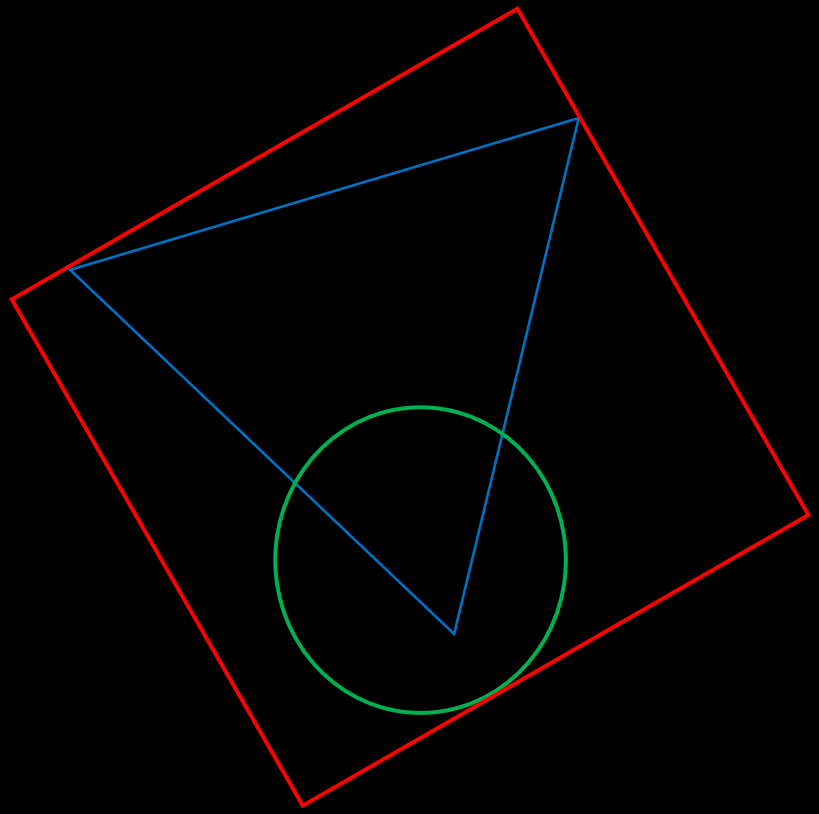
Module 1

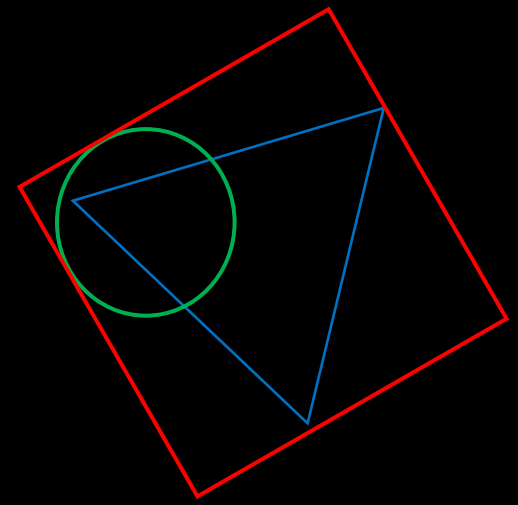
Digital Image Formation and Enhancement

Ron Michael V. Acda
AP 157 WFY-FX2



Objectives

1. To create synthetic images mathematically.
2. To use appropriate file formats for saving images.
3. To open images using Python.
4. To apply backprojection to change the image histogram.
5. To improve the appearance of images.



Results and Discussion

1. To create synthetic images mathematically.
2. To use appropriate file formats for saving images.
3. To open images using Python.
4. To apply backprojection to change the image histogram.
5. To improve the appearance of images.

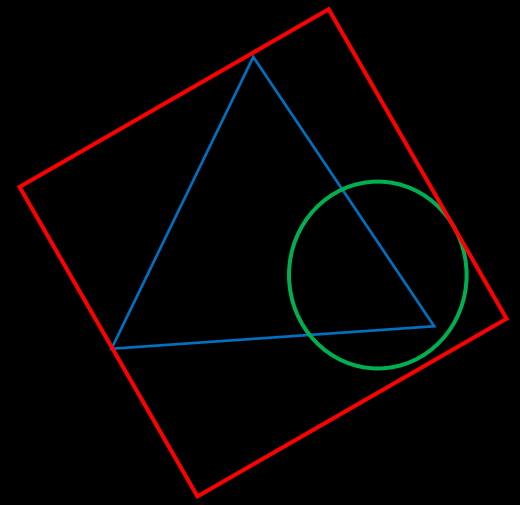


Image DIY

Modules/Libraries Used

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.patches import Polygon
import matplotlib.pyplot as plt
```

For dark backgrounds (optional)

```
plt.style.use('dark_background')
```

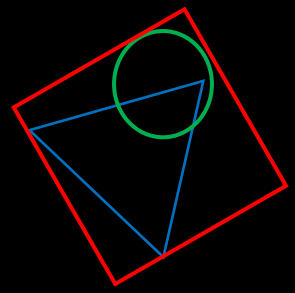
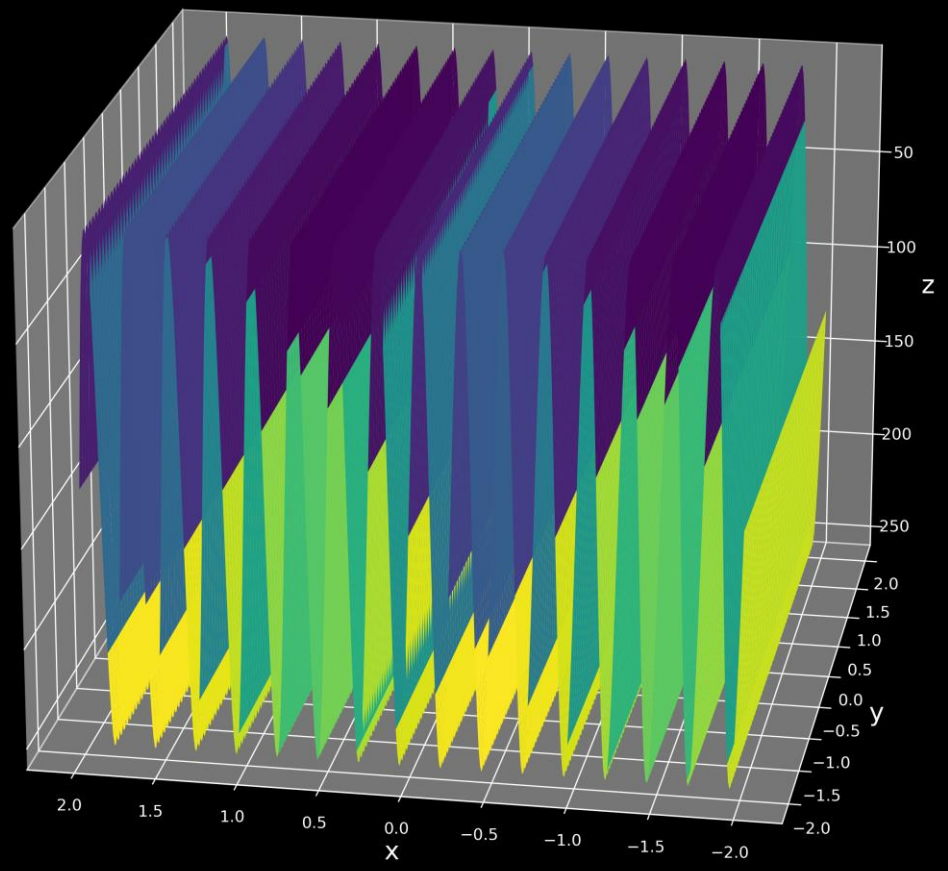


Image DIY




```
N = 500 # The higher N, the finer the resolution
f = 4 # Frequency, cycles per unit.
x = np.linspace(-2,2,num = N)
y = x
X,Y = np.meshgrid(x,y)

omega = 2*np.pi*f #Angular frequency
vals = (np.sin(omega*X)+1)*255/2 #Equation for sinusoid
fig = plt.figure(figsize=(10,10))
ax = Axes3D(fig)
ax.plot_surface(X,Y,vals, cmap='OrRd')
ax.view_init(-160, 80)
ax.set_xlabel('x', fontsize=16)
ax.set_ylabel('y', fontsize=16)
ax.set_zlabel('z', fontsize=16)
```

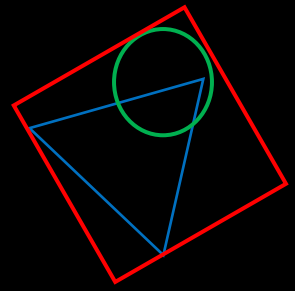
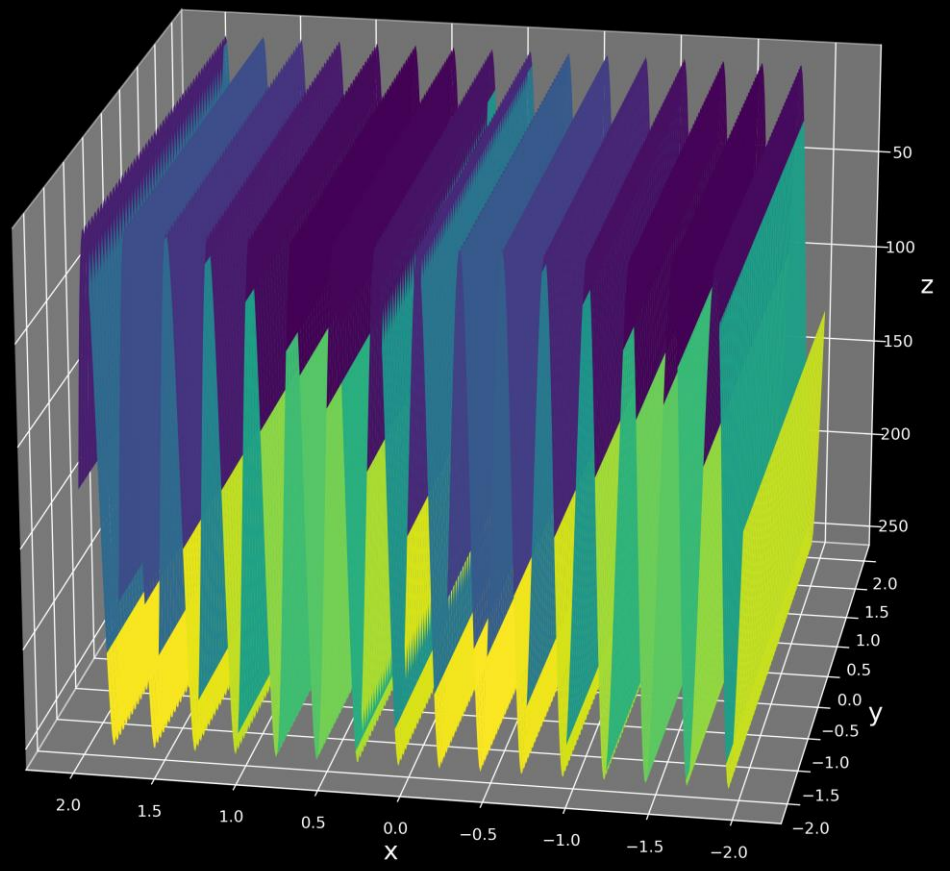
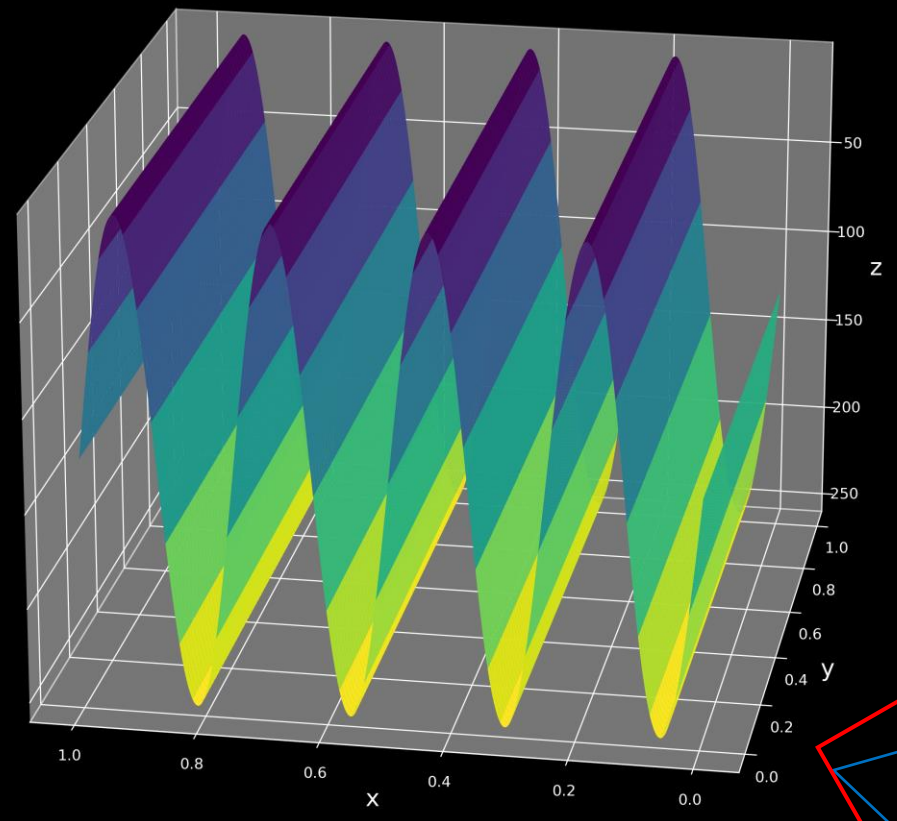


Image DIY

Note that the spacing might be too small for some viewers, so we can change the X and Y range (to zoom in).





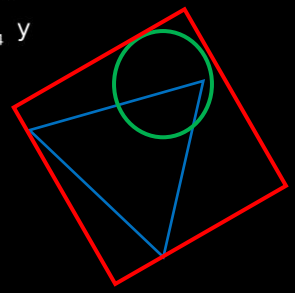
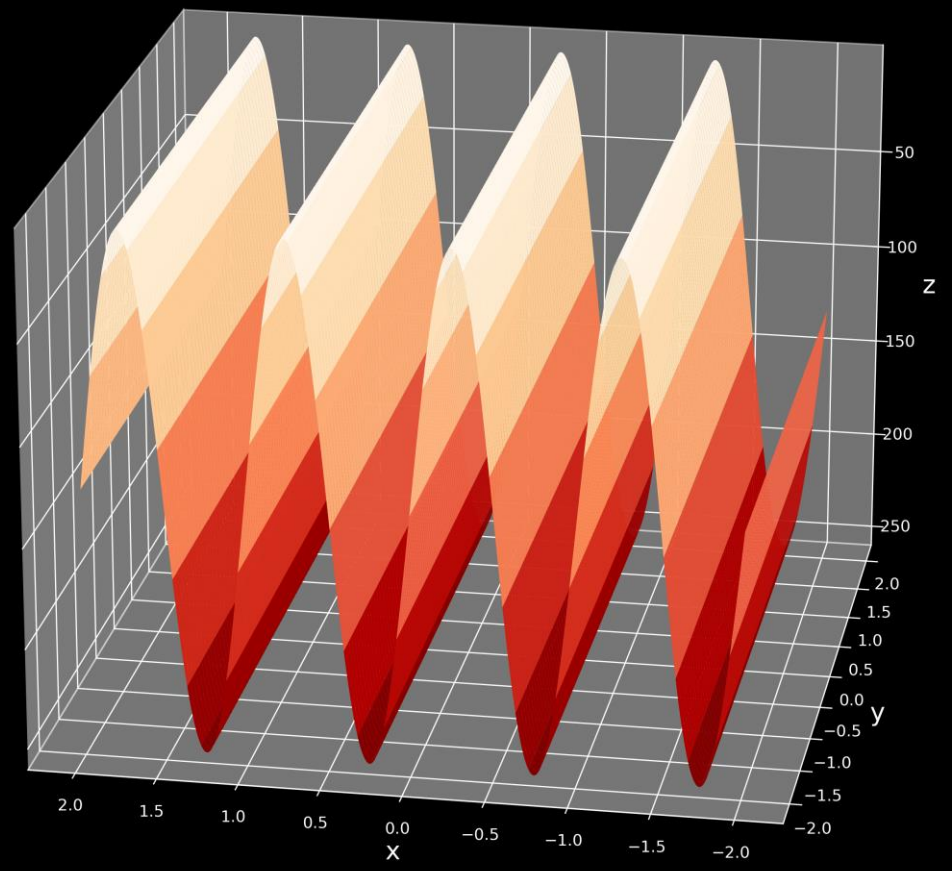


Image DIY

Same sinusoid, different frequency (1 cycle per second instead of 4). Colormap also changed for variety.



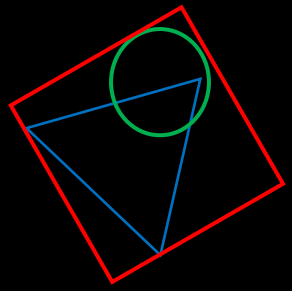
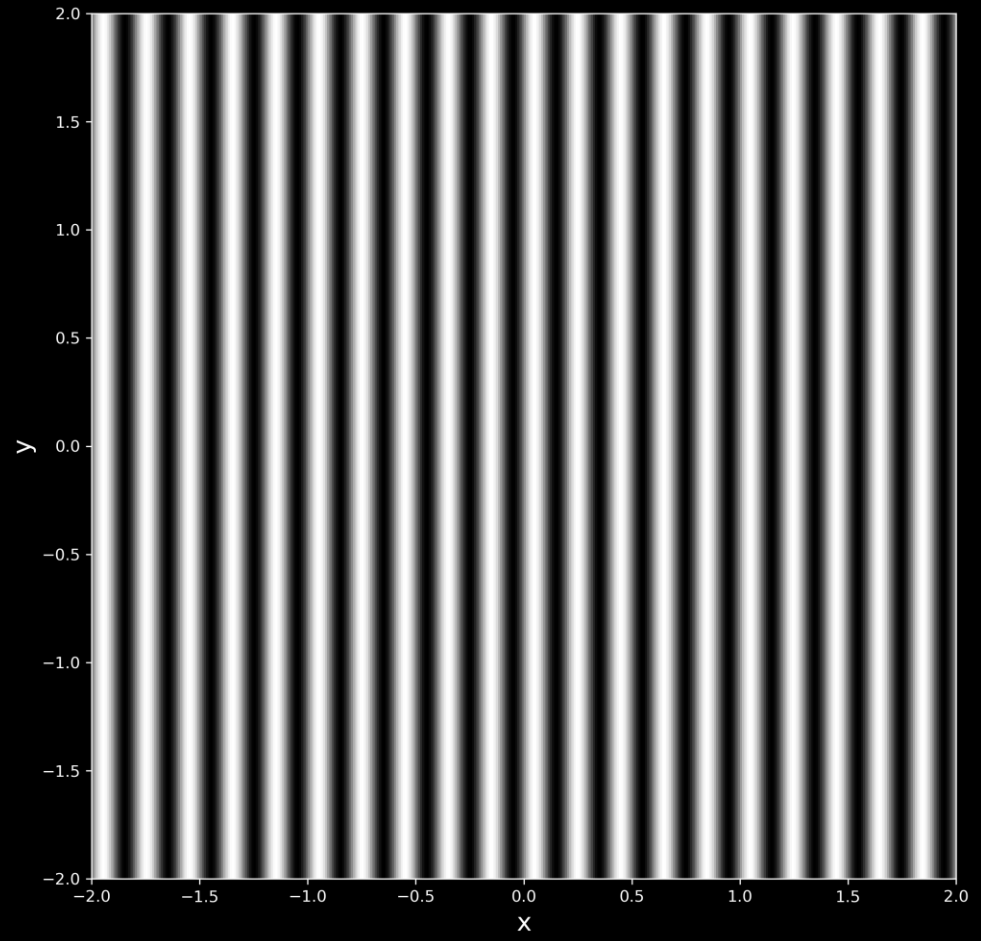


Image DIY

An image of grating lines can be produced by plotting the Z values on the XY meshgrid using a gray colormap. Here, the grating frequency is 5 per unit length.



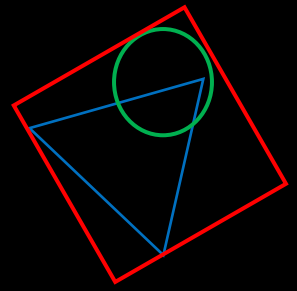
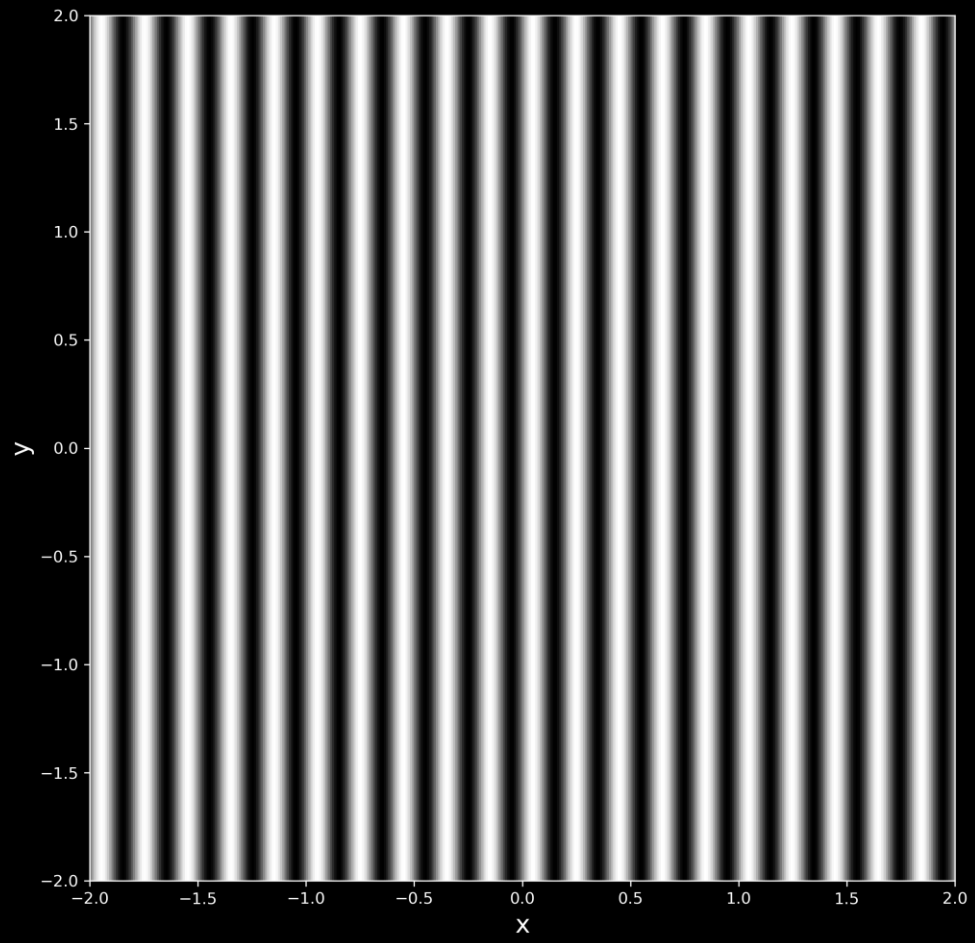
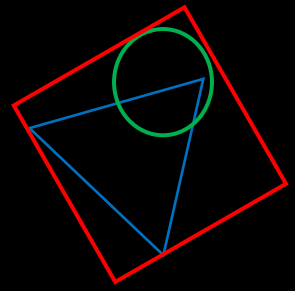


Image DIY




