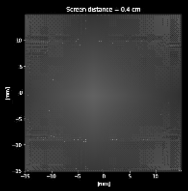
Digital Image Formation and Enhancement

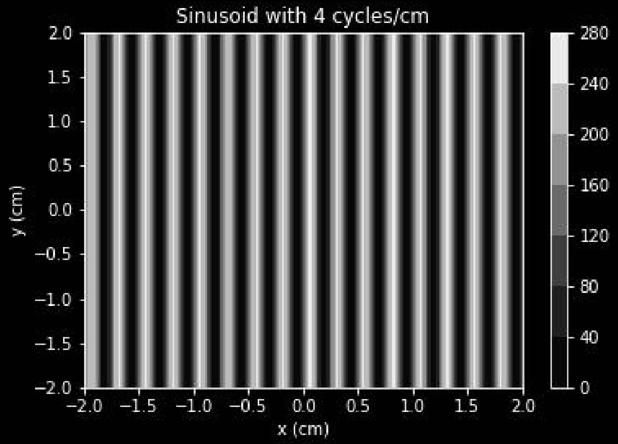
digital-eyes:~# A plethora optical information remains hidden from our eyesight. Here, we explore them using digital vision.

!WARNING! The following presentation contains flashing lights and fast-moving visuals that may trigger seizures or cause discomfort to individuals with photosensitive epilepsy. Viewer discretion is advised.

GitHub repo: https://github.com/schwarzschlyle/image-processing



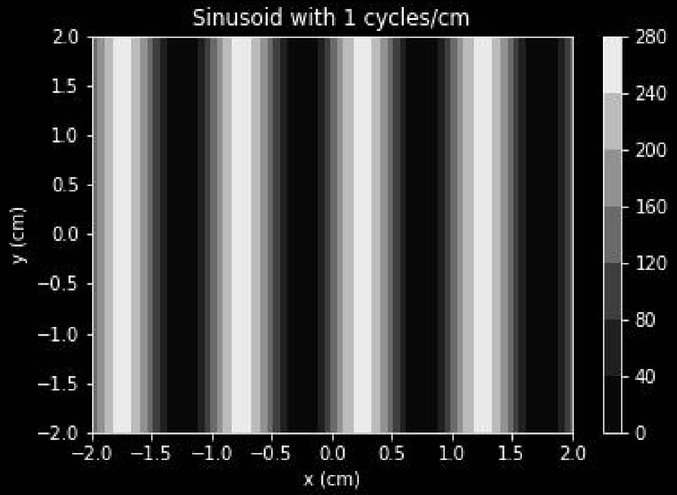
discussion-format:~# Throughout the exploration, image processing is first demonstrated with the corresponding code on a separate slide image-synthesis:~# Sinusoidal synthetic images are one of the most fundamental building blocks of any image. Similar to the decomposability of any function to a superposition of sines and cosines, images can be likewise decomposed to a sum of sine and cosine images. This can be taken advantages in compression, edge detection, and analysis which requires or can be aided by analyzing each frequency modes usually with the aid of a Fourier transform.



```
plot_sine(-2,2,-2,2, 100, 4,'Greys')
```

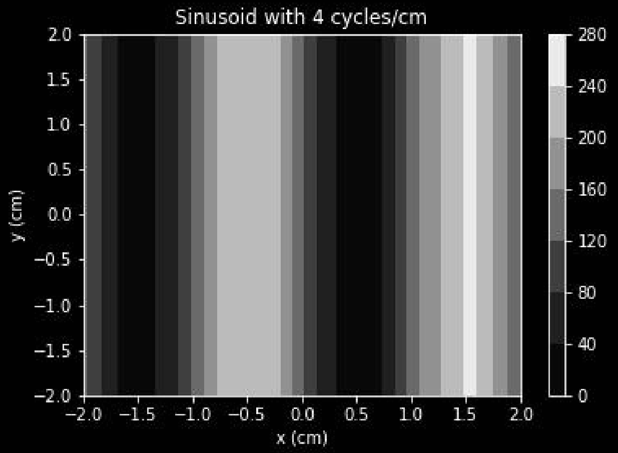
The constructed function takes in the meshgrid dimension, the number of total points per coordinate, the frequency of the sine wave, and the color scheme used to display the 0 to 255 pixel values. Here, we display a sinusoidal image with a frequency of 4 cycles per cm.

