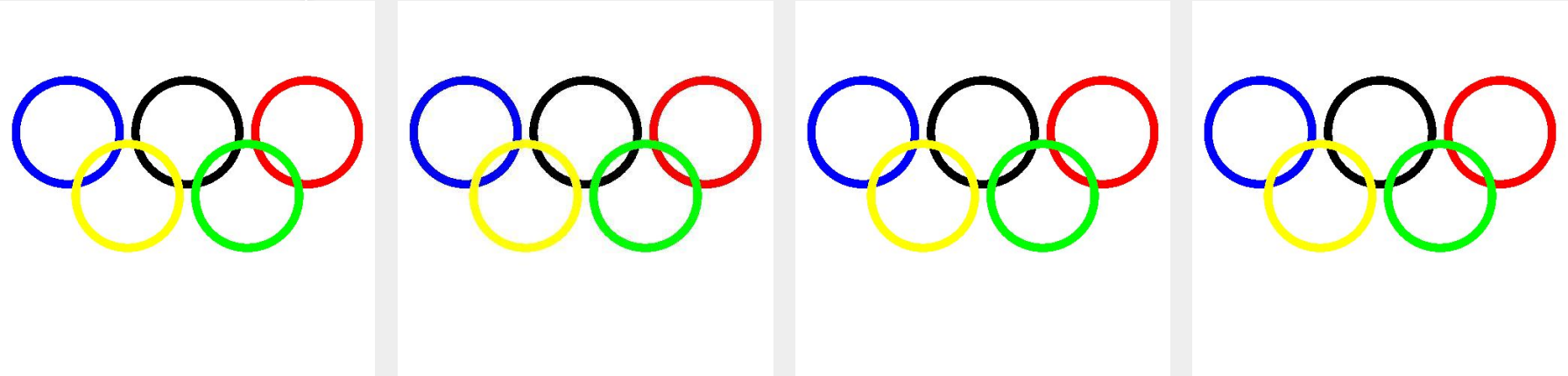
(a) n = 2

(b) n = 3

Fig. 5 Mathematical spatial simulation of the JWST primary mirror given by a hexagonal array function. (a) shows its 2D RGB projection with n = 2 layers while (b) shows n = 3 layers

As we can see from above, **the JWST primary simulation is a series of stacked hexagonal arrays where n is the number of layers.** To construct the hexagons, we use a a modified formula for the solving the apothem (line segment starting from the center that is perpendicular to the side). We combine this with a specified horizontal coordinate so that the algorithm determines the center of a singular hexagon shape. Each hexagon is a colored ‘patch’. **Further layers may be stacked for whatever purpose.**

Color image

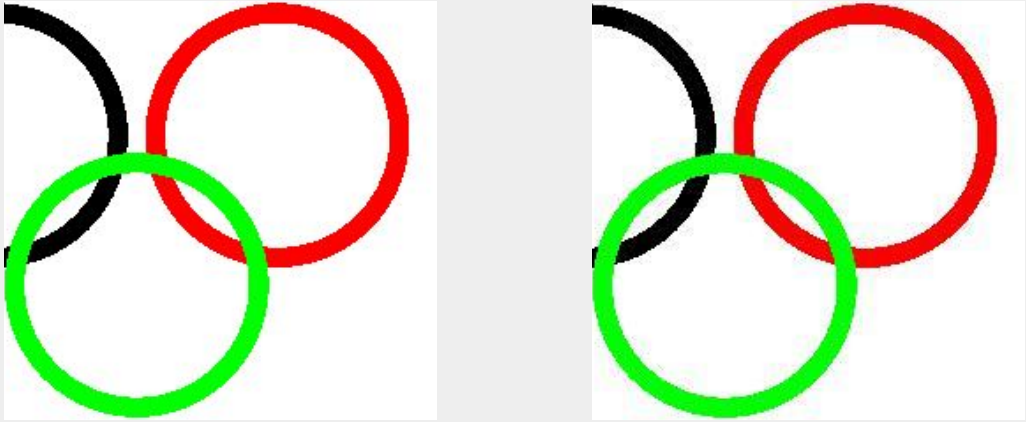


(a) PNG (b) JPG (c) TIFF (d) BMP

Fig. 6 Mathematical spatial simulation of the Olympic logo using RGB layering. (a) shows exported PNG format, (b) shows exported JPG format, (c) shows exported TIFF format, and (d) shows exported BMP format.