```
fig = plt.figure(figsize=(10,10))
N = 400 # the higher num, the finer
f = 5 # grating frequency
x = np.linspace(-2,2,num = N)
y = x
X,Y = np.meshgrid(x,y)
omega = 2*np.pi*f
vals = 0.5*np.sin(omega*X)
plt.imshow(vals, cmap='gray', extent=[-2,2,-2,2])
plt.xlabel('x', fontsize=16)
plt.ylabel('y', fontsize=16)
plt.savefig('grating.png', dpi=300)
```



Key Concepts Used

To transform a function $f(x)$ with range $[a, b]$ into a new function $\mathbf{g}(x)$ with range $[c, d]$, we apply:

$$g(x) = \frac{1}{b-a}[(d-c)f(x) + (bc - ad)]$$

Proof: Suppose $g(x) = Af(x) + B$. We solve for A, B such that:

$$Aa + B = c$$
$$Ca + B = d$$

Solving for A and B gives the above equation.

Consider $f(x) = \sin(2\pi f x)$. The range is $[-1,1]$. To normalize it to an arbitrary range $[c, d]$,

$$y = \frac{1}{2}[(d-c)\sin(2\pi f x) + (d+c)].$$

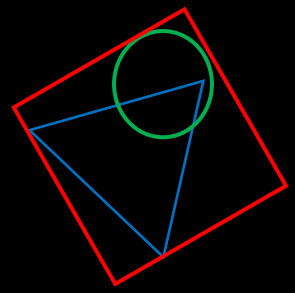
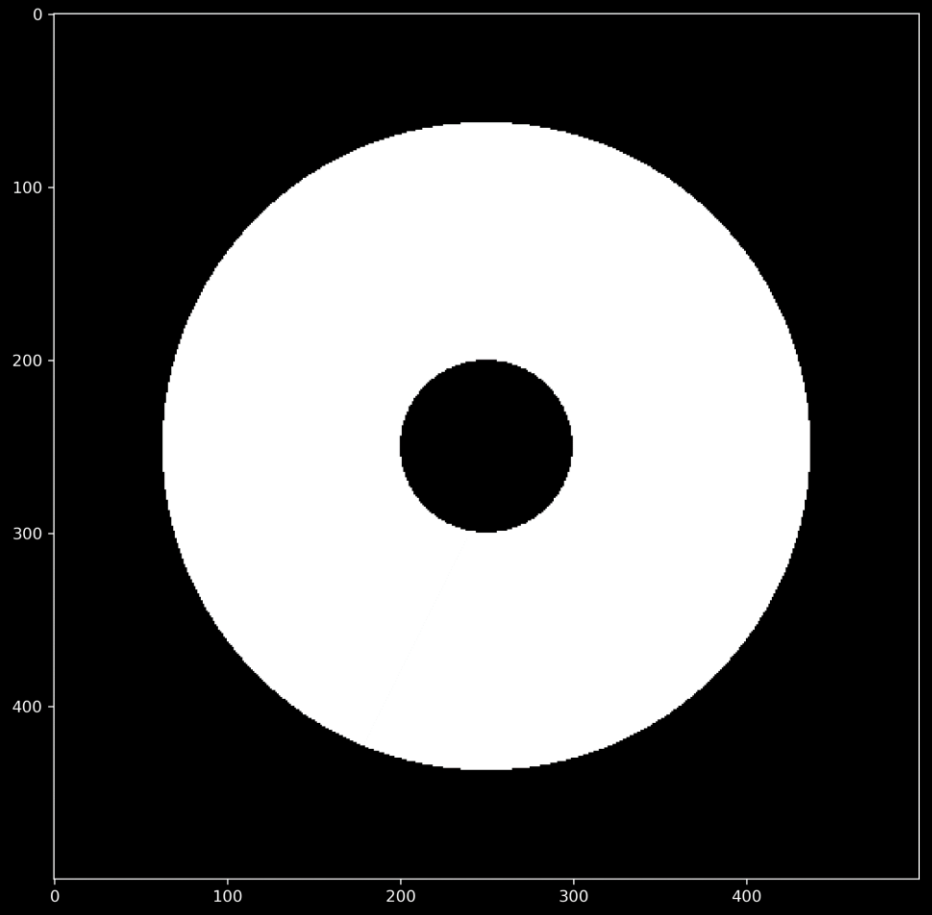


Image DIY



The strategy here is:

1. Create a meshgrid (X,Y).
2. Create an array of zeroes (representing black) with the same size as the meshgrid.
3. Change the value of the array to 1.0 (white) if the indices X,Y satisfy $r_i \leq X^2 + Y^2 \leq r_o$, where r_i and r_o are the inner and outer radius, respectively.

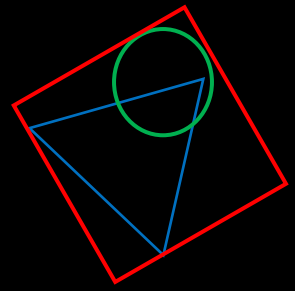
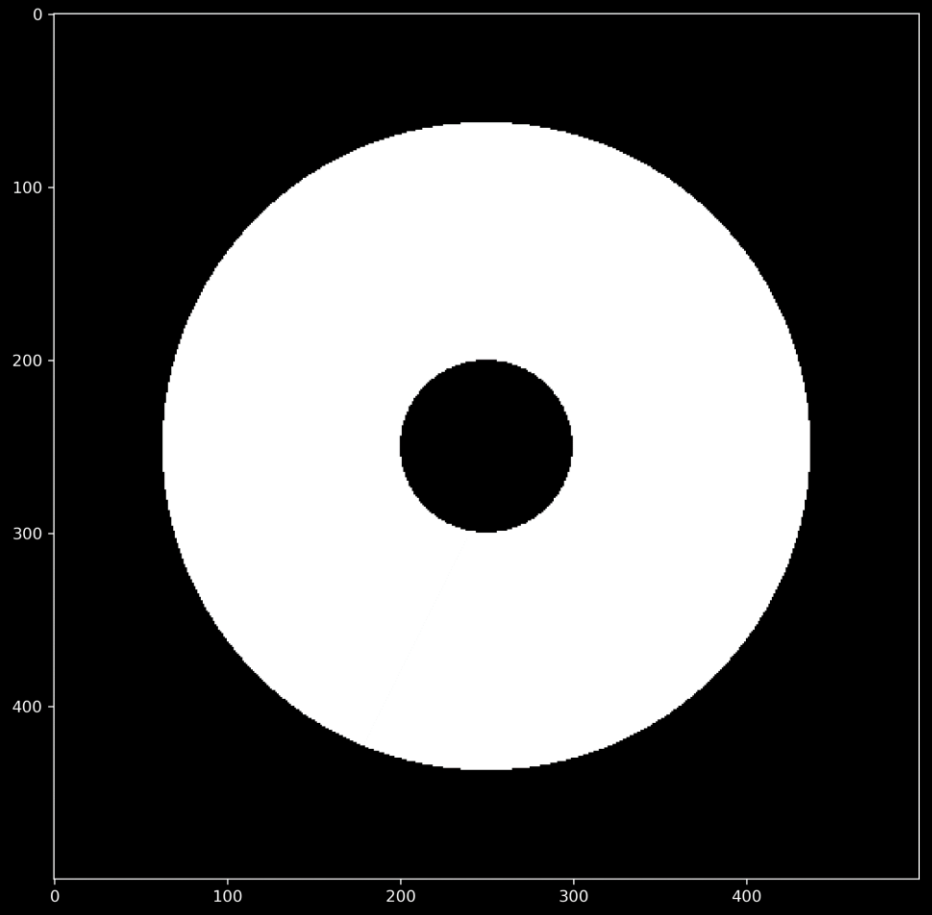


Image DIY




```
N = 500 #the higher num, the finer
x = np.linspace(-1,1,num = N)
y = x
X,Y = np.meshgrid(x,y)
R = np.sqrt(X**2 + Y**2)
A = np.zeros(np.shape(R))

A[np.where(R<0.75)] = 1.0
A[np.where(R<0.2)] = 0.0
plt.figure(figsize=(10,10))
plt.imshow(A, cmap = "gray")
```

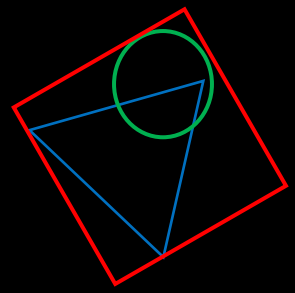
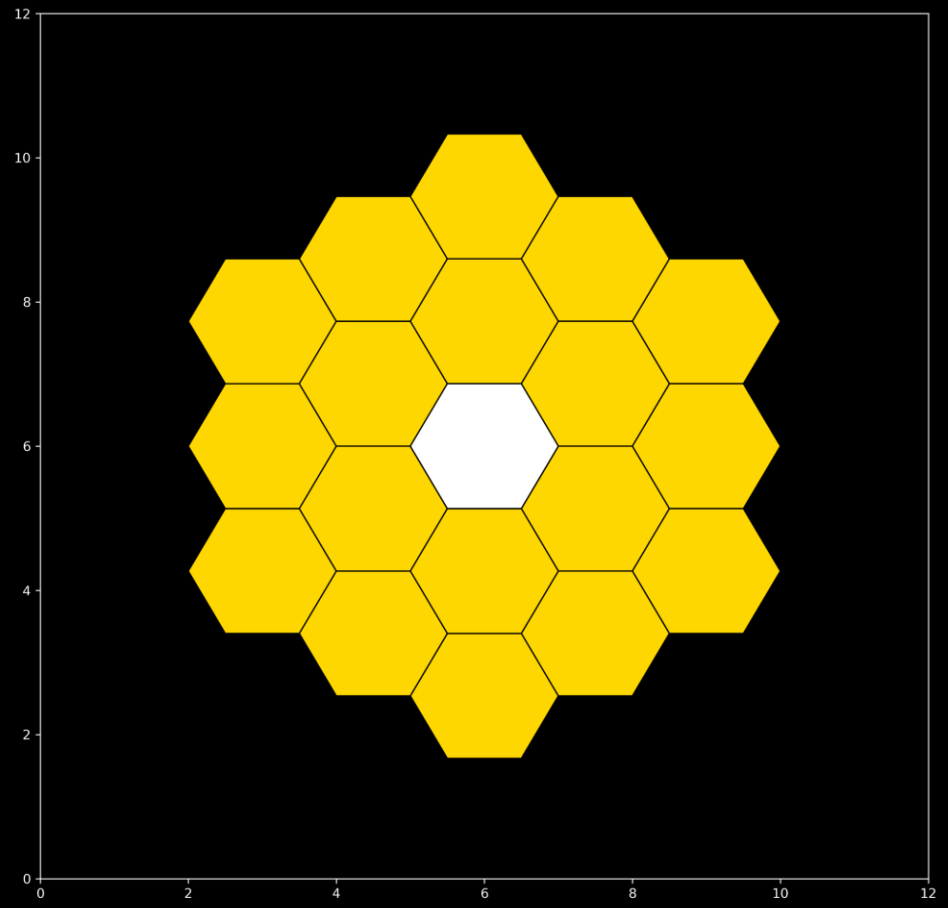


Image DIY (JWST)



Strategy

My plan of attack is to write two functions that do the following:

1. Plot a hexagon patch given the coordinate of the center.
2. Locate the centers of the six adjacent hexagons.

Once these two functions are written, it's a matter of writing a program that uses nested for-loops utilizing the two functions.

TIL: Brute-force Cartesian coordinates!

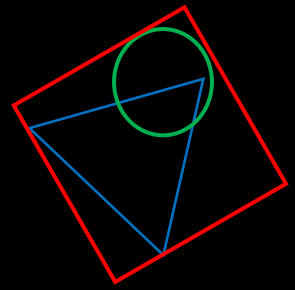
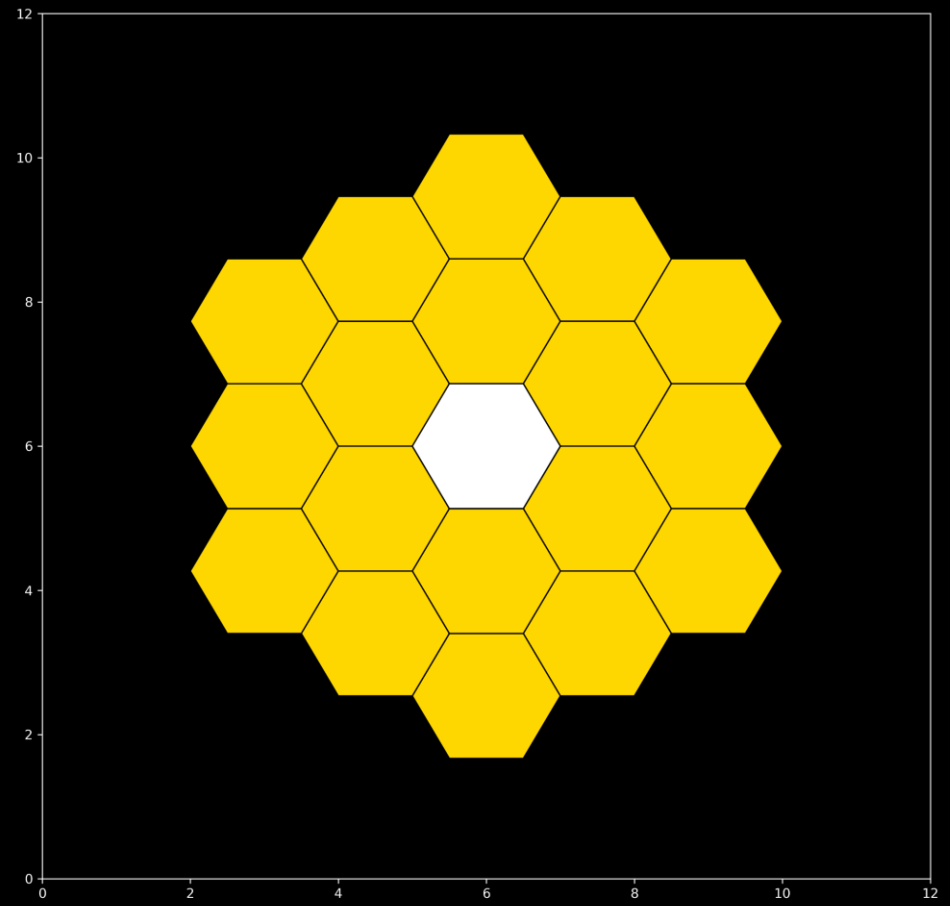


Image DIY




```
#Plot a hexagon given the center coordinate c
def hexagon_plotter(c, col='#FFD700'):
    s = 1
    s1 = s/2
    s2 = s*(np.sqrt(3))/2
    v1, v2, v3, v4, v5, v6 = (c[0] + s1, c[1] + s2), (c[0]+s,
    c[1]), (c[0] + s1, c[1] - s2), (c[0] - s1, c[1] - s2),
    (c[0]-s, c[1]), (c[0] - s1, c[1] + s2)
    polygon1 = Polygon([v1,v2,v3,v4,v5,v6,])
    polygon1.set_facecolor(col)
    polygon1.set_edgecolor('k')
    ax.add_patch(polygon1)
    plt.ylim(0,12)
    plt.xlim(0,12)