image-synthesis:~# Sinusoidal synthetic images are one of the most fundamental building blocks of any image. Similar to the decomposability of any function to a superposition of sines and cosines, images can be likewise decomposed to a sum of sine and cosine images. This can be taken advantages in compression, edge detection, and analysis which requires or can be aided by analyzing each frequency modes usually with the aid of a Fourier transform.



```
for i in range(1,10,1):
    plot_sine(-2,2,-2,2, 100, i,'Greys')
```

The rendered simulation sweeps through different frequency sine wave images from i=1 to i=9. Observe that, obviously, higher frequency sine wave images correspond to thinner lines.



```
for i in range(1,10,1):
    plot_sine(-2,2,-2,2, 5*i, 4,'Greys')
```

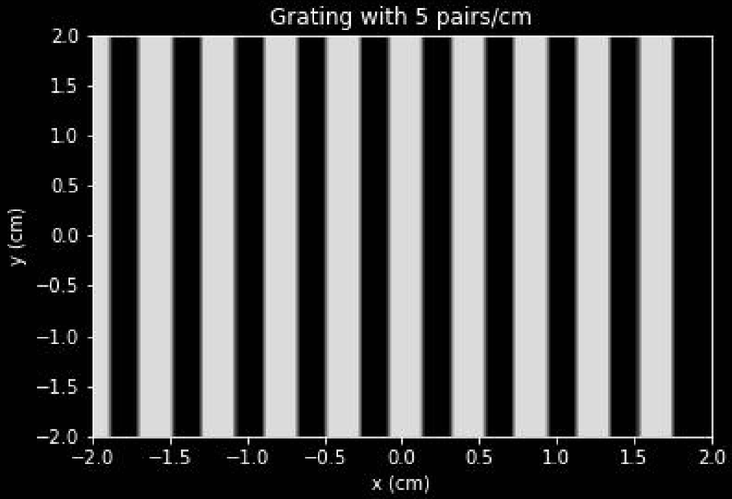
Moreover, sweeping through different coordinate points at a fixed frequency of 4 cycles per second, one can observe the importance of resolution in terms of the number of points to appropriately synthesize an image.

```
Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

image-synthesis:~# code it yourself!
```

```python
def plot_sine(x_min, x_max, y_min, y_max, resolution, freq,  contour_color):
    """
    User defined function to plot a sine wave along a custom mesh

    Parameters:
    x_min (float): minimum grid point x-coordinate
    x_max (float): minimum grid point x-coordinate
    y_min (float): minimum grid point x-coordinate
    y_max (float): minimum grid point x-coordinate
    resolution (int): number of grid points in each coordinate
    freq (float): frequency of the sine wave which will be scaled into angular frequency
    contour_color (str): color theme of the contour
        available color maps:
            viridis
            plasma
            inferno
            magma
            cividis
            Greys
            Purples
            Blues
            Greens
            Oranges
            Reds
            coolwarm
            seismic
            jet

    Try experimenting effects of varying these parameters.

    Author:
    Lyle Kenneth Geraldez
    """
```

```python
# Mesh grid creation
# First two lines generate an array of values for x and for y
# which are equally spaced according to the set resolution
# Final line generates a mesh grid using these generated arrays

x = np.linspace(x_min, x_max, resolution)
y = np.linspace(y_max, y_min, resolution)
X, Y = np.meshgrid(x, y)

# Calculation of sine wave
# Input frequency is converted to the angular frequency by scaling with 2*pi
# Sinusoidal contour (z) values are calculated by taking sine values
# with each coordinate multiplied by the angular frequency

# Note that the domain here is [-1,1]
Z = np.sin(2*np.pi*freq*X)

# We can force it into the pixel domain [0, 255]
Z = (Z + 1) * 127.5

# Plotting sine wave contour
# Mainly for illustrative and aeeshtic purposes
plt.contourf(X, Y, Z, cmap=contour_color)
plt.colorbar()
plt.xlabel('x (cm)')
plt.ylabel('y (cm)')
plt.title('Sinusoid with ' +  str(freq) + ' cycles/cm')
# plt.savefig(f'sine_plot_{freq}.jpg')
# plt.savefig(f'r_sine_plot_{resolution}.jpg')
plt.show()
```

image-synthesis:~# Sometimes, images taken from a screen capture, from a website, or from a camera go through different lossy compression processes. Knowing how to construct synthetic images allow a pixel-precise image construction that is vital in a plethora of applications. This can be crucial in constructing accurate optical simulation by digital means



```
plot_grating(-2, 2, -2, 2, 5)
```