To construct the circles above, **we manipulate the same circular function as before by translating the circles into their appropriate positions on the plot.** To set the amount of radius we want to fill in with color, we simply adjust the radius condition within the function. This allows us to make the 'ring' as thick or as thin as we want. **To set the color, we simply treat the presence of each color array in the RGB as 1.** If we want to make the color blue, we set R = 0, G = 0, and B = 1. Another example is for black wherein we set every color array to 0 since **black has an RGB value of (0,0,0) which is the opposite of white (1, 1, 1).**

Comparing image file formats

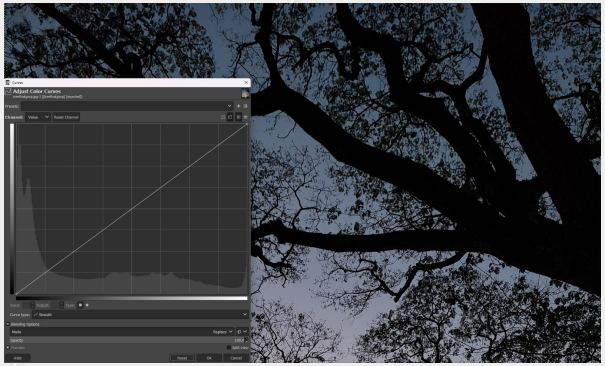


(a) PNG (b) JPG

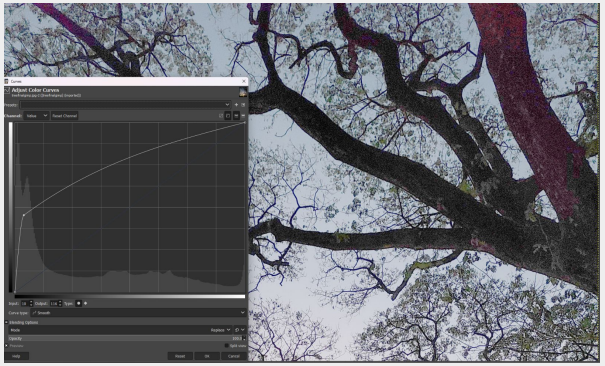
Fig. 7 Zoomed in PNG format vs. zoomed in JPG format

From the previous slide, **the difference between PNG, TIFF, and BMP formats were miniscule** and were not really visible especially since the image is not that detailed to begin with. However, **comparing a zoomed in PNG format with a zoomed in JPG format of the olympic logo clearly illustrates the disparity in image quality.** From above, **the JPG format is clearly more blurred with noisier edges around the rings.** The 'shadows' generated by the green ring is also heavily smudged around the red ring as seen on the JPG format. This is because **JPG uses a method of lossy compression which allows for easier and faster image sharing unlike PNG which uses a method of lossless image compression for better resolution retention.**