#Locate the centers of six hexagons adjacent to the hexagon
with center c.
def hex_centers(c):
    s = 1
    s1 = 1.5*s
    s2 = s*np.sqrt(3)/2
    v1 = (c[0] + s1, c[1] + s2)
    v2 = (c[0] + s1, c[1] - s2)
    v3 = (c[0], c[1] - s2*2)
    v4 = (c[0] - s1, c[1] - s2)
    v5 = (c[0] - s1, c[1] + s2)
    v6 = (c[0], c[1] + s2*2)
    return (v1,v2,v3,v4,v5,v6)
```

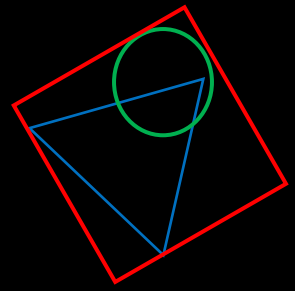
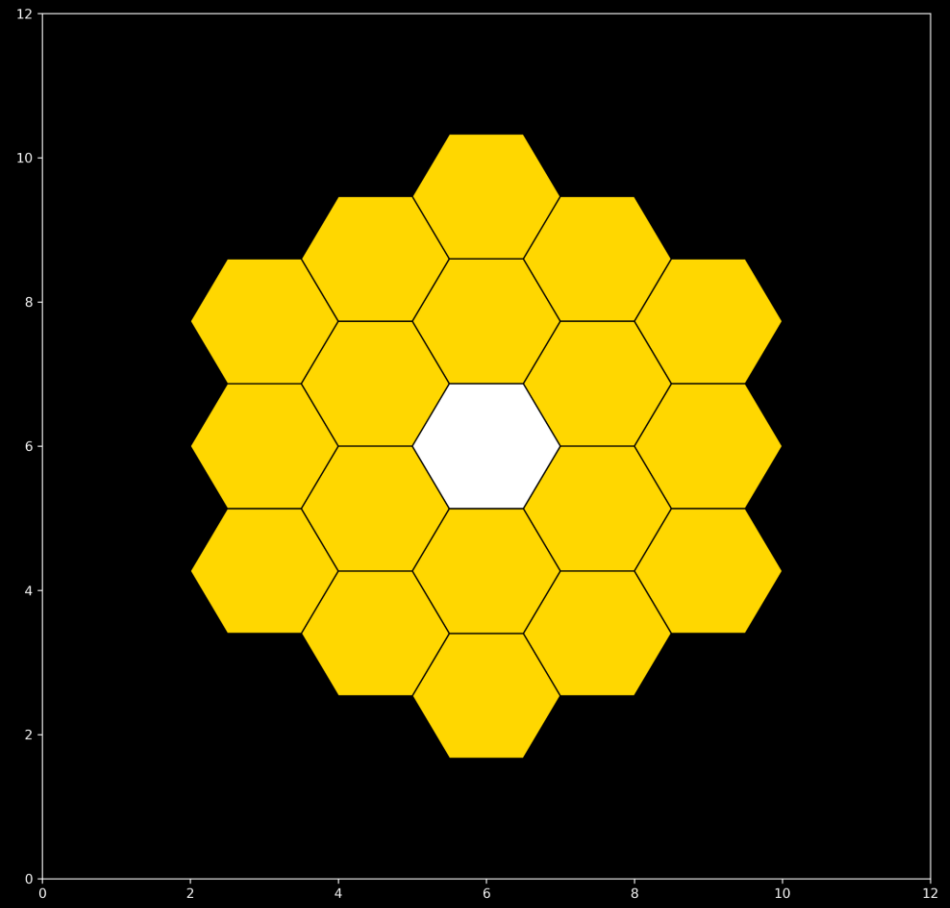
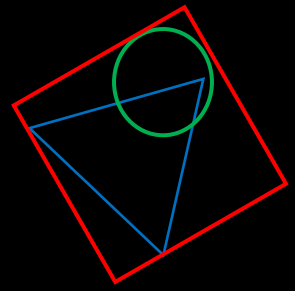


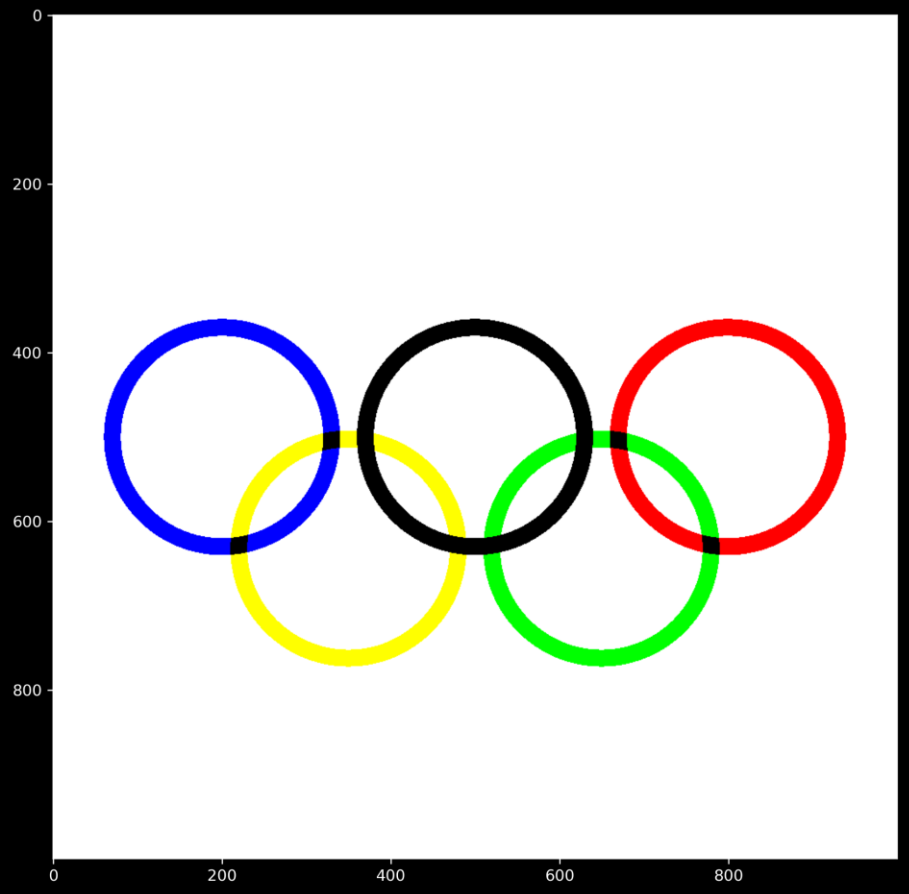
Image DIY




```
c = (6,6)
fig, ax = plt.subplots(1,1, figsize=(12,12))
hexagon_plotter(c)
a = hex_centers(c)
for center in a:
    hexagon_plotter(center)
    tier2 = hex_centers(center)
    for center2 in tier2:
        hexagon_plotter(center2)
hexagon_plotter(c, col='w')
```



Color Image

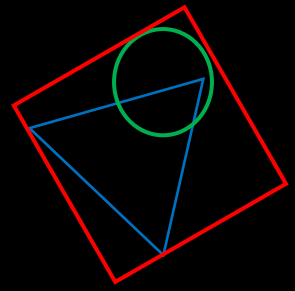


Strategy

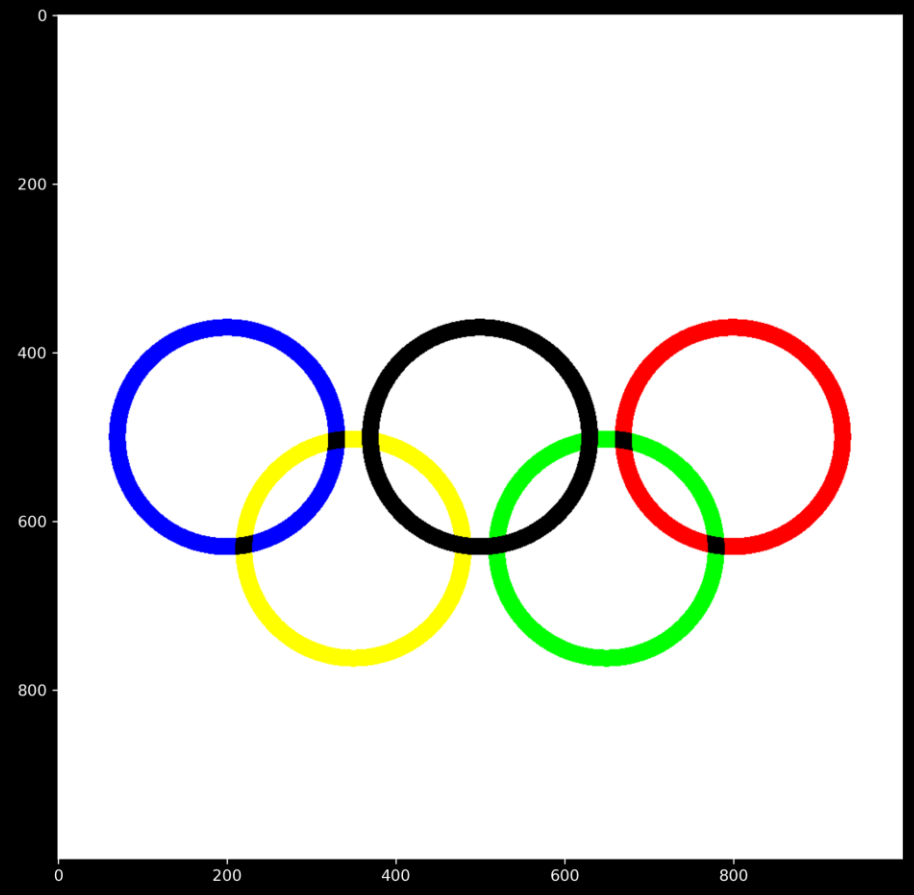
Same brute-force Cartesian coordinates as the JWST image!

1. Set a meshgrid and assign it a value 1. This turns the plot background white, so to get the desired ring color, I must perform **subtractive color mixing** from this overall white color. For example, I need to set the Red and Green to **0**, keep the Blue **1** in the leftmost ring.

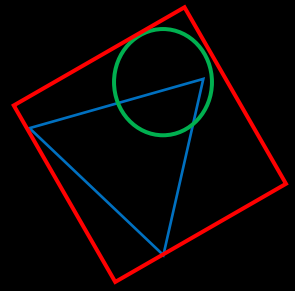
2. Locate the centers of the ring using Cartesian coordinates. The spacing can be adjusted according to preference.



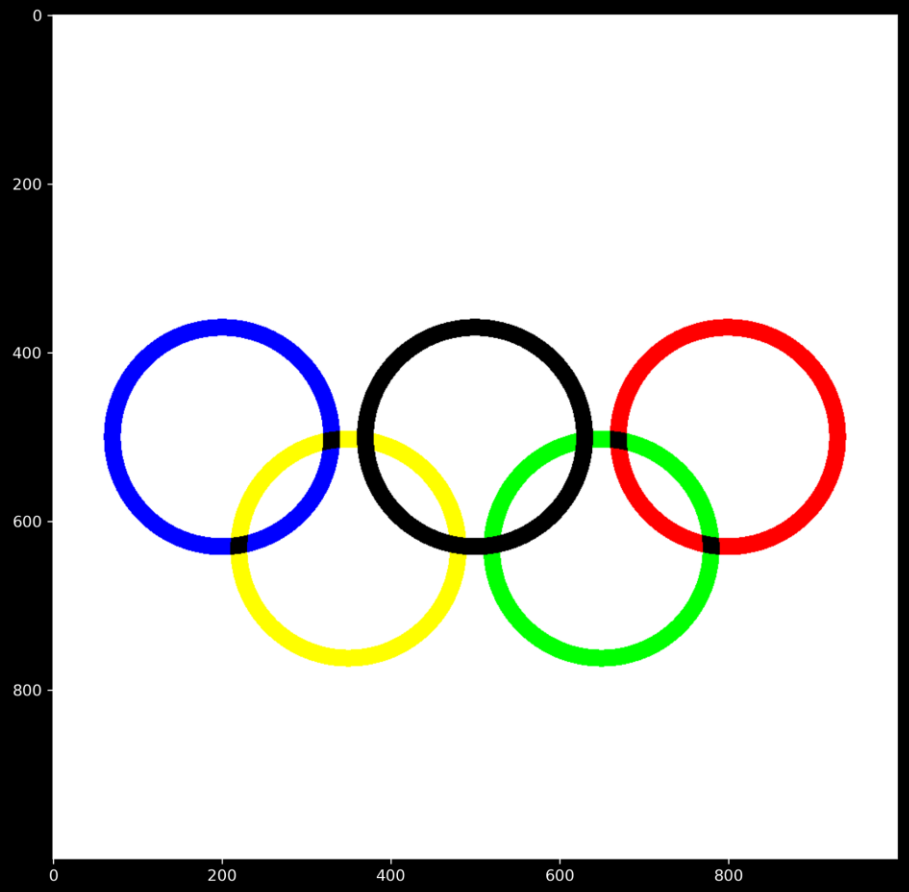
Color Image



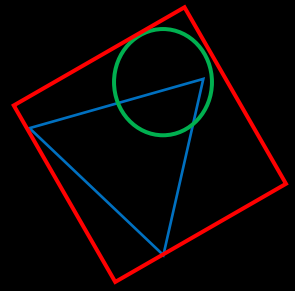

```
#Olympic rings, subtractive color mixing
N=1000
x= np.linspace(0,10,num=N)
y=np.linspace(0,30,num=3*N)
X,Y = np.meshgrid(x,y)
def circle_plotter(r_o, x,y, c, r_i, Rd, Gn, Bl):
    R = np.sqrt((X-x)**2 + (Y-y)**2)
    reg = (R<r_o)*(R>r_i) #color between inner and outer radius
    if c == 0: #blue circle
        Rd[np.where(reg)] = 0.0
        Gn[np.where(reg)] = 0.0
    if c == 1: #yellow circle
        Bl[np.where(reg)] = 0.0
    if c == 2: #black circle
        Rd[np.where(reg)] = 0.0
        Gn[np.where(reg)] = 0.0
        Bl[np.where(reg)] = 0.0
    if c == 3: #green circle
        Rd[np.where(reg)] = 0.0
        Bl[np.where(reg)] = 0.0
    if c==4: #red circle
        Gn[np.where(reg)] = 0.0
        Bl[np.where(reg)] = 0.0
    return (Rd, Gn, Bl)
```



Color Image

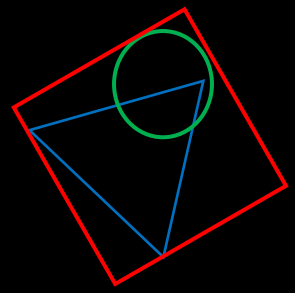



```
I = np.ones((N,N,3))
l, s = 1.5, 2 #circle spacing parameters
c = [2,5] #blue circle center

#circle centers, from yellow to red (left to right)
c1 = [c[0]+l, c[1] + np.sqrt(s**2 - l**2)]
c2= [c[0]+2*l, c[1]]
c3= [c[0]+3*l, c[1]+np.sqrt(s**2 - l**2)]
c4=[c[0]+4*l, c[1]]
C=[c,c1,c2,c3,c4]

#inner and outer radius
R_o, R_i = 0.4*(s+l),  0.4*(s+l)-0.2
r,g,b = np.ones((N,N)), np.ones((N,N)), np.ones((N,N))

#plot circles
for col in range(5):
    a=circle_plotter(R_o, x=C[col][0], y = C[col][1], r_i = R_i, c=col,
Rd=r, Gn=g, Bl=b)
    r,g,b = a[0], a[1], a[2]

I = np.zeros((N,N,3))
I[...,0] = r
I[...,1] = g
I[...,2] = b

fig = plt.figure(figsize=(10,10))
plt.imshow(I)
```

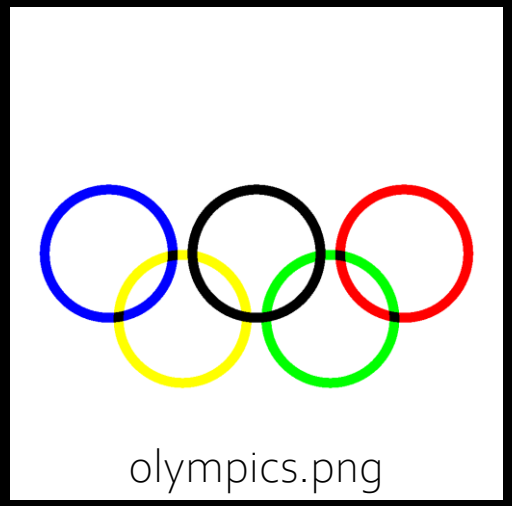


Results and Discussion

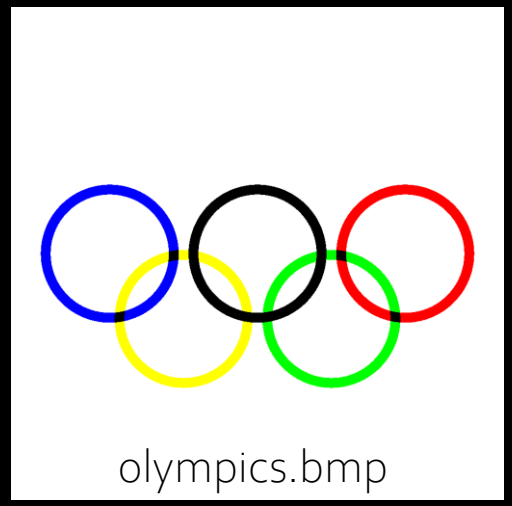
1. To create synthetic images mathematically.
2. To use appropriate file formats for saving images.
3. To open images using Python.
4. To apply backprojection to change the image histogram.
5. To improve the appearance of images.



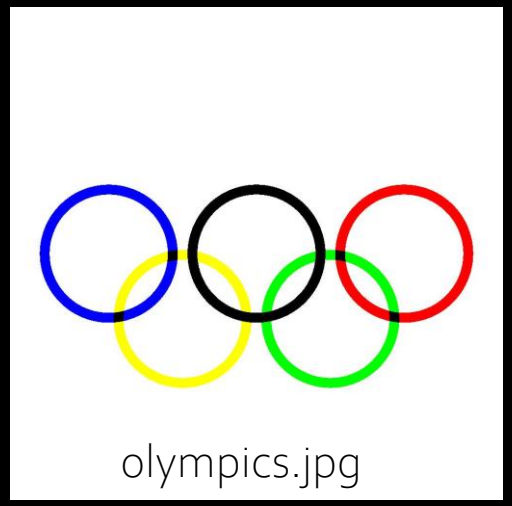
Results and Discussion



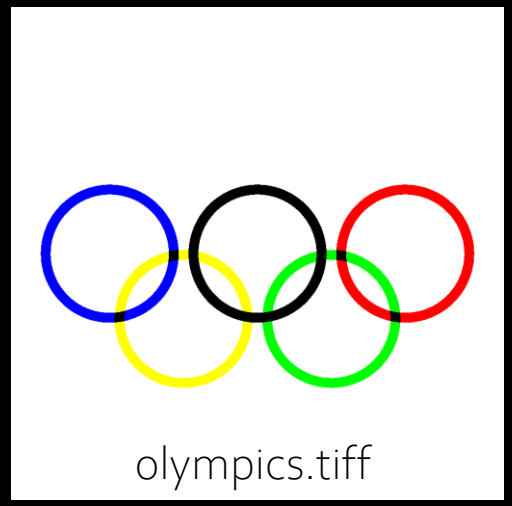
olympics.png



olympics.bmp



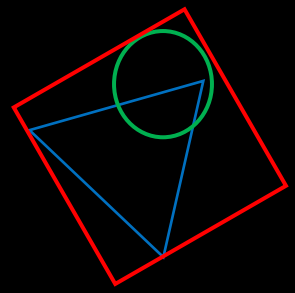
olympics.jpg



olympics.tiff

I is an array of the image.