The function generates a uniform grating by fixing the grating interval to be equal to the grating width of each point. Here, the displayed image contains five pairs per cm totalling 10 pairs along the 2 cm by 2 cm grid.

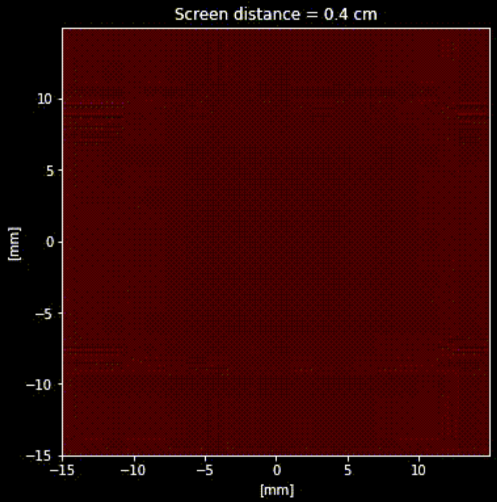
Although it may seem like a rookie exercise in pixel-precise image synthesis via code, the discussion shows different forms of applications such as digital holography simulations and wave propagation simulations.

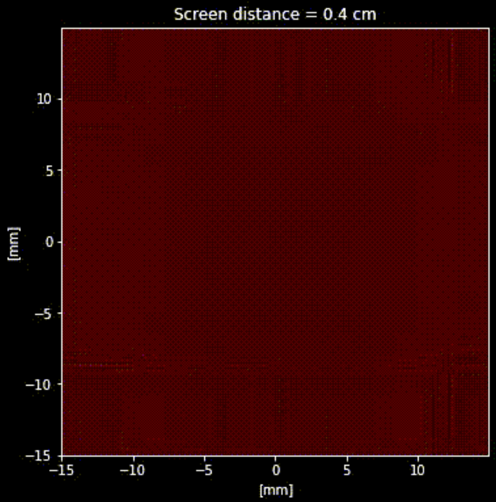
```
Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

image-synthesis:~# code it yourself!
```

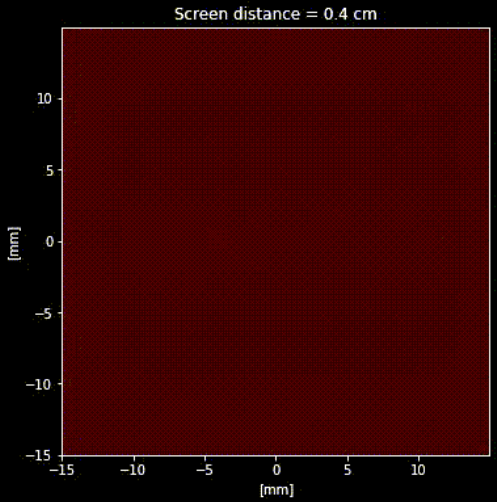
```python
def plot_grating(x_min, x_max, y_min, y_max, pairs_per_cm):
    """
    User defined function to plot a sine wave along a custom mesh

    Parameters:
    x_min (float): minimum grid point x-coordinate
    x_max (float): minimum grid point x-coordinate
    y_min (float): minimum grid point x-coordinate
    y_max (float): minimum grid point x-coordinate
    resolution (int): number of grid points in each coordinate

    Author:
    Lyle Kenneth Geraldez
    """
    # Mesh grid creation
    # First two lines generate an array of values for x and for y
    # which are equally spaced according to the set resolution
    # Final line generates a mesh grid using these generated arrays

    resolution = 100

    x = np.linspace(x_min, x_max, resolution)
    y = np.linspace(y_max, y_min, resolution)
    X, Y = np.meshgrid(x, y)

    Z = np.zeros((resolution, resolution))

    # set values of Z to 1 every 0.1 cm

    pair_width = int(resolution/(1.95*pairs_per_cm))

    Z[:, np.arange(0, resolution, pair_width)] = 1


    plt.contour(X, Y, Z, cmap='binary', linewidths=pair_width+5)
    plt.xlabel('x (cm)')
    plt.ylabel('y (cm)')
    plt.title('Grating with ' +  str(pairs_per_cm) + ' pairs/cm')
    plt.savefig(f'r_sine_plot_{pairs_per_cm}.jpg')
    plt.show()
```

Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

image-synthesis:~# Pixel-precise image synthesis is crucial in applied optics simulations such as with digital phase retrieval in a holography simulation or with digital wave propagation using the angular spectrum method as accurate pixel-to-intensity mapping is needed to properly conform with the governing equations.



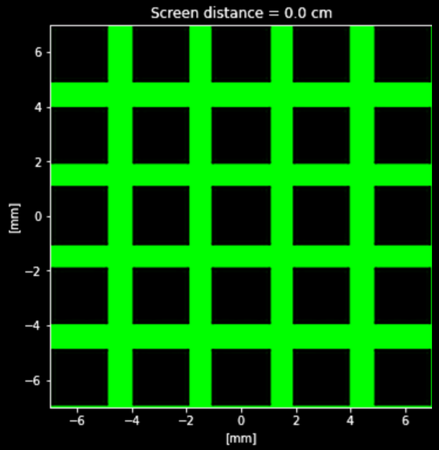




Top simulations show digital holographic phase retrieval in the red wavelength using different optimization algorithms: from left to right, adam optimizer, conjugate gradient optimizer, and stochastic gradient descent optimizer

The right simulation shows wave propagation in the green wavelength is it diffracts through a uniform grid slit. This is done using the angular spectrum method of wave propagation.

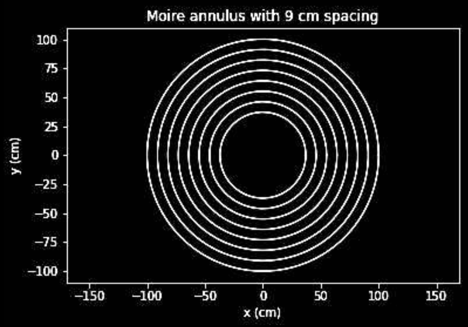
The simulation codes and report can be seen on the following GitHub repository.



Applied optics GitHub repo: https://github.com/schwarzschlyle/applied-optics

Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

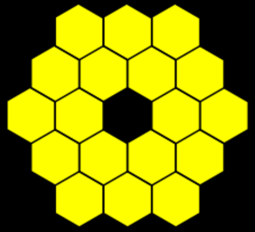
image-synthesis:~# Here are a few more synthesized images. Of course in most applications, one must tap into the RGB color channels to shade each pixel with a desired color.



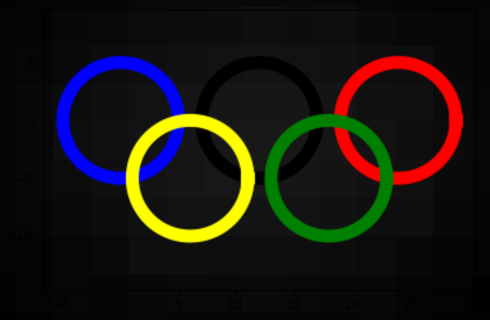
```
for i in range (1,10,1):
    draw_annulus((0,0),100.1, 70, 10-i, 1)
```

With proper code construction, one can synthesize an annulus (even with a Moire effect) to synthesize the Hubble primary mirror,

```
draw_hex_mirror()
```



... or draw a custom hexagonal array to synthesize the JWST primary mirror,



```
plot_circle(0,0,5,'blue')
plot_circle(12,0,5,'black')
plot_circle(24,0,5,'red')
plot_circle(6,-5,5,'yellow')
plot_circle(18,-5,5,'green')
```