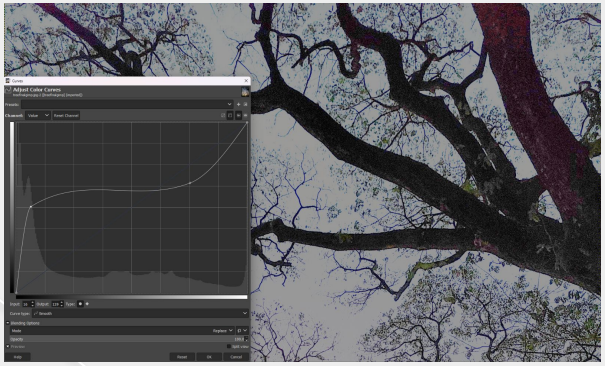
Altering the input-output curve



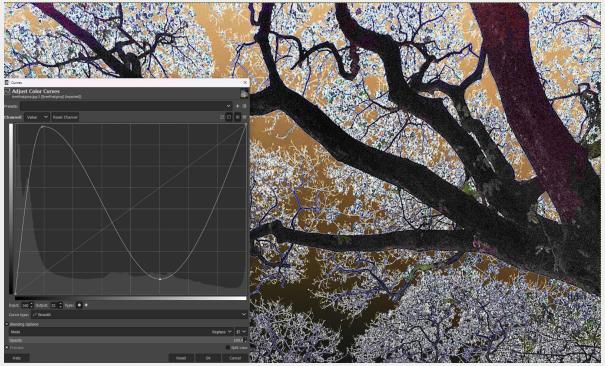
(a) original



(b) curve 1



(c) curve 2



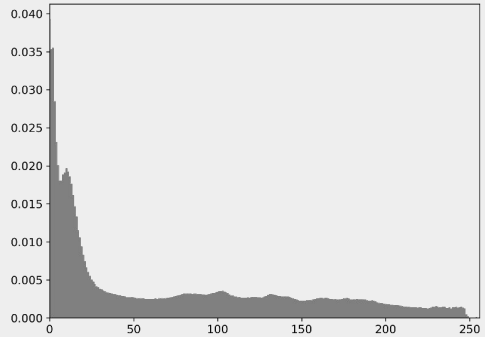
(d) curve 3

Fig. 8 Input-output curve manipulation results in revealing various details of the image

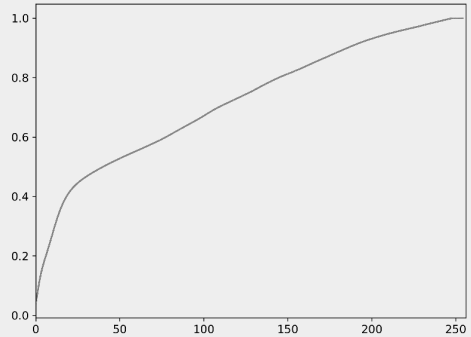
Using the GIMP software, a dark image of tree branches was enhanced to reveal the details hidden by poor lighting. **By manipulating the input-output (i-o) curve of the image, we can adjust the brightness of each corresponding pixel in the value histogram.** (b) shows that **by adjusting the left part of the i-o curve, the dark parts of the image are brightened** and thus, reveal more details. This is because **a majority of the image's pixels are dark as seen by the distribution of the histogram** which is evidently saturated at the left tail. This means that **altering the i-o curve at this part would also alter the dark pixels of the image.** The same can be said for (d) wherein **both dark and brighter pixels are altered simultaneously.**

Histogram Backprojection on grayscale images

Fig. 9 Grayscale of the original image



(a) Normalized PDF of the grayscale image



(b) Normalized CDF of the grayscale image

Fig. 10 Distribution functions of the grayscale image

From (a), we see that **most of the image's pixels have a lower grayscale value, suggesting that we are dealing with a significantly dark image.** This is further validated by (b) which shows that **about 60% of the pixels have a grayscale value that is lower than or equal to 100.** However, a fair amount of pixels still do seem to assume the upper grayscale values. This is not surprising since **the grayscale image does contain brighter whites** from the series of lampposts below, together with the somewhat bright sky.

Ideal CDF: Linear

(a) Grayscale of the original image

(b) Histogram equalized image using linear CDF

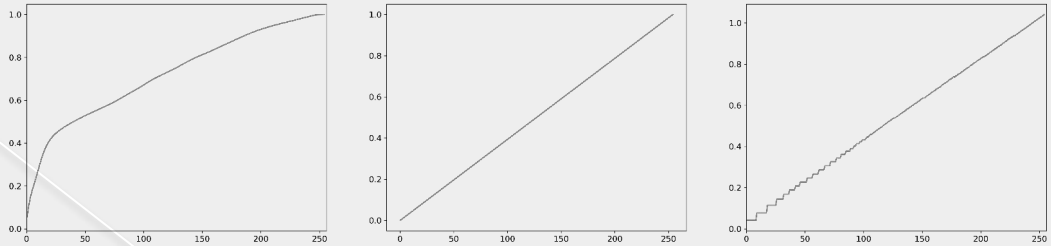
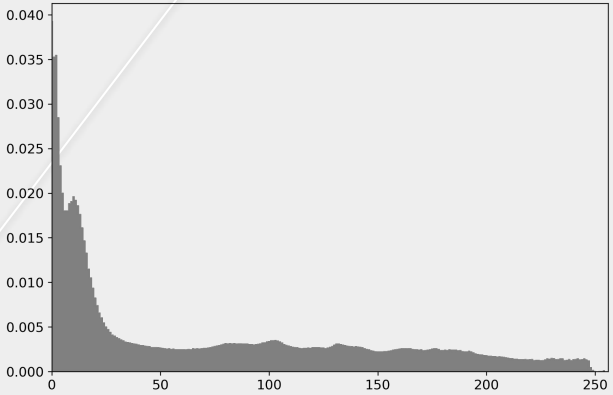