```
plt.imsave("olympics.jpg",I)
plt.imsave("olympics.png",I)
plt.imsave("olympics.bmp",I)
plt.imsave("olympics.tiff",I)
```

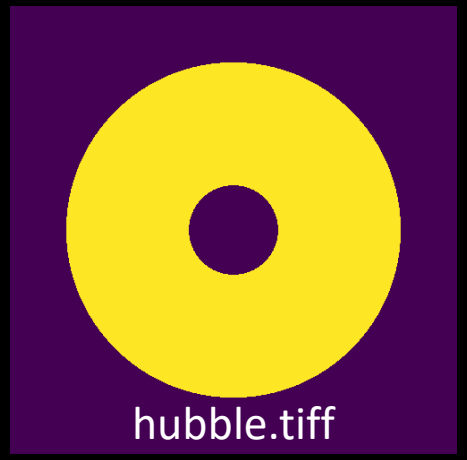


Results and Discussion


hubble.bmp

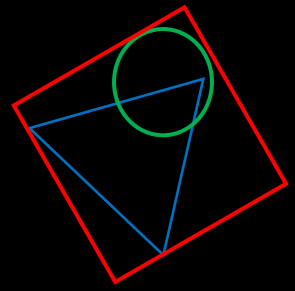

hubble.jpg


hubble.png

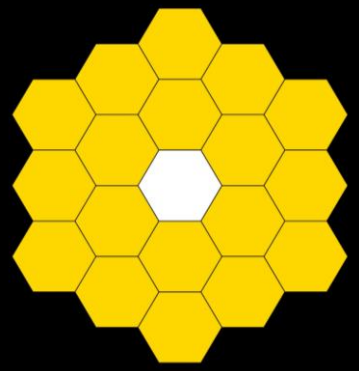

hubble.tiff

A is an array of the image.

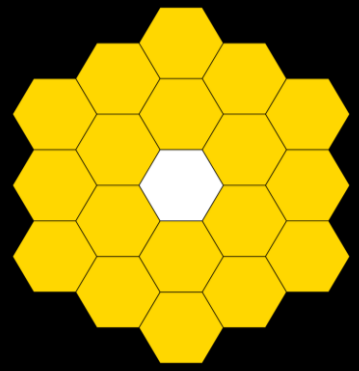
```
plt.imsave(“hubble.jpg",A)
plt.imsave(“hubble.png",A)
plt.imsave(“hubble.bmp",A)
plt.imsave(“hubble.tiff",A)
```



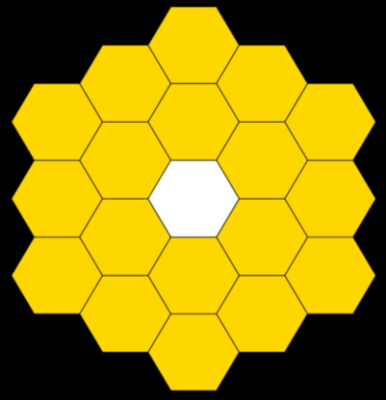
Results and Discussion



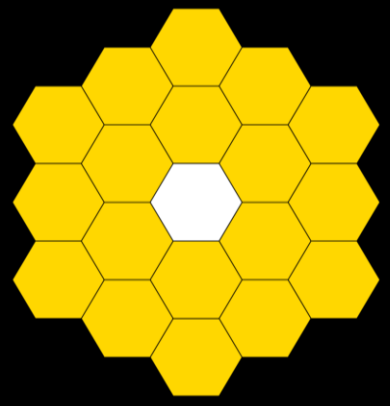
jwst.jpg



jwst.png



jwst.tiff



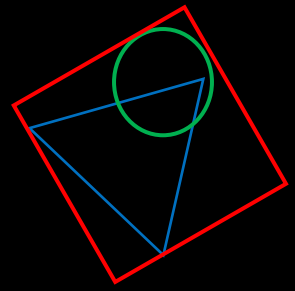
jwst.bmp

The JWST image in my code is not storred in an array, so I have to save the figure itself., with the x and y labels removed.

```
plt.savefig('jwst.jpg', dpi=300)
plt.savefig('jwst.png', dpi=300)
plt.savefig('jwst.tiff', dpi=300)
```

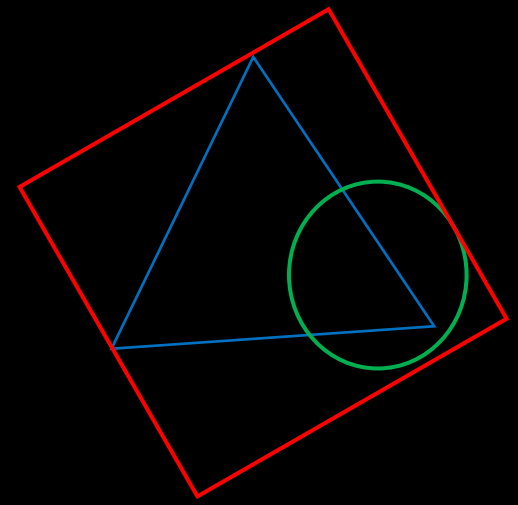
To save as bmp, I have to use PIL library because matplotlib won't allow me to directly save a figure into bmp.

```
from PIL import Image as im
im.open("jwst.png").save("jwst.bmp")
```



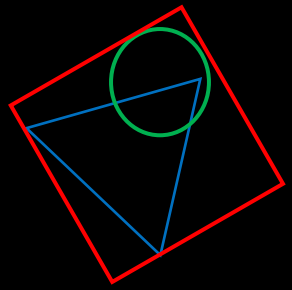
Results and Discussion

1. To create synthetic images mathematically.
2. To use appropriate file formats for saving images.
3. To open images using Python and GIMP
4. To apply backprojection to change the image histogram.
5. To improve the appearance of images.



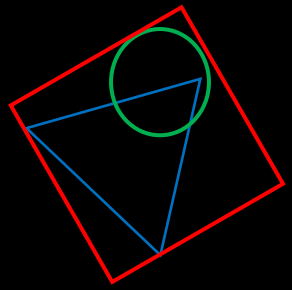
Altering the I-O Curve using GIMP

Screenshot of a cave I took while testing SEUS PTGI shaders in Minecraft 1.19 (original)

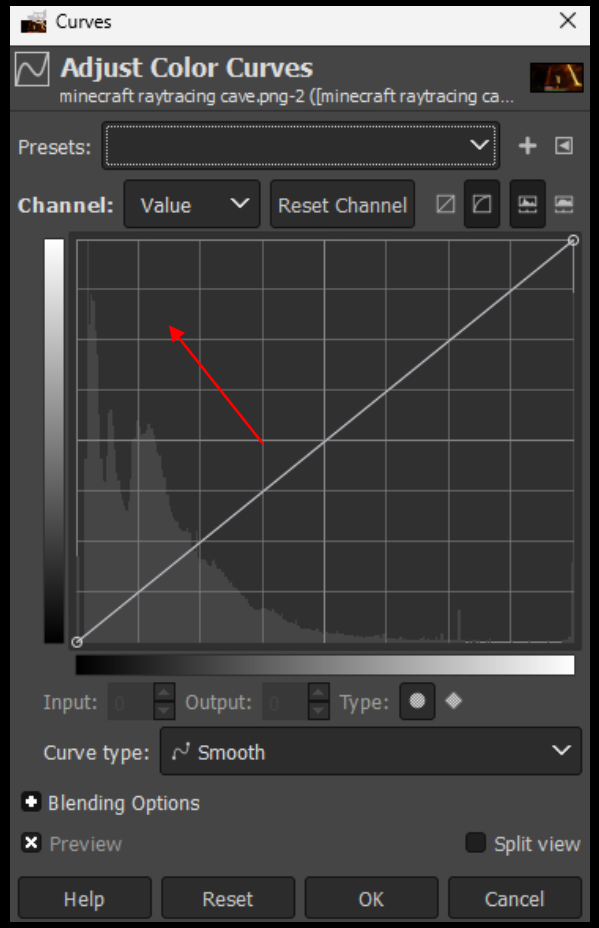


Altering the I-O Curve using GIMP

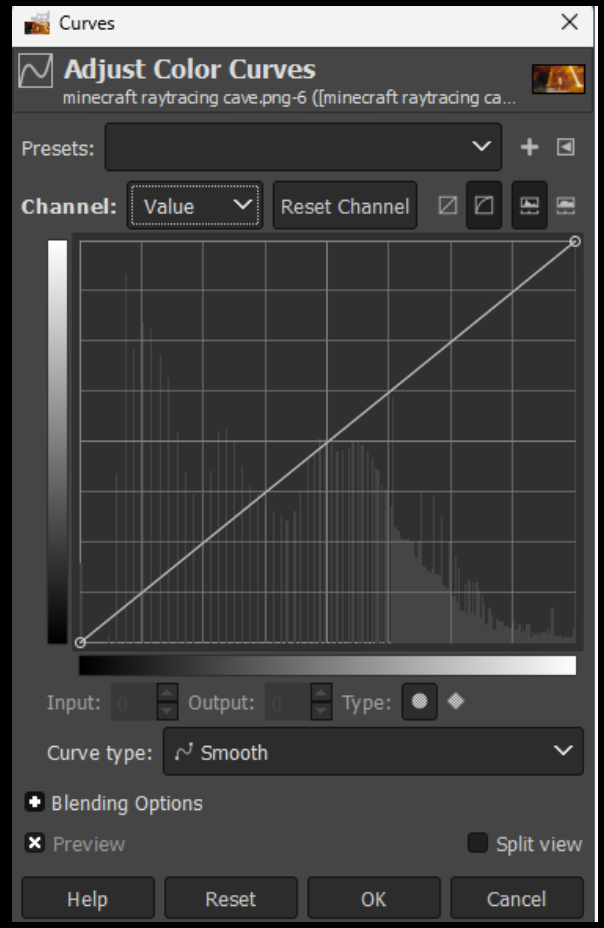
Screenshot of a cave I took while testing SEUS PTGI shaders in Minecraft 1.19 (processed)

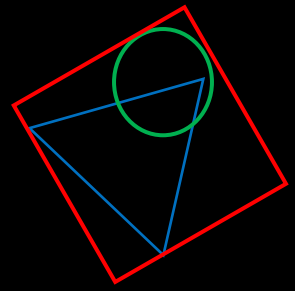


Altering the I-O Curve using GIMP



Saving the image





Altering the I-O Curve using GIMP

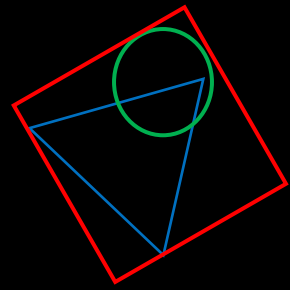
The RGB color curves can also be adjusted.

Blue adjustment

Green adjustment

Red adjustment

RGB + Value manipulation



Images Used

From “Towards the City” by Yutaka Takanashi (1974)

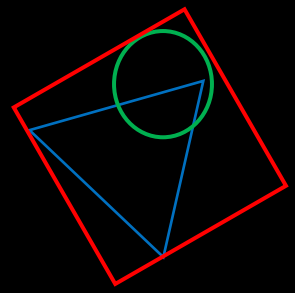
Images Used

Mr. Incredibles uncanny meme, source unknown

Histogram Backprojection

Modules/Libraries Used

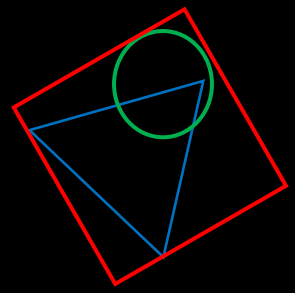
```
import numpy as np
import matplotlib.pyplot as plt
import os
from scipy import interpolate
import matplotlib.image as mpimg
import cv2
```



Histogram Backprojection

Predefined Functions

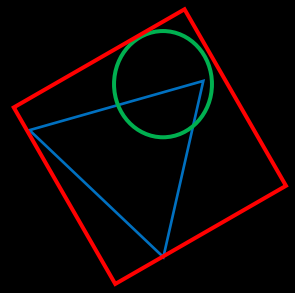
```
def normalizer(array, vmin = 0, vmax = 255): #transform an array into the range [0,255].
    max_val, min_val = np.max(array), np.min(array)
    result = (array-min_val)*(255/(max_val-min_val))
    return np.array(result, dtype='uint8')

def PDF(img): #obtain the probability distribution function of the image histogram.
    size = np.shape(img)
    pix = size[0]*size[1]
    hist = np.histogram(img.flatten(), bins=256)[0]
    pdf_normed = np.histogram(img, bins = 256)[0]/pix
    return pdf_normed

def CDF(img): # obtain the cumulative dist. function
    return np.cumsum(PDF(img))

def backproject(img, desiredCDF): # backproject old cdf array into new cdf by look-up table
    xvals = np.linspace(0,1,len(desiredCDF))
    f= interpolate.interp1d(desiredCDF, xvals, fill_value='extrapolate')
    xnew = normalizer(f(cdf_normed(img)))
    new_img = np.array(xnew[img], dtype='uint8')
    return new_img
```



Histogram Backprojection

Predefined Functions

```
def linear(N=1000): # linear function, range from 0 to 1
    xvals = np.linspace(0,1,N)
    return xvals/255

def sigmoid(N=1000, a=20): # sigmoid, range from 0 to 1, center at 0.5
    xvals = np.linspace(0,1,N)
    desiredCDF_sigmoid = 1/(1+np.exp(-a*(xvals - 1/2)))
    return desiredCDF_sigmoid
```

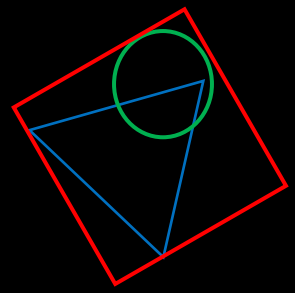


Histogram Backprojection

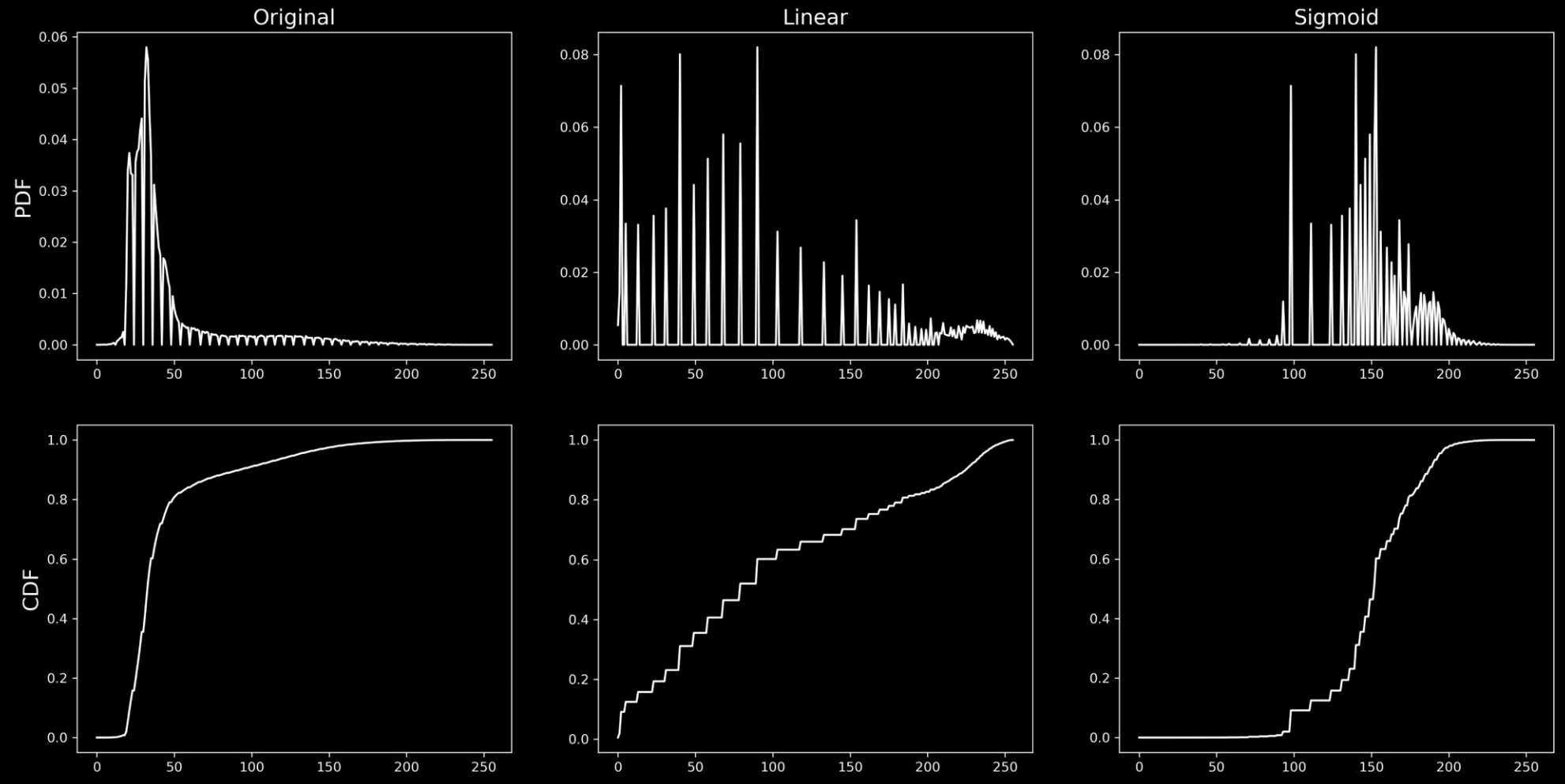
Original

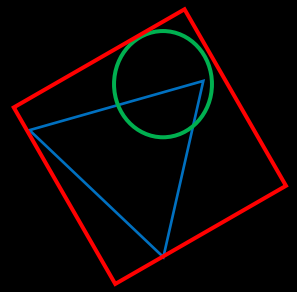
Linear

Sigmoid



Histogram Backprojection



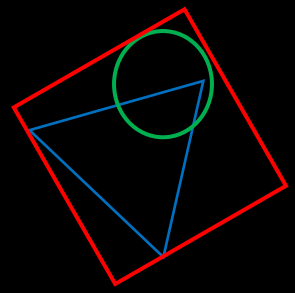


Histogram Backprojection

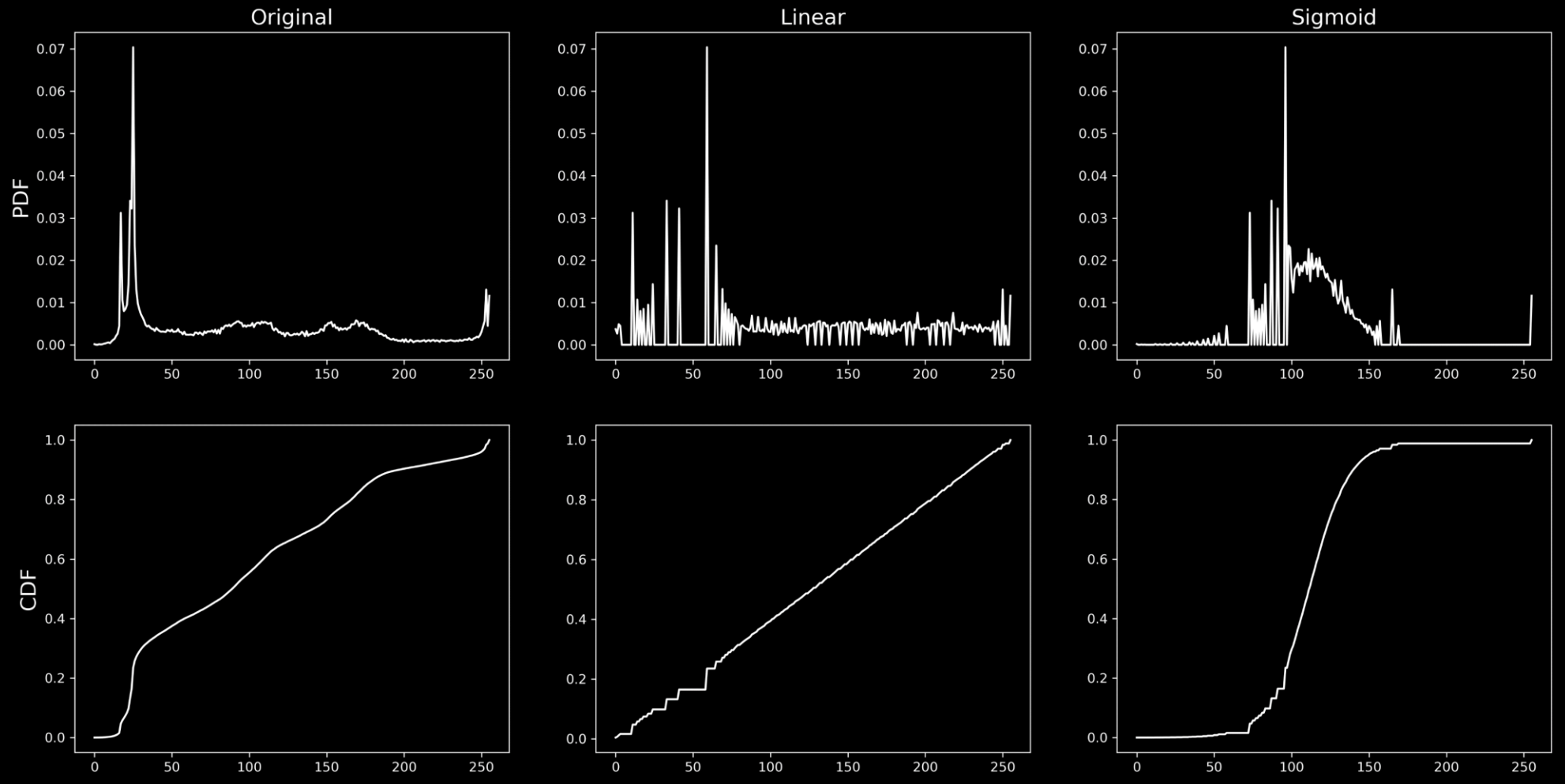
Original

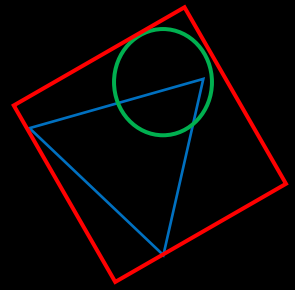
Linear

Sigmoid



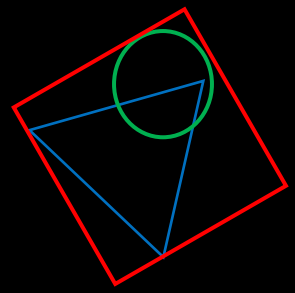
Histogram Backprojection



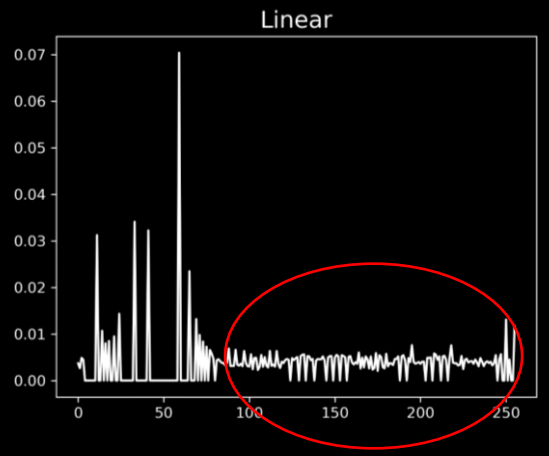


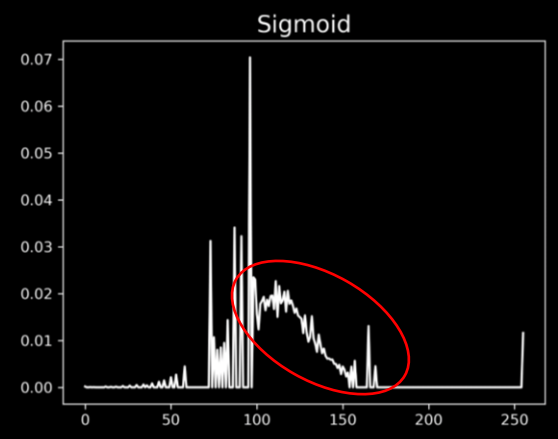
Histogram Backprojection

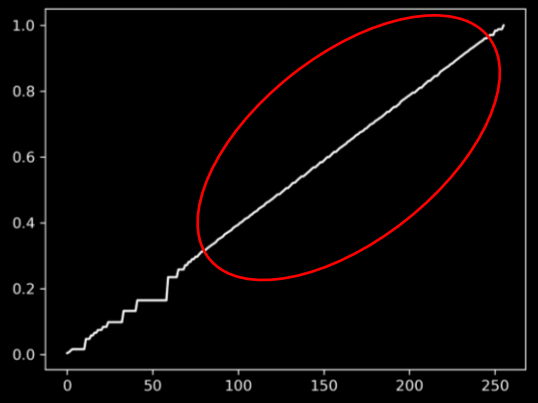
```
img1_linear = backproject(img1, linear())
img1_sigmoid = backproject(img1, sigmoid(a=30))
images = [img1, img1_linear, img1_sigmoid]