... or draw connected circles to synthesize the Olympics logo.

```
Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

image-synthesis:~# code it yourself!
```

```python
def draw_annulus(center, outer_radius, inner_radius, spacing, linewidth):
    """
    User defined function to draw an annulus filled to display a Moire effect.
    This is done by filling the annulus with concentric circles.
    Viewed from afar, different emerging patterns can be observed

    Parameters:

    center (tuple): set the center coordinates of the annulus as a tuple
    outer_radius (float): set the outer radius
    inner_radius (float): set the inner radius
    spacing (int): set the spacing of the concentric circles
    linewidth (float): set the line width of concentric circles


    Try experimenting effects of varying these parameters.

    Author:
    Lyle Kenneth Geraldez
    """
```

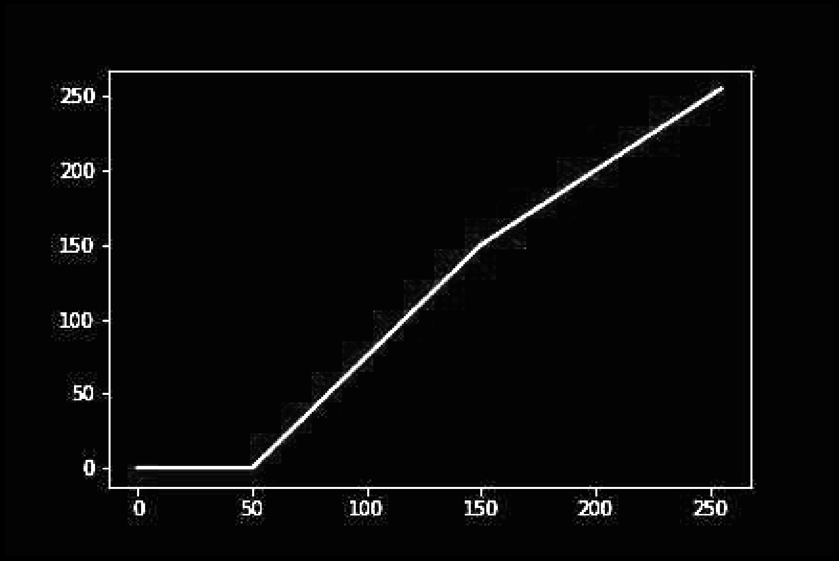
```python
# generate circle angular values
theta = np.linspace(0, 2*np.pi, 1000)


# generate concetric cirlces
for i in range(0,inner_radius,spacing):
    x = (outer_radius-i)*np.cos(theta) + center[0]
    y = (outer_radius-i)*np.sin(theta) + center[1]
    plt.plot(x, y, 'black', linewidth = 1)

# plot the generated concentric circles
plt.xlabel('x (cm)')
plt.ylabel('y (cm)')
plt.title('Moire annulus with ' +  str(spacing) + ' cm spacing')
plt.axis('equal')
plt.savefig(f'r_annulus_plot_{10-spacing}.jpg')
plt.show()
```

```
Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

image-synthesis:~# code it yourself!
```

```python
def draw_hex_mirror():
    """
    User defined function of (inverted) JWST primary mirror without variable paramete

    Author:
    Lyle Kenneth Geraldez
    """
    fig, ax = plt.subplots()
    for i in range(3,6):
        ax.plot(x+i*x_offset, y, 'k')
        ax.fill(x+i*x_offset, y, 'yellow')

    for i in range(2,6):
        ax.plot(x+i*x_offset+0.5*x_offset, y-y_offset, 'k')
        ax.fill(x+i*x_offset+0.5*x_offset, y-y_offset, 'yellow')

    for i in range(2,7):
        if i == 4:
            continue
        ax.plot(x+i*x_offset, y-(2*y_offset), 'k')
        ax.fill(x+i*x_offset, y-(2*y_offset), 'yellow')

    for i in range(2,6):
        ax.plot(x+i*x_offset+0.5*x_offset, y-(3*y_offset), 'k')
        ax.fill(x+i*x_offset+0.5*x_offset, y-(3*y_offset), 'yellow')

    for i in range(3,6):
        ax.plot(x+i*x_offset, y-(4*y_offset), 'k')
        ax.fill(x+i*x_offset, y-(4*y_offset), 'yellow')

    ax.set_aspect('equal')
    ax.axis('off')
    fig.savefig('hex_mirror.png', transparent=True)
    plt.show()
```

```python
def plot_circle(center_x, center_y, radius, color):
    """
    User defined function of (inverted) JWST primary mirror without variable parameters.

    Parameters:
    center_x (float): set the x coordinate of the circle's center
    center_y (float): set the y coordinate of the circle's center
    radis (float): set the radius of the circle
    color (int): set the color of the circle

    Author:
    Lyle Kenneth Geraldez
    """
    theta = np.linspace(0, 2 * np.pi, 1000)
    x = radius * np.cos(theta) + center_x
    y = radius * np.sin(theta) + center_y
    plt.plot(x, y, color=color, linewidth=10)
    plt.savefig('olympics.png', transparent=True)
    plt.axis('equal')
```