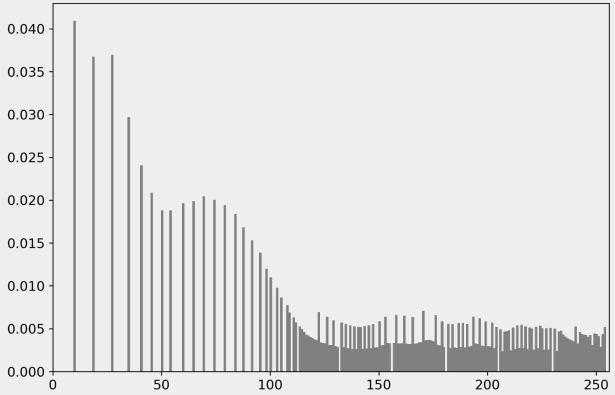
Fig. 11 Image enhancement using histogram backprojection with a linear CDF. *From left to right: original CDF, desired linear CDF, and CDF of new image.*

After applying a uniform CDF given by a linear function, we analyze the resulting backprojected image (b) on the left. Looking at its resulting CDF below, we can see that **we have successfully altered the original CDF into having a more linear trend.** We can physically observe this through the resulting backprojected image which clearly shows a significant amount of detail over its previous iteration. **Various leaves, twigs, and branch profiles are now visible in the resulting image whereas the original grayscale image only seemed to show a solid black color over the entire tree.** Moreover, we can even observe **multiple signages at the lower part** that were clearly obscured by darkness before. We can also **grasp the depth between the branching trees** as opposed to the original wherein most of the lower part just seemed to combine into a singular black color.

Ideal CDF: Linear



(a) Normalized PDF of the grayscale image



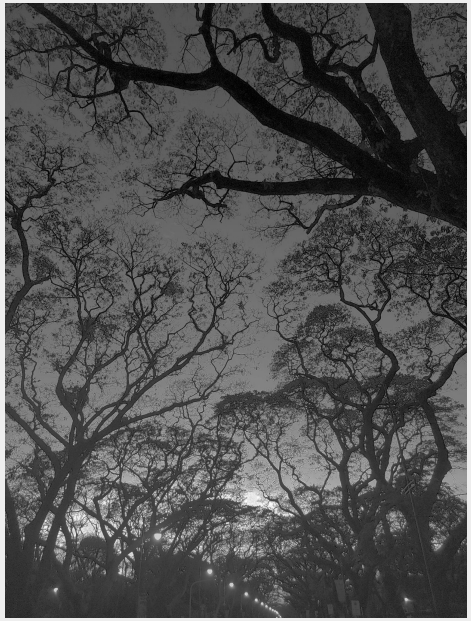
(b) New PDF of the histogram equalized image

Fig. 12 Old PDF of the grayscale image vs. its PDF after histogram equalization using a linear CDF

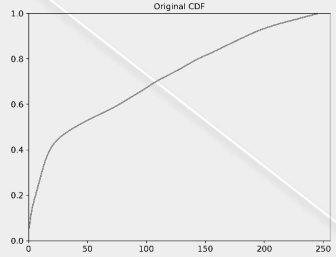
After observing the histogram equalized image, we get its updated PDF and compare it to its former. As we can see from above, **(b) shows a stretched histogram that occupies the entire spectrum of the grayscale values.** This is to be expected since our resulting image is brighter and thus, have significantly increased pixel intensities occupying the upper range.