fig, ax = plt.subplots(1,3, figsize=(20,5)) # Plot the three images
ax=ax.flatten()
for i in range(3):
    ax[i].imshow(images[i], cmap='gray')
    ax[i].set_xticks([])
    ax[i].set_yticks([])
ax[0].set_title('Original')
ax[1].set_title('Linear')
ax[2].set_title('Sigmoid’)

fig, ax = plt.subplots(2,3, figsize=(20,10)) # Plot the PDFs and CDFs
ax=ax.flatten()
for i in range(3):
    ax[i].plot(PDF(images[i]), c='w')
    ax[i+3].plot(CDF(images[i]), c='w')
ax[0].set_title('Original', fontsize=16)
ax[1].set_title('Linear', fontsize=16)
ax[2].set_title('Sigmoid', fontsize=16)
ax[0].set_ylabel('PDF', fontsize=16)
ax[3].set_ylabel('CDF', fontsize=16)
```

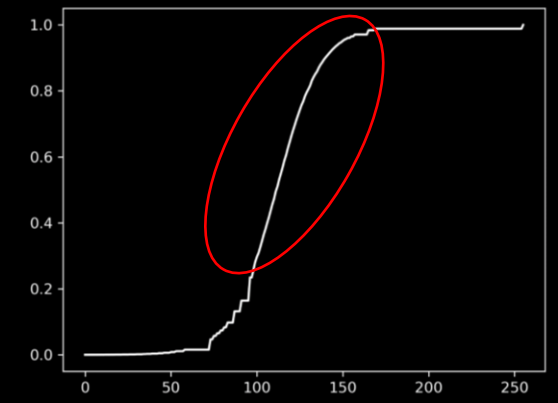


Analysis









Pixel intensities are (approximately) uniformly distributed from 100 to 250.

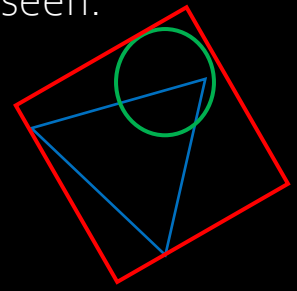
Gaussian behavior at 100 to 150 intensity

1. Ideally, a linear cumulative distribution function (CDF) will produce a uniform probability distribution function (PDF), and a sigmoid CDF will produce a Gaussian PDF.

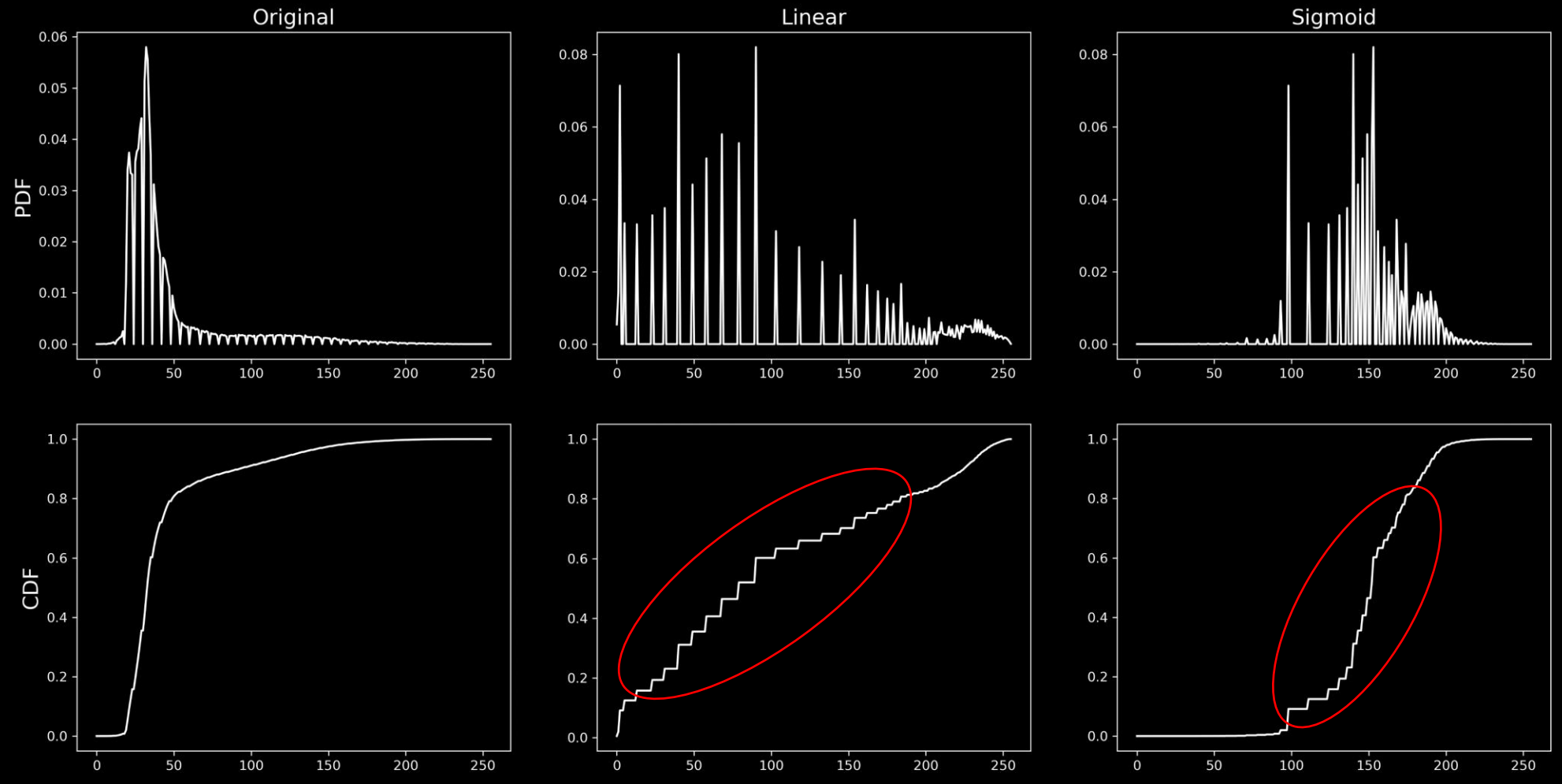
2. However, horizontal lines in the linear CDF mean that there are no pixels with intensity equal to the x values spanned by the horizontal lines, so the PDF falls to zero (sharp dips).

3. Sudden climbs (stairsteps) in the CDF mean that there is an abundance of pixels whose intensities are equal to the corresponding x-value of this stairstep, so the PDF rises (sharp peaks)

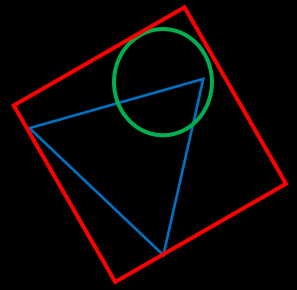
4. For regions that do not deviate from the expected CDF behavior drastically, the theoretical PDF can be seen.



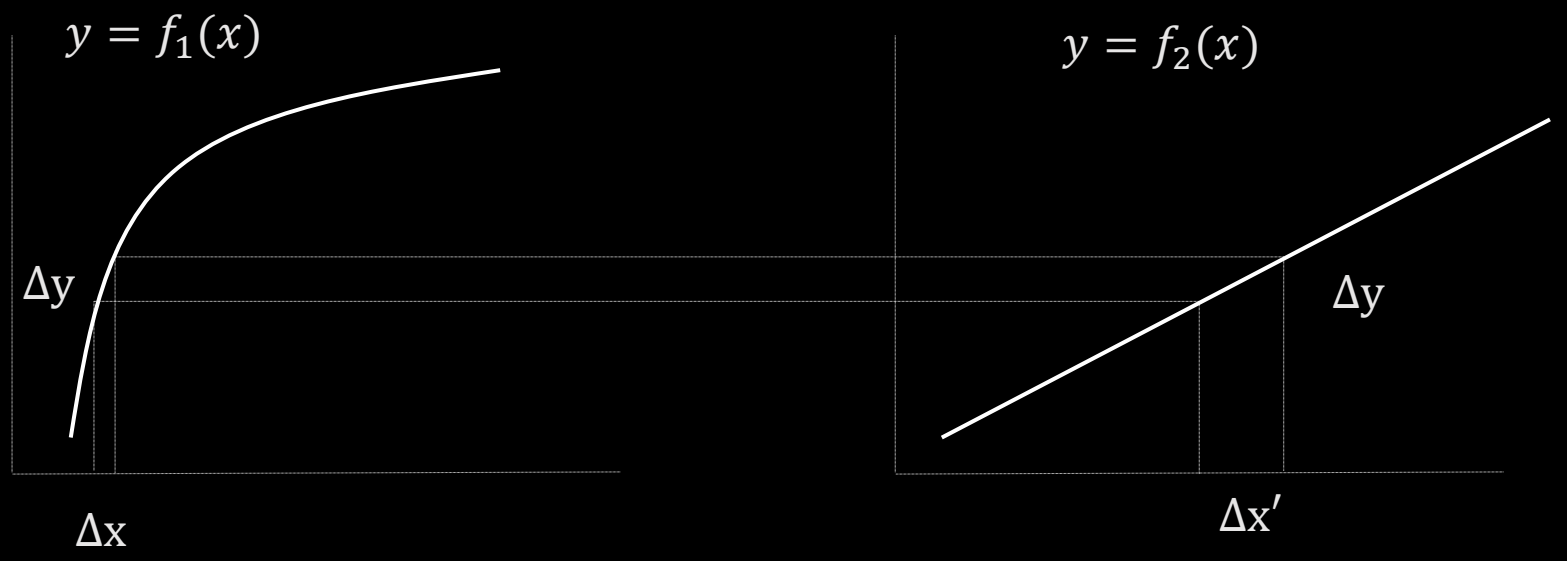
Analysis



Why the jagged lines?

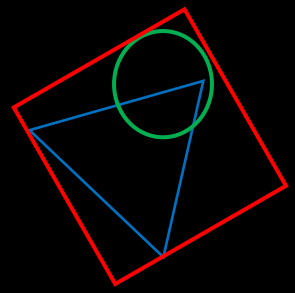


Histogram Backprojection



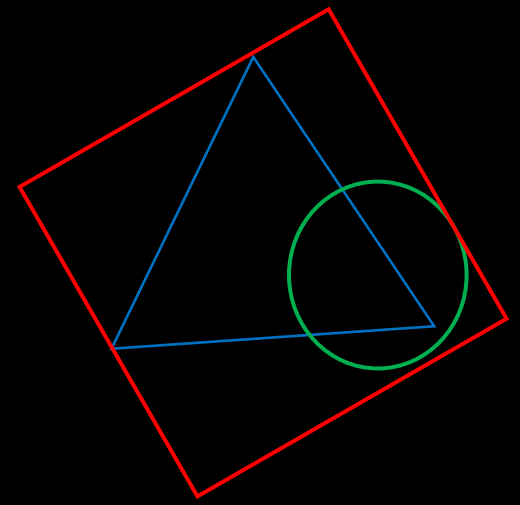
$$\Delta y \approx f_1(x_0)\Delta x = f_2(x_0)\Delta x' \Rightarrow \Delta x' \approx \frac{f_1'(x_0)}{f_2'(x_0)}\Delta x$$

If the instantaneous slope of f_1 at a point is steeper than that in f_2, then $\Delta x' > \Delta x$ resulting to "jumps" or flat lines in the new CDF. The rounding off to uint8 also gives the sharp rises.



Results

1. To create synthetic images mathematically.
2. To use appropriate file formats for saving images.
3. To open images using Python.
4. To apply backprojection to change the image histogram.
5. To improve the appearance of images.



Contrast Stretch

Formula:

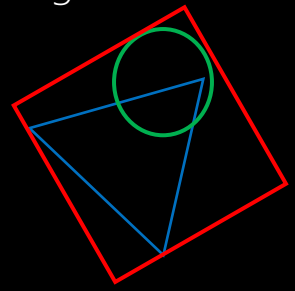
$$I_{new} = 255\left(\frac{I_{old} - I_{min}}{I_{max} - I_{min}}\right)$$

where I_{max} and I_{min} are the maximum and minimum intensities that will be mapped to 255 and 0, respectively.

If $I_{new} > 255$, set it to 255. If $I_{new} < 0$, clip it to zero. Apply this transformation to every channel for RGB stretching.

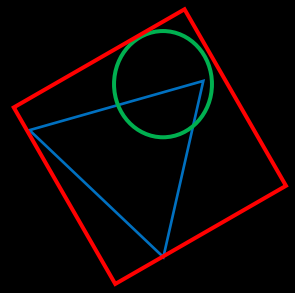
Result:

Stretches the histogram. If the I_{max} is lower than the maximum value of the array, then expect to have more whites in a grayscale image. If I_{min} is higher than the minimum value, then expect to have more blacks than the original image.



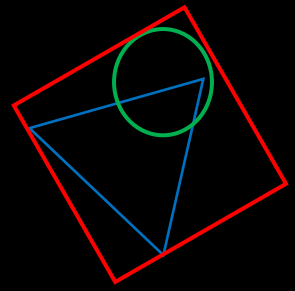
Grayscale Contrast Stretch

```
I_min = np.percentile(img, 5) #5th percentile of the intensity of the grayscale img
I_max = np.percentile(img, 90) #90th percentile of the intensity
new_img = (img - I_min)*255/(I_max - I_min) #contrast stretch image
new_img[new_img<0] = 0 #clip those pixels below I_min
new_img[new_img>255] = 255 #clip those pixels below I_max
```



RGB Contrast Stretch

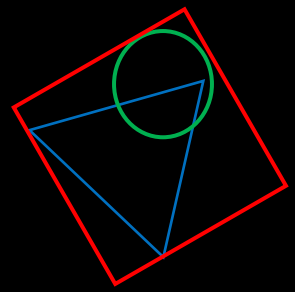
```
def RGB_contrast_stretch(img, low, high):
    img = img[:,:,0:3] #Select the RGB channels only.
    I_min = np.array([np.percentile(img[:,:,i], low[i]) for i in range(len(low))])
    I_max = np.array([np.percentile(img[:,:,i], high[i]) for i in range(len(high))])
    new_img = (img - I_min)*255/(I_max - I_min)
    new_img[new_img<0] = 0 #clip those pixels below I_min
    new_img[new_img>255] = 255 #clip those pixels below I_max
    processed = np.array(new_img, dtype='uint8')
    return processed
```



Grayscale Contrast Stretch

Original

"Towards the City" by Yutaka Takanashi (1974)

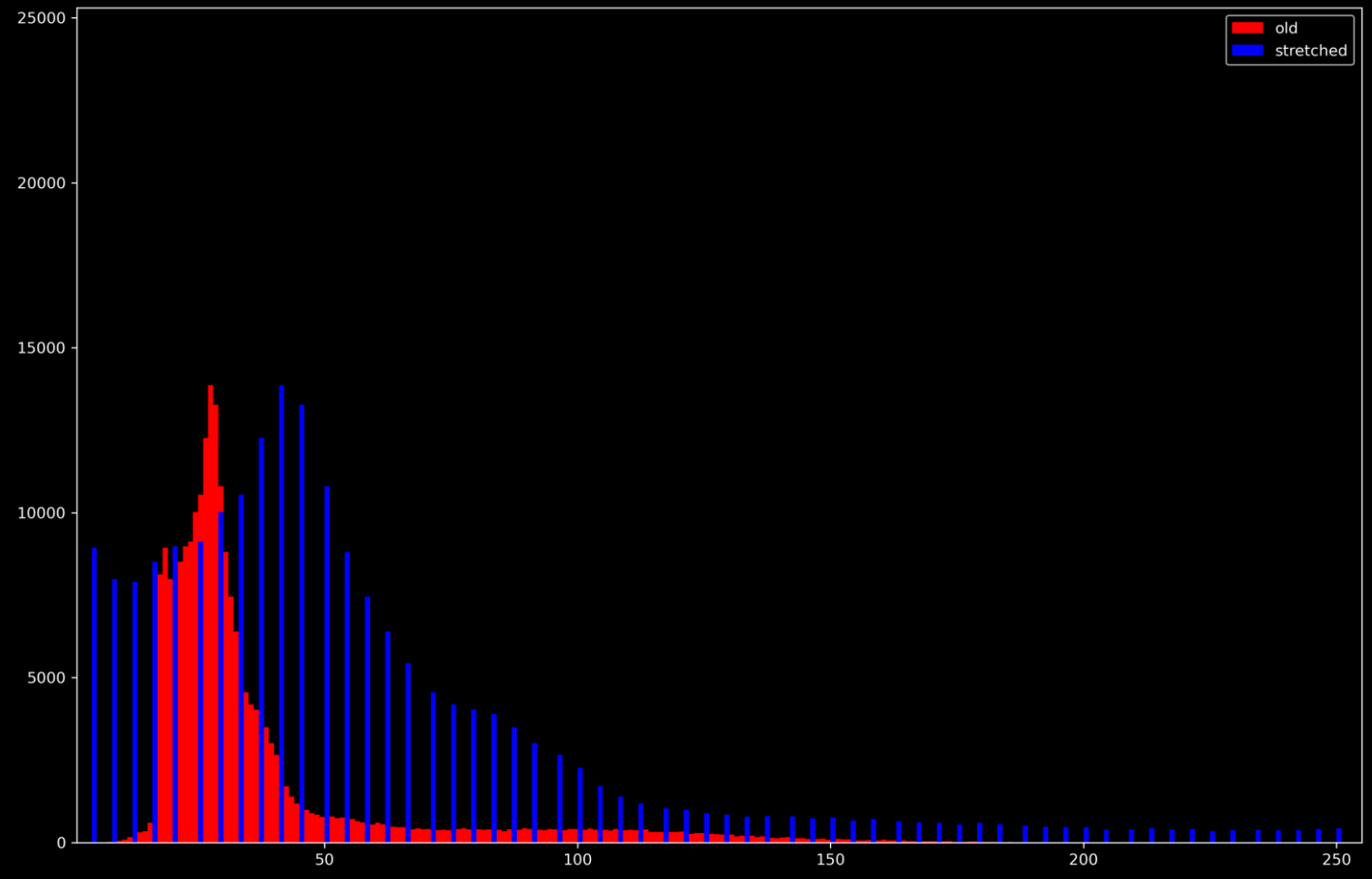


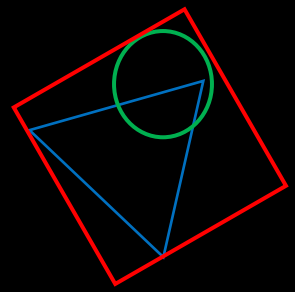
Grayscale Contrast Stretch

Processed (5th percentile min. intensity, 90th percentile max intensity)

"Towards the City" by Yutaka Takanashi (1974)

Grayscale Contrast Stretch

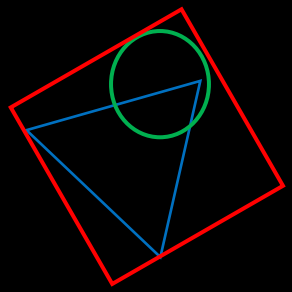




RGB Contrast Stretch

Original

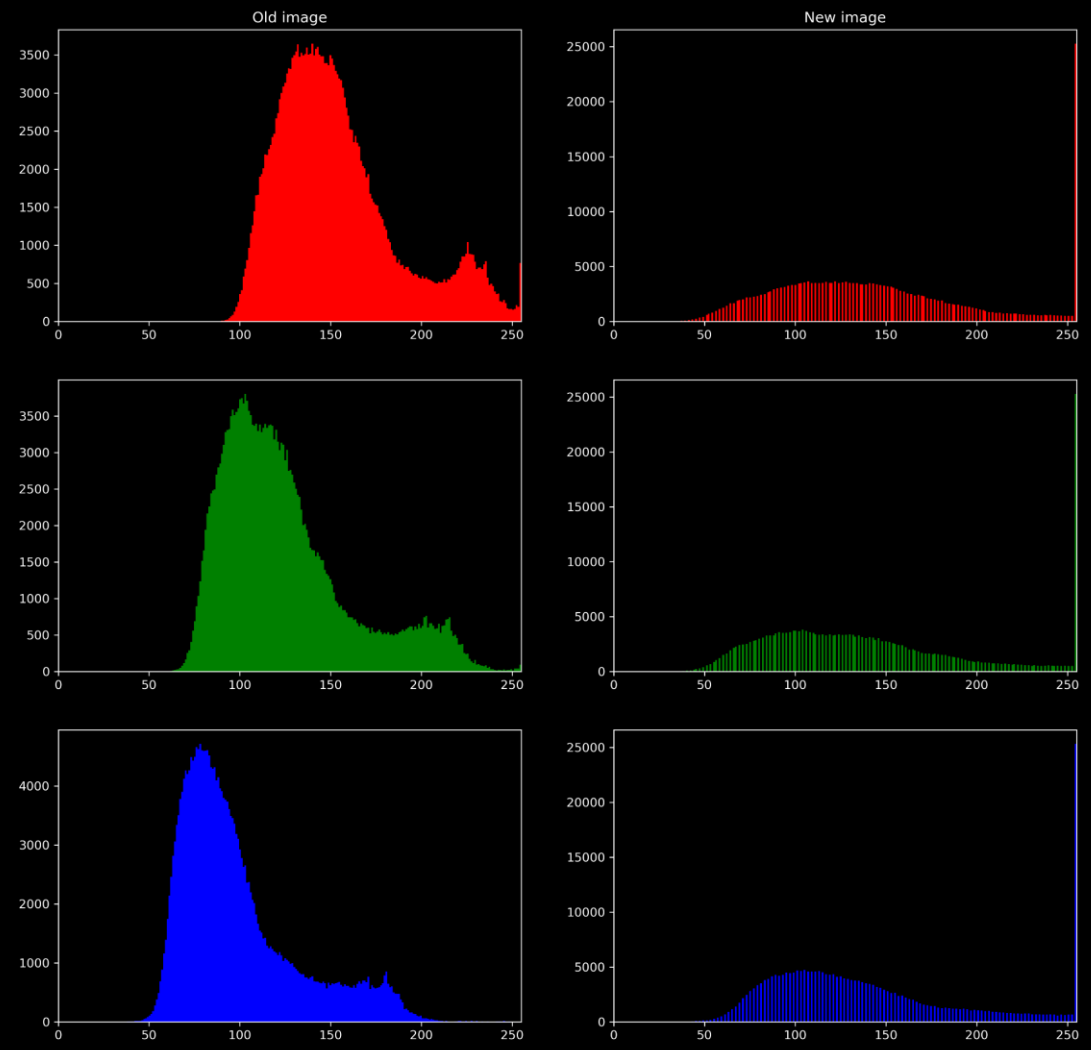
Vintage faded photograph of two men. England circa 1890. By duncan1890 from istockphoto.com

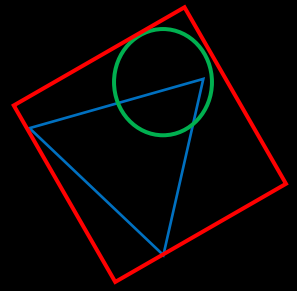


RGB Contrast Stretch

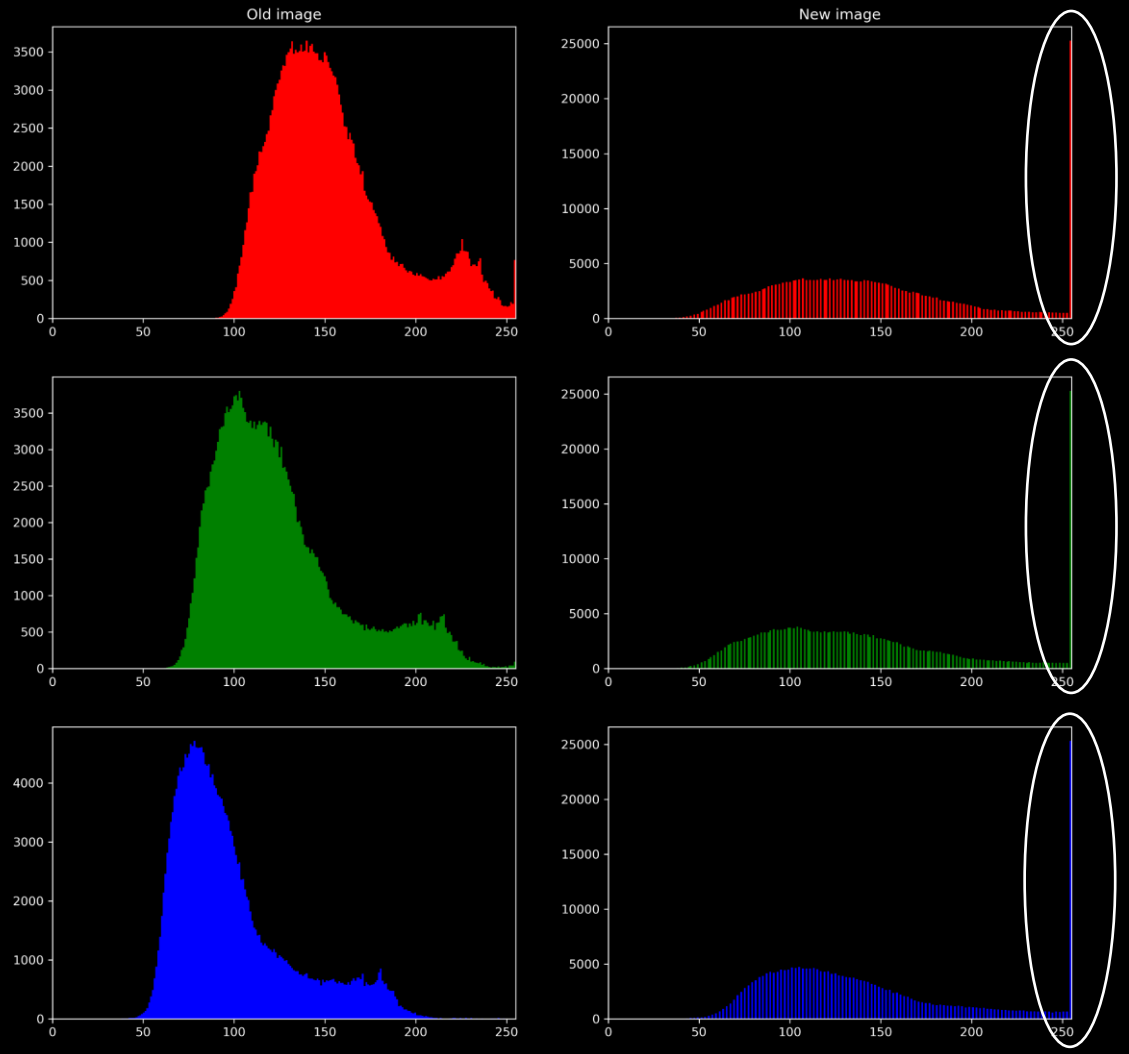
Processed

RGB Histogram

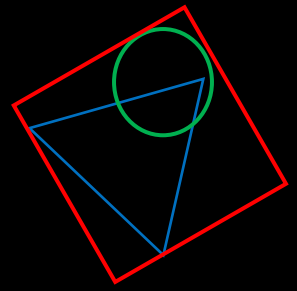




RGB Histogram



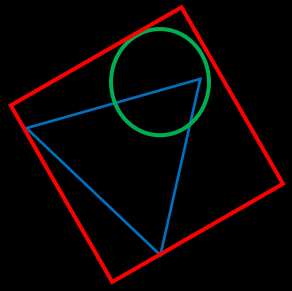
Note that most of the pixels are clipped at 255, turning the image bright white from a brown faded photograph.



RGB Contrast Stretch

Original

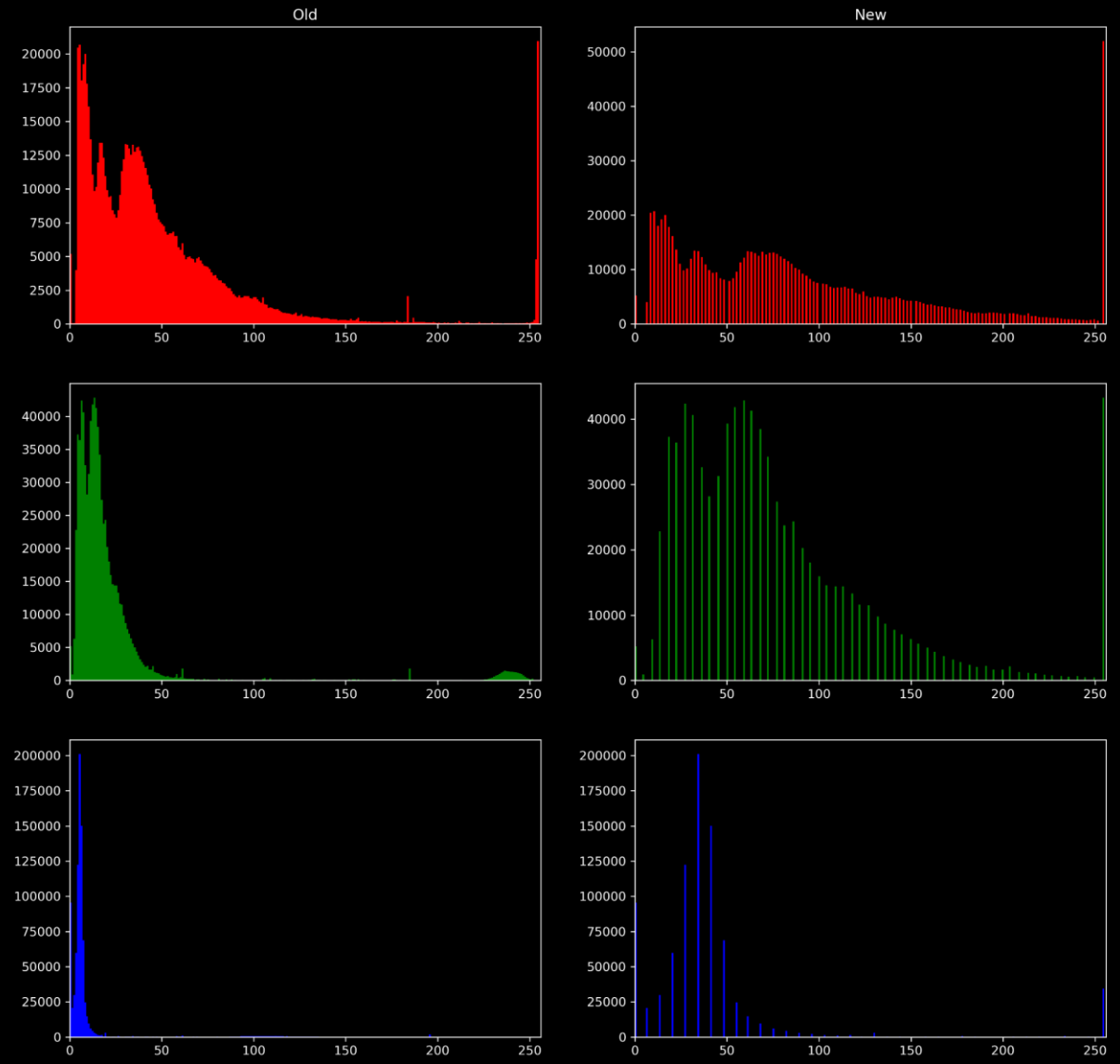
Screenshot of a cave I took while testing SEUS PTGI shaders in Minecraft 1.19

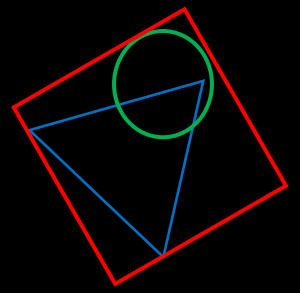


RGB Contrast Stretch

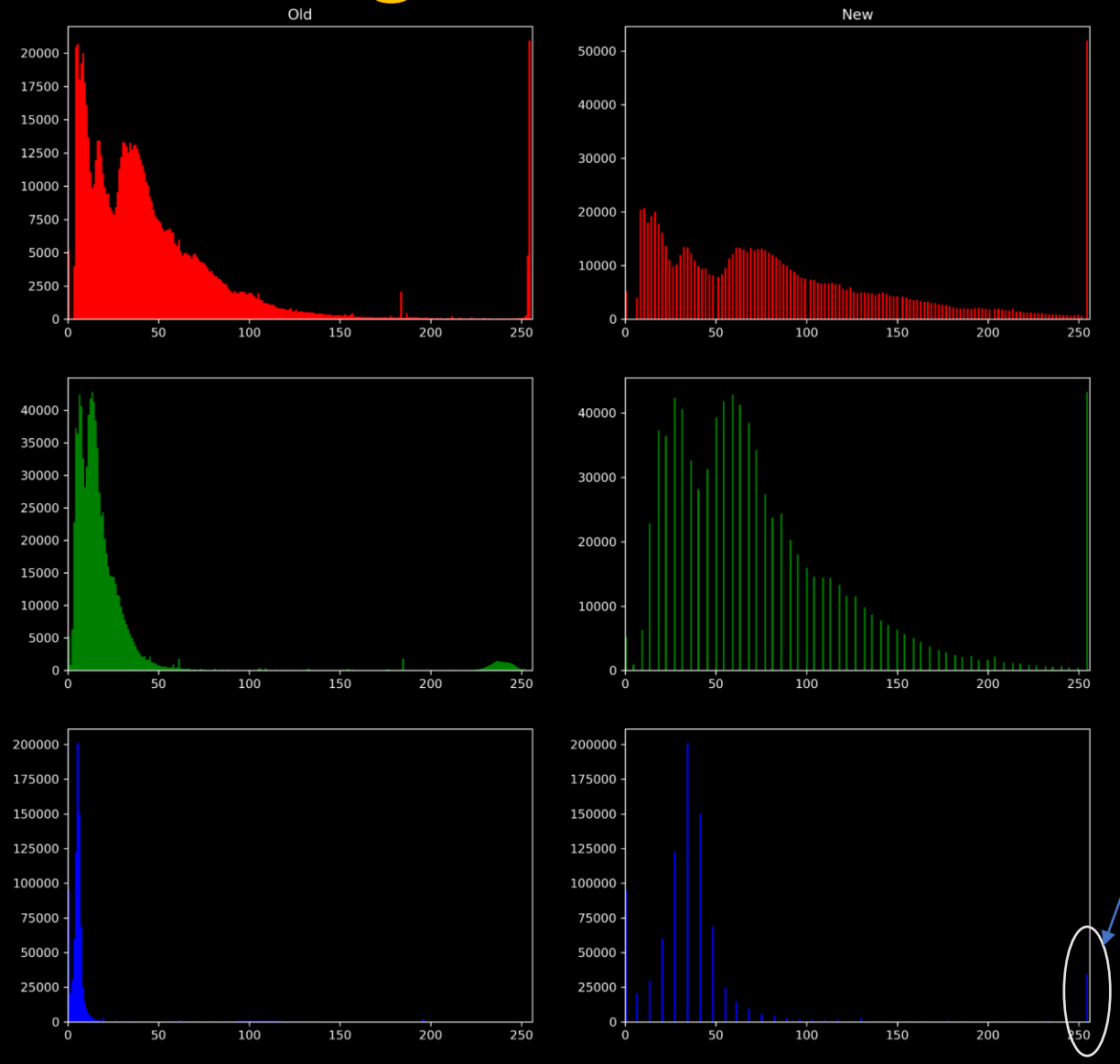
Processed

RGB Histogram

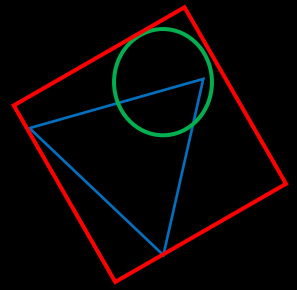




RGB Histogram



The clipping at 255 for the blue channel explains why the processed image has a significantly more pronounced blue color (the water column and the blue soul torches)

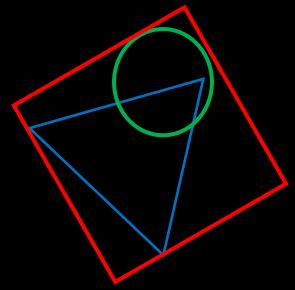


RGB Contrast Stretch

```
victorian_new = RGB_contrast_stretch(victorian, [0,0,0], [90,90,90]) #All RGB channels trimmed at 0 and 90th
percentile.
fig, ax = plt.subplots(1,2, figsize=(15,5))
victorians=[victorian, victorian_new]
for i in range(2):
    ax[i].imshow(victorians[i])
    ax[i].set_xticks([])
    ax[i].set_yticks([])
ax[0].set_title('Original', fontsize=16)
ax[1].set_title('New', fontsize=16)
```

Original

New

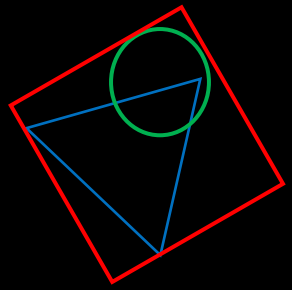


RGB Contrast Stretch