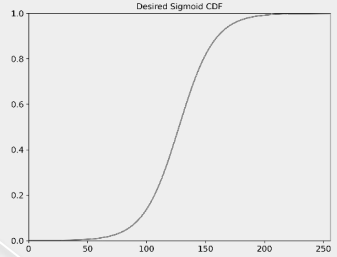
Ideal CDF: Sigmoid

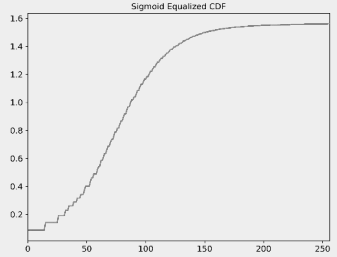
(a) Grayscale of the original image



(b) Histogram equalized image using sigmoid CDF



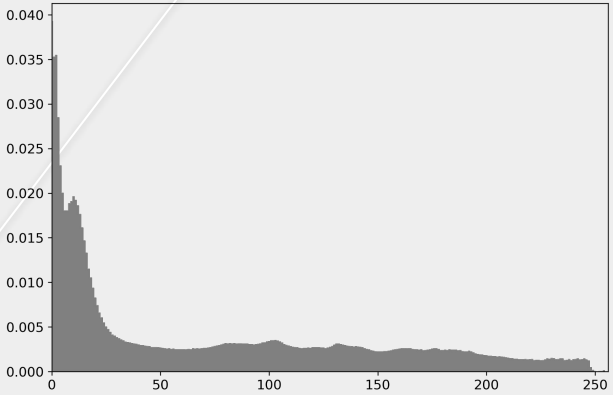




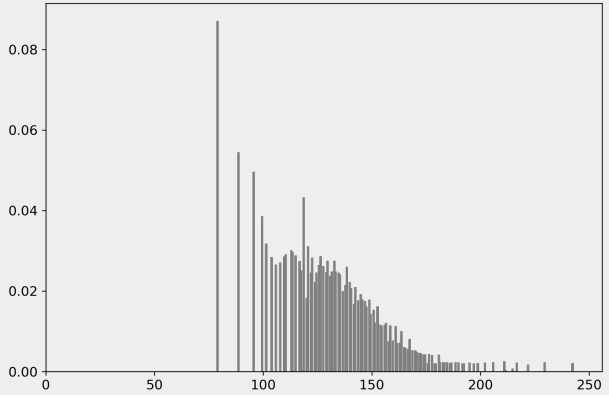
Next, we perform the same algorithm on the grayscale image using a sigmoid function for the CDF. As we can see on the graphs below, **the sigmoid function is an s-shaped curve that maps out the pixel intensities between 0 and 1.** It is characterized by **a steep inflection point** right in the middle of the grayscale values. This suggests that **we are trying to shift the intensities of our original CDF to have grayscale values that are within its median.** This is accurate with the resulting image after equalizing since the background is mostly gray. However, it is evident that **much of the details are harder to see since our original image is very dark and has the highest saturation of pixels at the lower bound of the grayscale values.** Unlike the dramatic enhancement of the linear CDF, **the sigmoid CDF is closer to the human eye response especially in low-light conditions due to its nonlinearity.** Thus, we are more likely to observe the same kind of quality in nature.

Fig. 13 Image enhancement using histogram backprojection with a sigmoid CDF. *From left to right: original CDF, desired sigmoid CDF, and CDF of new image.*

Ideal CDF: Sigmoid



(a) Normalized PDF of the grayscale image



(b) New PDF of the histogram equalized image

Fig. 14 Old PDF of the grayscale image vs. its PDF after histogram equalization using a sigmoid CDF

After observing the histogram equalized image, we get its updated PDF and compare it to its former. As we can see from above, **(b) shows a stretched histogram that mostly occupies the central spectrum of the grayscale values.** This is to be expected since **our resulting image has a wider range of grays which is accurate with the steep inflection right in the middle of the sigmoid curve.** Again, since the original PDF has the highest saturation at the lower grayscale values, this high peak would just shift right about the center of the grayscale values, resulting to a significantly dimmer image than the linear CDF.