```
cave_new = RGB_contrast_stretch(cave, [0,0,0], [94,95,96]) #Red trimmed at 94rd, green at 95th and blue at 96th percentile.
fig, ax = plt.subplots(1,2, figsize=(15,5))
caves=[cave, cave_new]
for i in range(2):
    ax[i].imshow(caves[i])
    ax[i].set_xticks([])
    ax[i].set_yticks([])
ax[0].set_title('Original', fontsize=16)
ax[1].set_title('New', fontsize=16)
```

Original

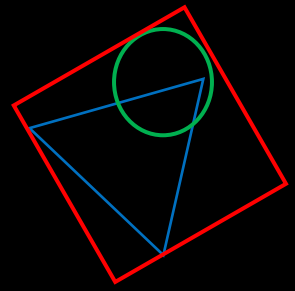
New



Analysis

1. The RGB contrast stretch algorithm stretches and clips the RGB intensities at specified percentiles.

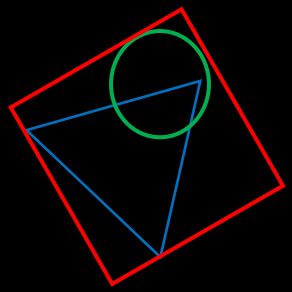
2. If the image has lots of dark pixels (resp. bright pixels), then stretching the distribution and clipping max intensity (resp. min intensity) will brighten the image and improve contrast (as seen in the cave screenshot).



Gray World

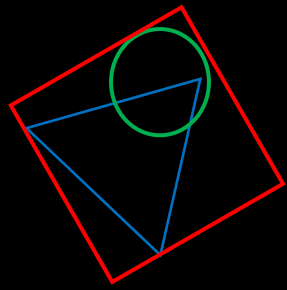
Original

Vintage faded photograph of two men. England circa 1890. By duncan1890 from istockphoto.com



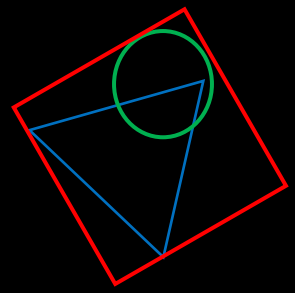
Gray World

Processed



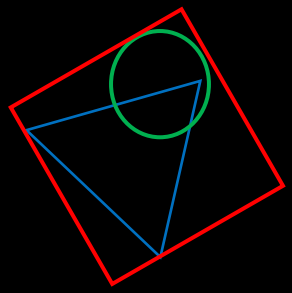
Gray World

Original



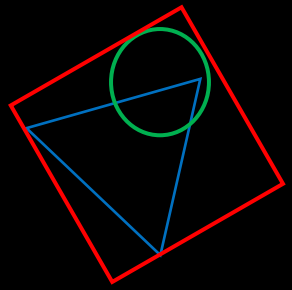
Gray World

Processed

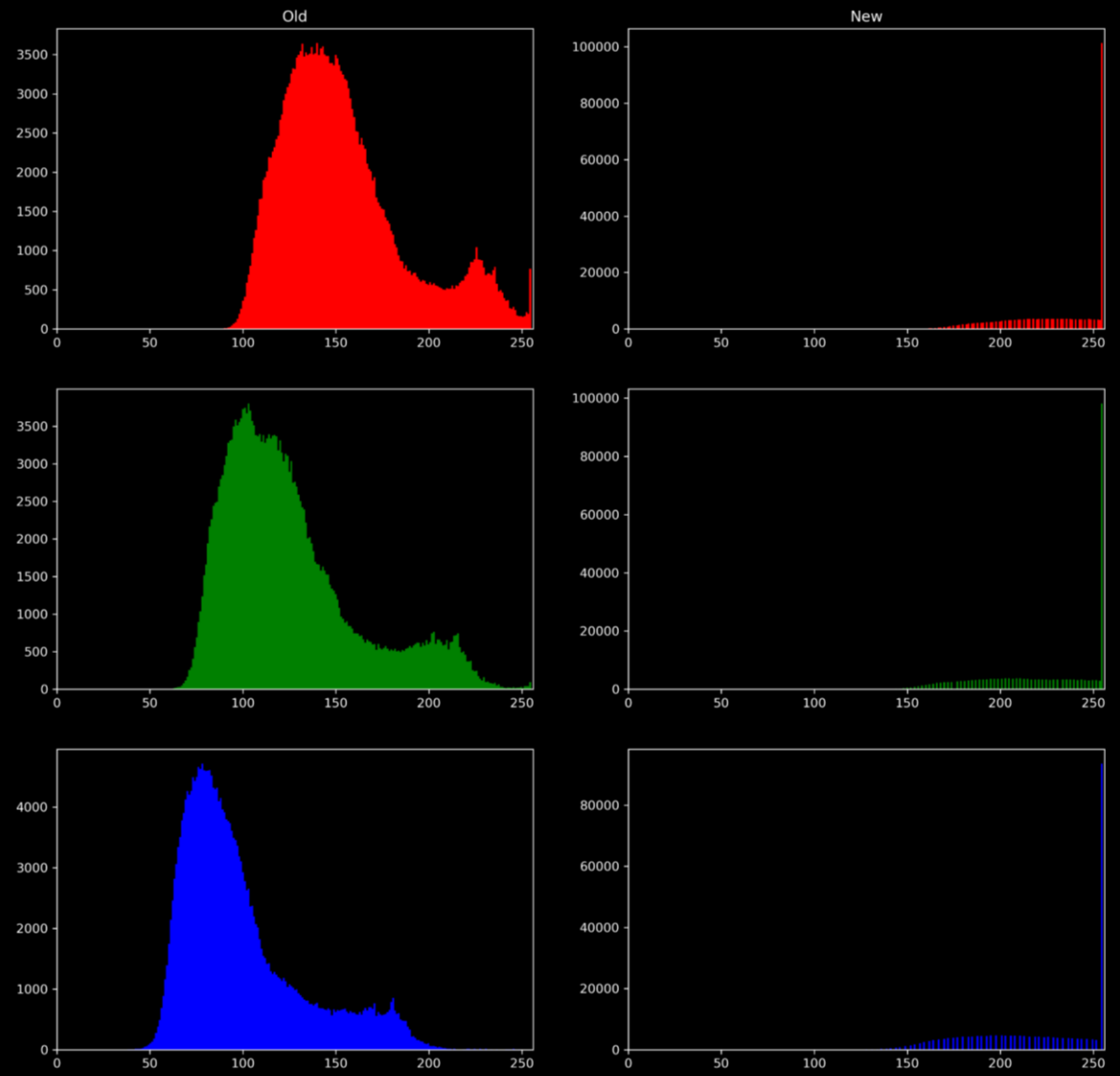


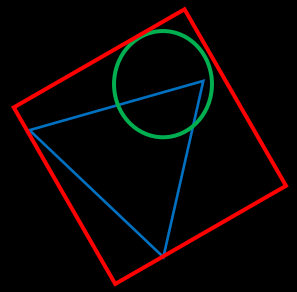
Gray World

```
def gray_world1(img): #Values below and above 255 are clipped.
    Rave, Gave, Bave = img.mean(axis=(0,1))[:3]
    new = np.copy(img)
    new = np.array(new, dtype='float')
    new[...,0], new[...,1], new[...,2] = img[...,0]/Rave, img[...,1]/Gave, img[...,2]/Bave
    new[...,:]= new[...,:]*255
    new[new>255] = 255
    return np.array(new, dtype='uint8')

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,15))
ax1.imshow(victorian)
ax1.set_title('Old image')
victorian_img3 = gray_world1(victorian)
ax2.imshow(victorian_img3)
ax2.set_title('New image')
```

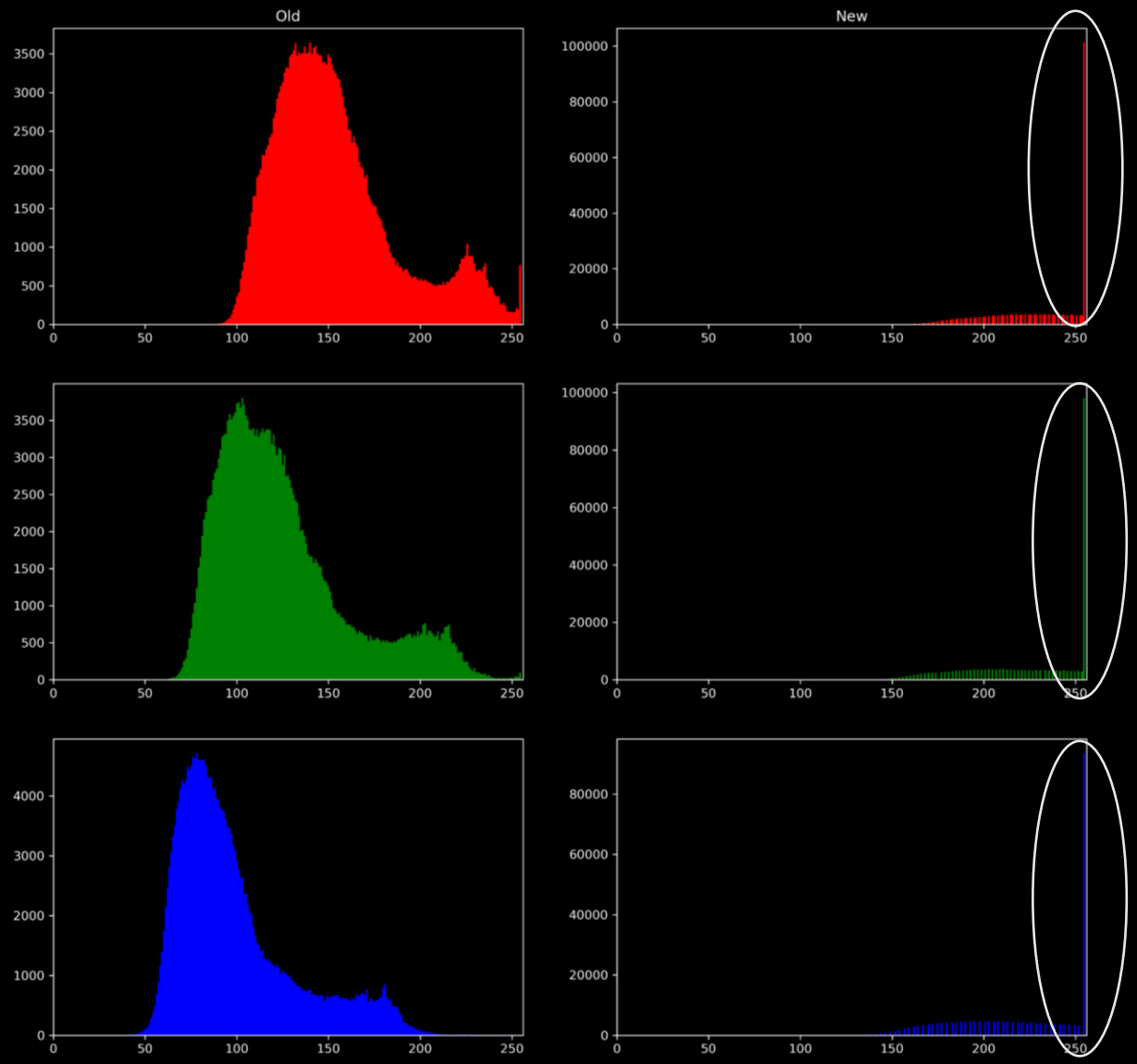


Gray World

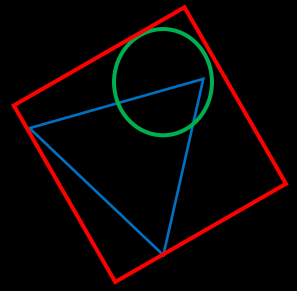

Old
New
3500
3000
2500
2000
1500
1000
500
0
100000
80000
60000
40000
20000
4000
0
50
100
150
200
250



Gray World



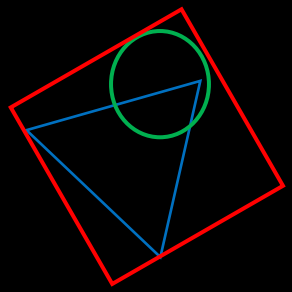
Note that most pixels are clipped at 255 in all channels. This results to a whitewashed image.



Gray World (version 2)

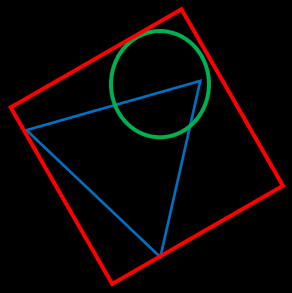
Original

Vintage faded photograph of two men. England circa 1890. By duncan1890 from istockphoto.com



Gray World (version 2)

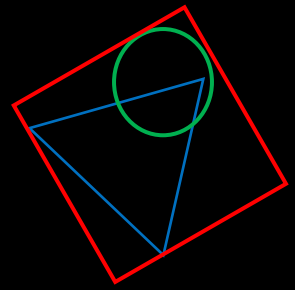
Processed



Gray World (version 2)

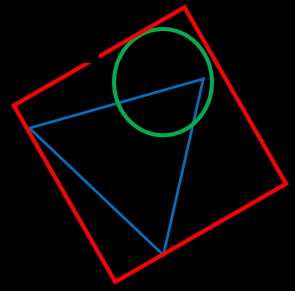
Original

New



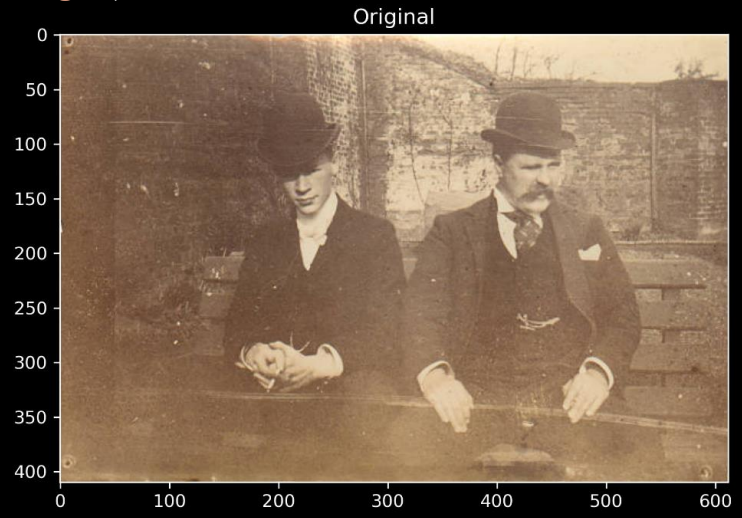
Gray World (version 2)

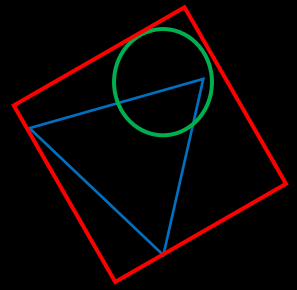
Photo of a room. Obtained from
http://alumni.media.mit.edu/~wad/color/exp1/newgray/



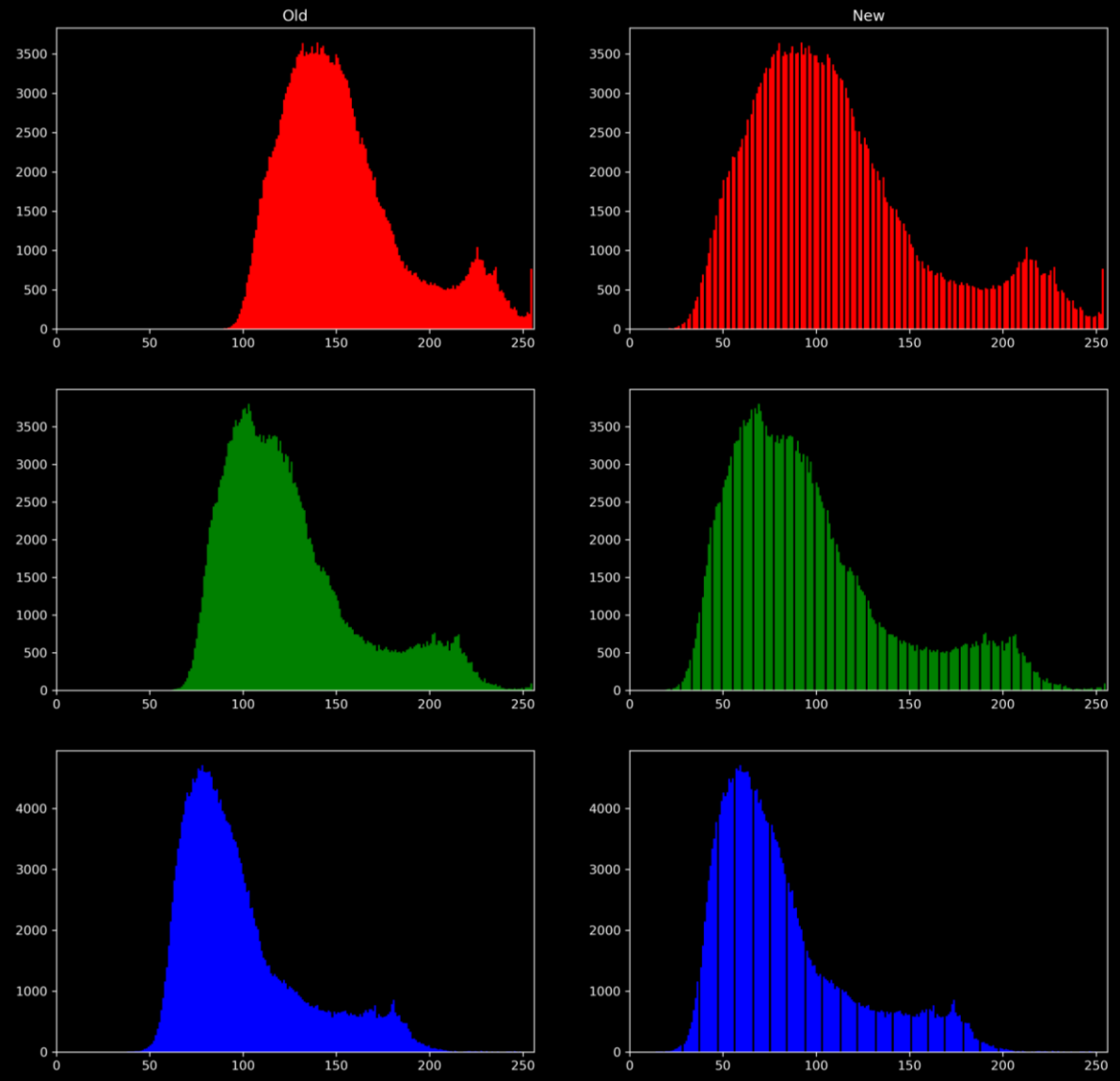
Gray World (version 2)

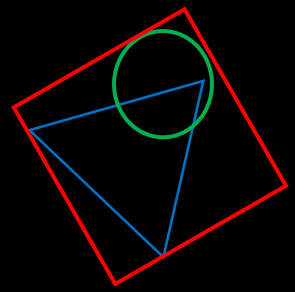
```
def gray_world1(img): #Values below and above 255 aren't clipped; rather, the resulting array is normalized to 0 to
255.
    Rave, Gave, Bave = img.mean(axis=(0,1))[:3]
    new = np.copy(img)
    new = np.array(new, dtype='float')
    new[...,0], new[...,1], new[...,2] = img[...,0]/Rave, img[...,1]/Gave, img[...,2]/Bave
    new[...,:]= new[...,:]*255
    new[new>255] = 255
    return np.array(new, dtype='uint8')

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,15))
ax1.imshow(victorian)
ax1.set_title('Old image')
victorian_img3 = gray_world1(victorian)
ax2.imshow(victorian_img3)
ax2.set_title('New image')
```





Gray World (version 2)



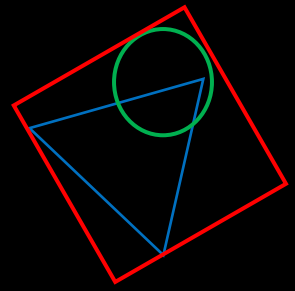


Analysis

1. The second version "re-normalizes" the divided array into the range [0, 255].
Nothing much will happen if the range of the RGB channels already spans this full range.

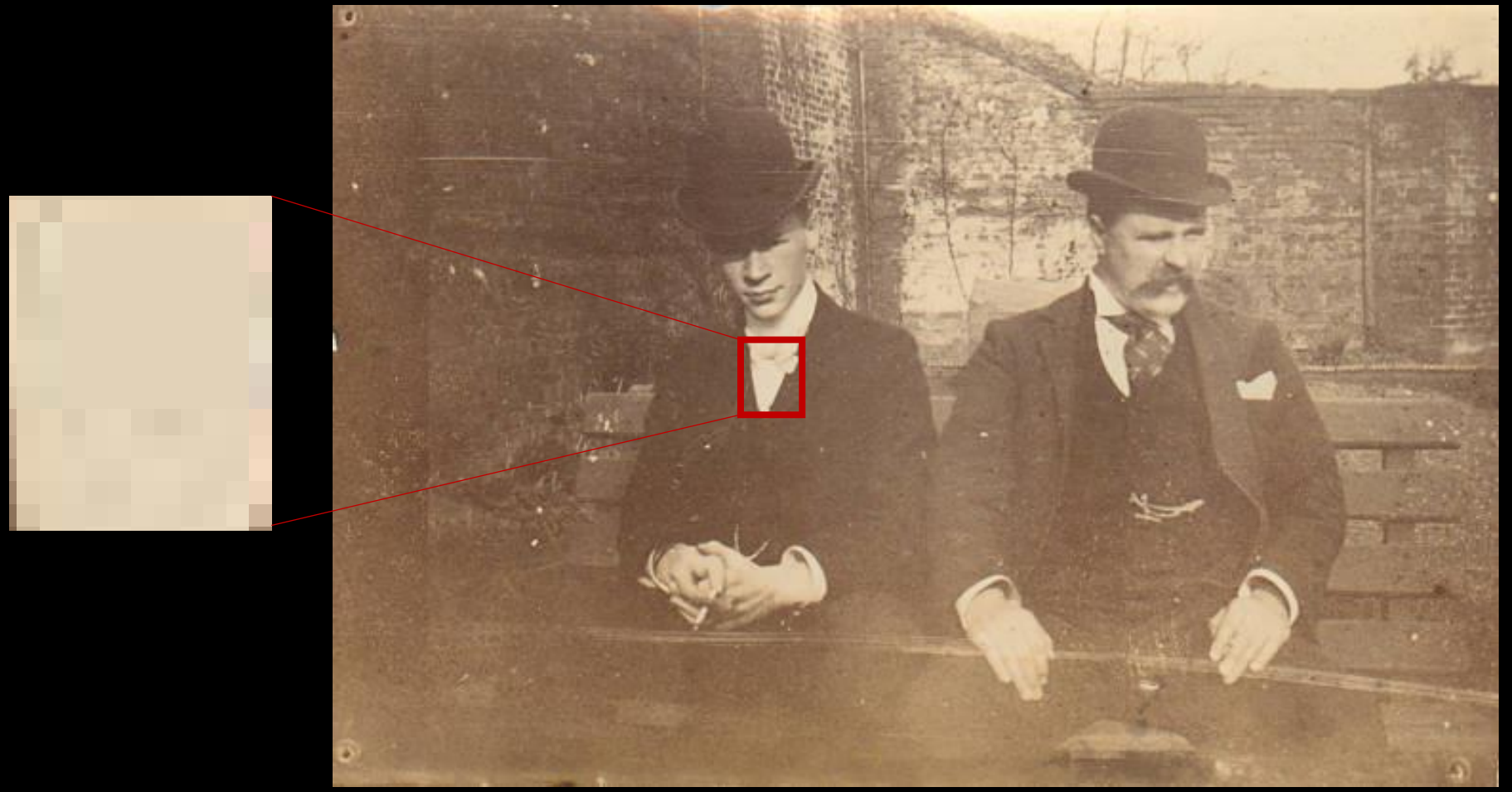
2. If there are a large number of pixels whose RGB intensities are above the "gray" average (in this case, the mean of the distribution), then these values will get clipped at 255 (white), resulting in a bright, whitish image.

3. For this set of images, the gray world algorithm tends to brighten the intensity compared to the RGB stretch (at the specified percentiles). This indicates that there are many pixels with intensities above the gray average.

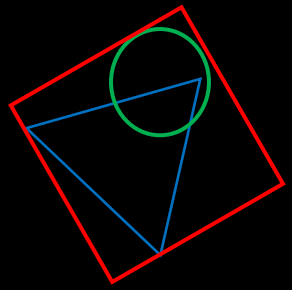


White Patch Algorithm

Original



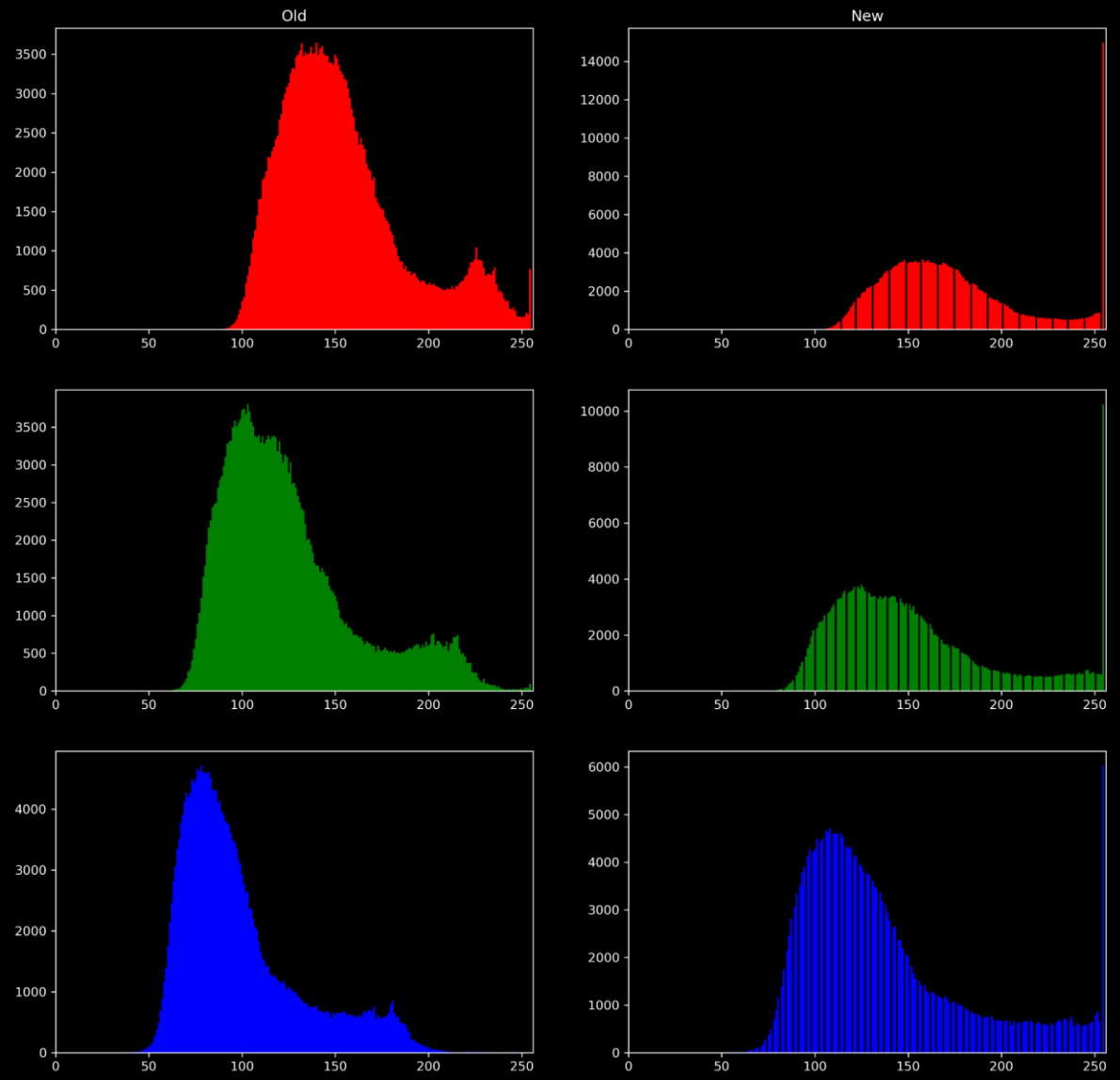
Vintage faded photograph of two men. England circa 1890. By duncan1890 from istockphoto.com

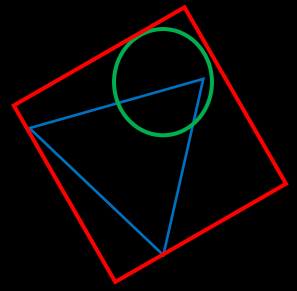


White Patch Algorithm

New

White Patch Algorithm

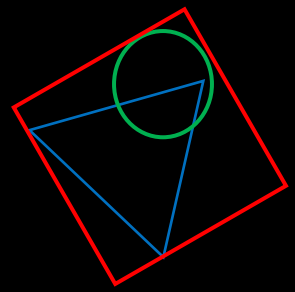




White Patch Algorithm

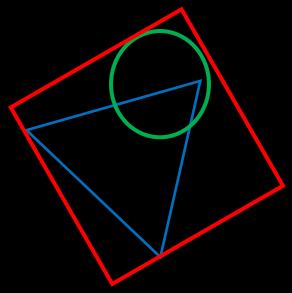
Original

Photo of a room. Obtained from
http://alumni.media.mit.edu/~wad/color/exp1/newgray/

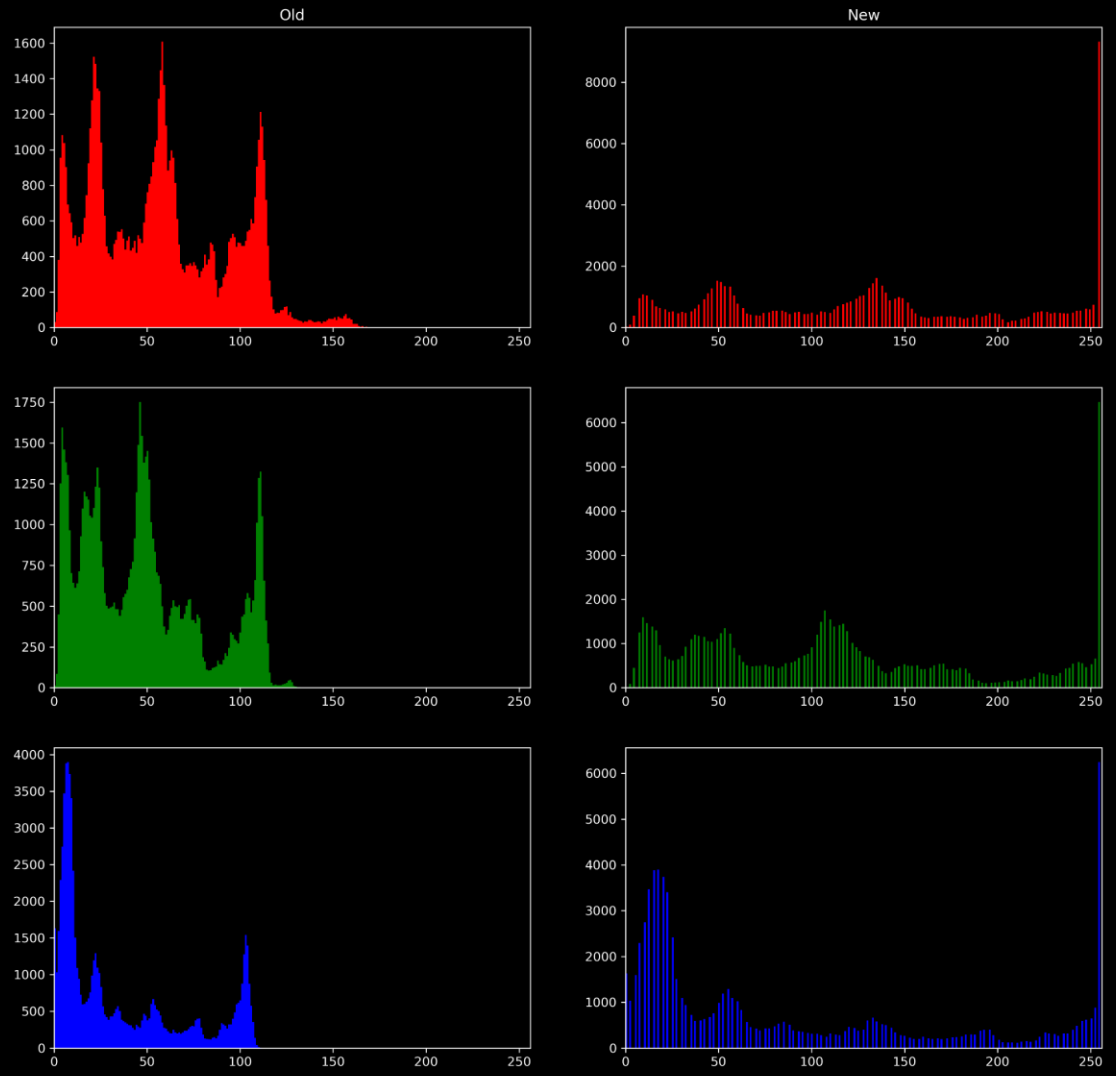


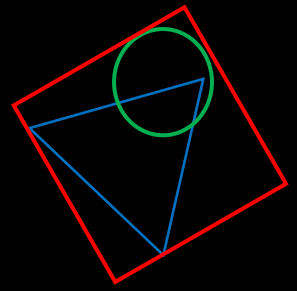
White Patch Algorithm

New



White Patch Algorithm



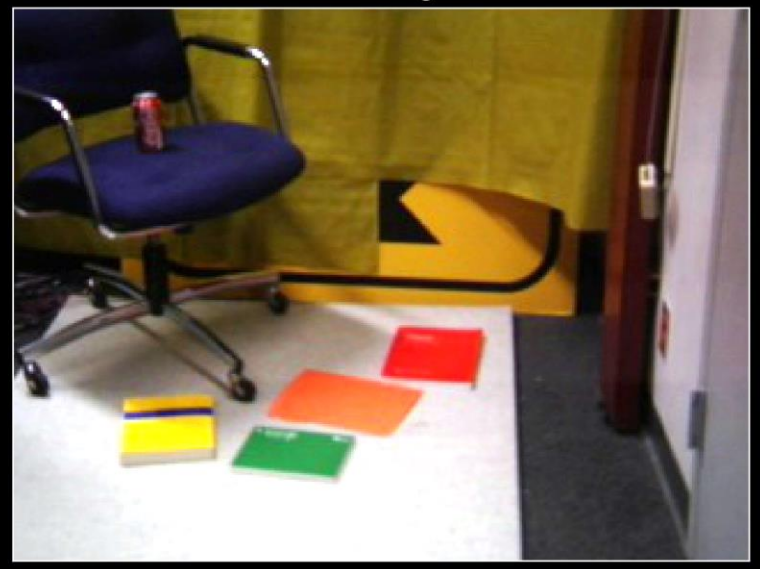


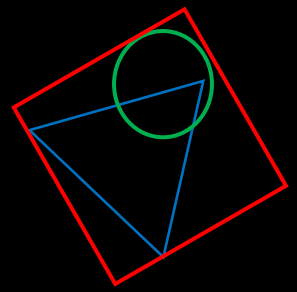
White Patch Algorithm

```
def whitePatch(img, patch):
    patch = np.array(patch[:,:,0:3]*255, dtype='int')
    Rave, Gave, Bave = patch.mean(axis=(0,1))[:3]
    new = np.copy(img)
    new = np.array(new, dtype='float')
    new[...,0], new[...,1], new[...,2] = img[...,0]/Rave, img[...,1]/Gave, img[...,2]/Bave
    new[...,:]= new[...,:]*255
    new[new>255] = 255
    return np.array(new, dtype='uint8')
```

Old image

New image



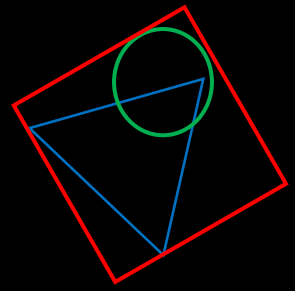


Analysis

1. This is identical to the gray world algorithm, except that the "gray" RGB value is obtained by sampling a white patch, taking the average intensity, and using this as a factor to white-balance the image.

2. Works really well if there is a large white patch in the photograph (larger sample size to white-balance the entire image)

3. Will only work if there are white patches in the photos (obviously!)



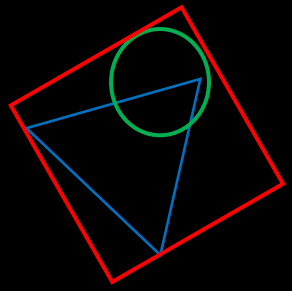
Comparison of Algorithms

The RGB percentile contrast stretch provides more flexibility in finetuning the maximum and minimum pixel intensities. The brightness of the lava column is more subdued in the RGB stretch algorithm compared to the gray-world algorithm.

RGB contrast stretch
(not too bright, looks playable lol)

Gray world
(too bright, hurts the eyes)

Original



Comparison of Algorithms

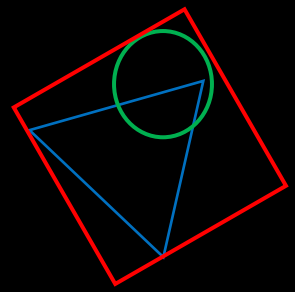
The same issue is observed for the gray world restoration (image becomes too bright). The RGB contrast stretch provides the best restoration.

RGB contrast stretch

Gray world

White Patch

Original



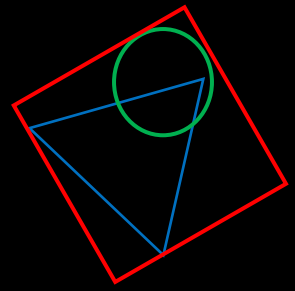
Self-Reflection

Pitfalls and Struggles I've encountered

1. Applying the correct transformation formula for a function with range [a,b] to a new function with range [c,d] with correct translation and scaling constants.
2. Correctly performing coordinate bashing on the hexagon coordinates. This is a usual exercise for competitive mathematics in high school but I was getting rusty with it; still, it was a fun exercise!
3. In this module I learned that you cannot export into a .bmp extension a matplotlib figure directly. I had to use Pillow library (import PIL) as a workaround.
4. Be careful of re-normalizing an array after contrast stretching and performing gray world and white patch algorithms! (see Gray World ver. 2). Re-normalizing to [0,255] an intensity array that was originally divided by a constant restores the original histogram save for some rounding errors.
5. Regarding Point 4, this is why we tend to clip values instead. Careful though, as too low an intensity maximum for the contrast stretch will result to too bright an image, and too high an intensity minimum will result to a very dark image.

Score I'll give myself: 105/100

Reason: I worked really hard to streamline my code (trying to make use of list comprehensions and numpy vectorization as much as possible), and I experimented with different photographs and examined the intensity clipping thoroughly. Above all else, I had fun!



References

Main Python Libraries Used

J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007. (matplotlib documentation)

Clark, A. (2015). *Pillow (PIL Fork) Documentation*. readthedocs. Retrieved from https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

Stack Exchange Websites

https://stackoverflow.com/questions/43971259/how-to-draw-polygons-with-python (Shrish's answer)

Photos

Photo of a room. Obtained from http://alumni.media.mit.edu/~wad/color/exp1/newgray/

Vintage faded photograph of two men. England circa 1890. By duncan1890 from istockphoto.com

"Towards the City" by Yutaka Takanashi (1974)

Minecraft Shader

https://minecraftshader.com/seus-ptgi-shaders/