Contrast enhancement

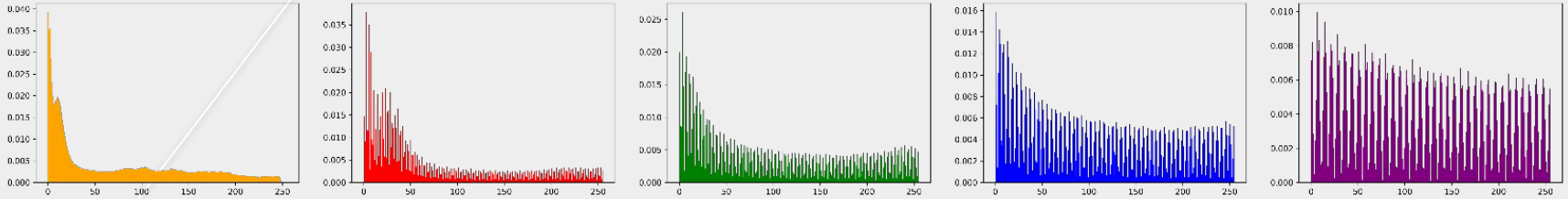


Fig. 15 New PDFs of contrast stretched images. Grayscale percentiles from left to right: [0,255], [1,99], [15,85], [20,75], [30,65]

After applying the formula for contrast stretching on the various grayscale percentiles of the image, we obtain the following distributions. However, **since the dynamic range of our original grayscale image already spans the entire grayscale range such that its minimum value is zero and its maximum value is 255, the old values are just retained.** Looking at the different percentiles for contrast stretching, we indeed confirm the fact that **contrast stretching just linearly maps or 'stretches' the input values to the entire grayscale range.** That is **mapping the minimum input to zero and the maximum input to 255.** As opposed to histogram equalization, **contrast stretching the image did not yield any visible result apart from its computational aspect.** Again, this is to be expected since the original image already contains the entire dynamic range of grayscale values.

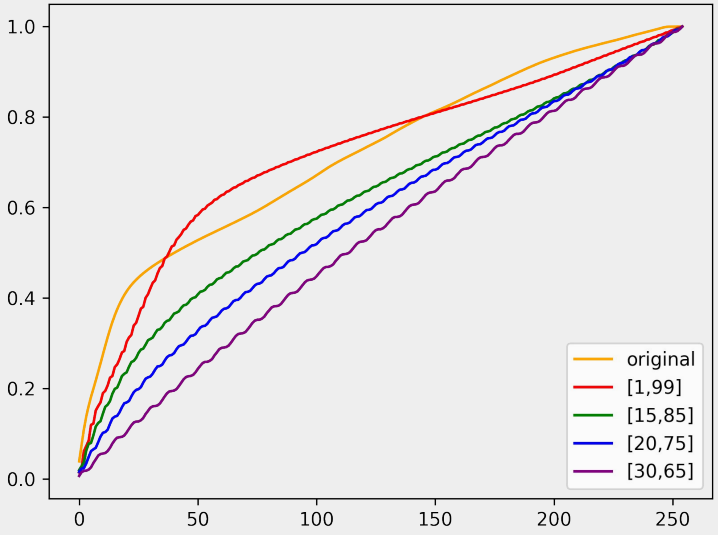


Fig. 16 New CDFs of the contrast stretched images

Restoring faded colored photographs

Fig. 17 Unrestored faded colored photograph from [6]

(a) Red channel sum: 87997308

(b) Green channel sum: 70504326

(c) Blue channel sum: 65986291

Fig. 18 RGB channels of the faded image color-mapped into gray with their corresponding summated values

Using the the faded image from [6], we apply various white balancing algorithms and analyze their effects on the overall quality of the picture. Fig. 18 shows the resulting decompositions of each individual RGB channels after color-mapping them into gray. **The graylevel in each image indicates the brightness of their corresponding color channel upon reconstructing the original image.** This essentially tells us that **red is the dominant color on the faded photograph since it has the largest graylevel sum out of the three.** It also follows that **blue is the least dominant since it evidently has the least sum out of all.**

Restoring faded colored photographs

(a) Original

(b) Contrast stretched

(c) Gray world algorithm

(d) White patch algorithm

Fig. 19 White balancing algorithms for image restoration

From the previous slide, we have established that red was the dominant color on (a). However, after applying various white balancing algorithms, we have most visibly altered and 'lifted' the intensity of the color red on the resulting pictures in Fig. 19. **Contrast stretching the image 'stabilized' the RGB values by lowering their respective intensities (result of the stretch), resulting to a more balanced image overall** despite still being predominantly red. **Applying the gray world algorithm, we see a more dramatic change in (c) since the intensity of each RGB channel is now equalized at about ~74500000 each after summation.** However, the resulting image now clearly has a dominant greenish and bluish tint all over. **This is a very common overshooting problem of the gray world algorithm since it averages the RGB channels wherein red is very high in our case while blue is significantly lower, resulting to overcorrection of the pixel intensities.** Finally, we move on to the best looking image in (d) from a personal standpoint. Applying the white patch algorithm, **the RGB channels were also equalized but unlike the gray world algorithm in (c) which dramatically equalized them, the red channel still retained the highest summated value which was closely followed by the blue and green channels.** This could be explained by how the algorithm used the patch as a reference for white in the context of the faded photograph, thus resulting to a more accurate shade of white overall.

Reflection

Besides submitting late, activity 1 is one of the most fun I have ever had in coding! I learned a lot about the fundamentals of image processing: computational representations of images, decomposing and reconstructing colors, mapping pixels, and so much more. Honestly, I do not think I would have learned this much or would have gone above and beyond if I did not take my time in learning the basics (although much of that time could be attributed to something else entirely which is my ADHD). I swear I really did try my best! It was just really hard getting my prescription and medication from the province especially since my psychiatrist is 4 hours away from QC :(

However, I believe that my results are accurate and stayed true to the topic, with additional cross references and analyses. I also cross-validated my results a long time ago with my peers and my generous lab instructor. However, if I had even more time to allot, I would have presented more of my test runs with various images under different lighting conditions and color tints. I think that would have been interesting to present.

Overall, this was a great activity!

Self-Grade

Technical Correctness: 35/35

I believe that my results are correct through math, research, and through validation with my peers and with my instructors.

Quality of Presentation: 35/35

I believe that the quality of my powerpoint is up to par while my code is much more sufficient in GitHub. I constructed the figures as instructed, and exported my data accordingly.

Self-Reflection: 30/30

I believe that I have acknowledged and reflected upon the activity well enough. I also have complete citation on the next slide.

Initiative: 10/10

I went above and beyond with my data presentation, and included extra analyses like the additional layers for the hexagonal array function, sigmoid function, and more.

References

[1] *Colorchecker*. MATLAB & Simulink. (n.d.). Retrieved March 23, 2023, from https://www.mathworks.com/help/images/comparison-of-auto-white-balance-algorithms.html

[2] *Displaying a hexagonal grid with matplotlib*. Stack Overflow. (2021, May 17). Retrieved March 9, 2023, from https://stackoverflow.com/questions/67563362/displaying-a-hexagonal-grid-with-matplotlib

[3] Gruppetta, S. (2021, August 30). *2d Fourier transform in python: Create any image using only sine functions*. The Python Coding Book. Retrieved March 6, 2023, from https://thepythoncodingbook.com/2021/08/30/2d-fourier-transform-in-python-and-fourier-synthesis-of-images/

[4] Image processing with python. (n.d.). Retrieved March 9, 2023, from https://datacarpentry.org/image-processing/aio/index.html

[5] Innat, M. (n.d.). *Basic image processing in python - part 1*. Codementor. Retrieved March 6, 2023, from https://www.codementor.io/@innat_2k14/image-data-analysis-using-numpy-opencv-part-1-kfadbafx6

[6] James. (2019, December 8). *Sun damaged photo restoration*. Flashback Photo Co. Retrieved April 1, 2023, from https://flashbackphoto.com.au/sun-damaged-photo-restoration/

[7] Manansala, J. (2021, February 14). *Image processing with python: Color correction using white balancing methods*. Medium. Retrieved April 1, 2023, from https://jmanansala.medium.com/image-processing-with-python-color-correction-using-white-balancing-6c6c749886de