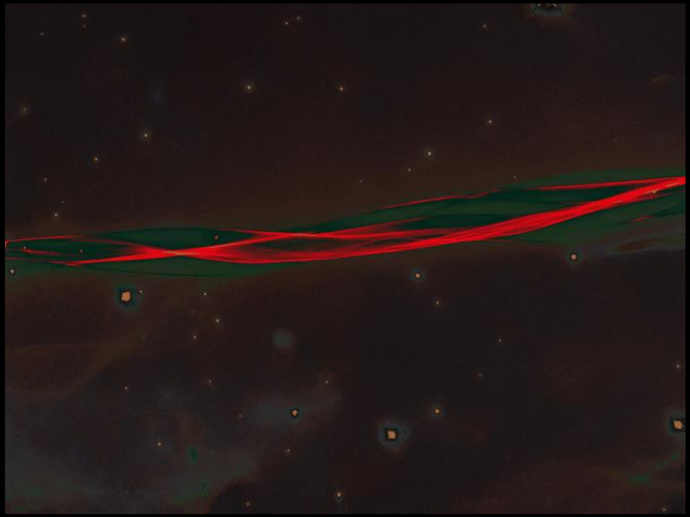
color-curve:~# Aside from the RGB pixel description of an image, one can also use an HSV (hue, saturation, value) description respectively describing pixel color, purity, and intensity. One way to enhance an image is by modifying its value channel which can be described by changes in its io curve.

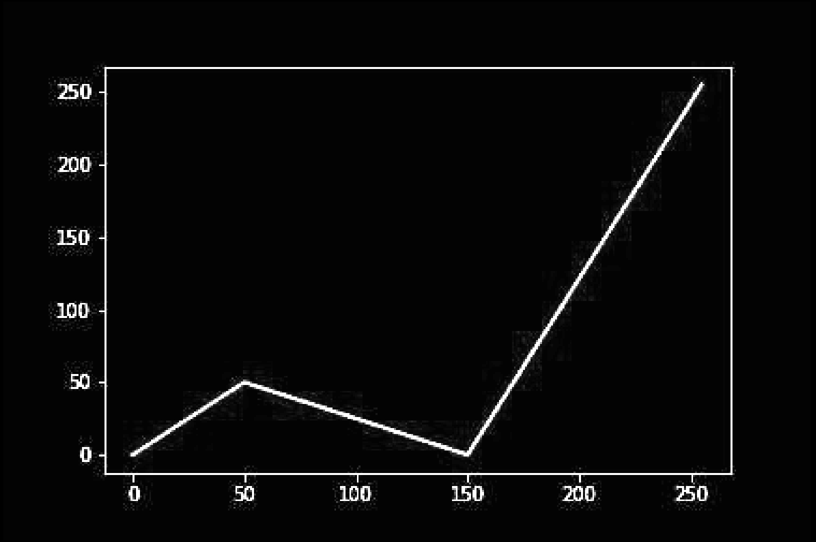


```
for i in range(0,10,1):
    alter_io('dark_galaxy.jpg', i*50, 150, 255)
```

Mimicking GIMP capability in python using three-point interpolated lookup table, this code varies the value channel by modifying the value = 50 pixel elements according to the for loop on the left.

color-curve:~# Aside from the RGB pixel description of an image, one can also use an HSV (hue, saturation, value) description respectively describing pixel color, purity, and intensity. One way to enhance an image is by modifying its value channel which can be described by changes in its io curve.

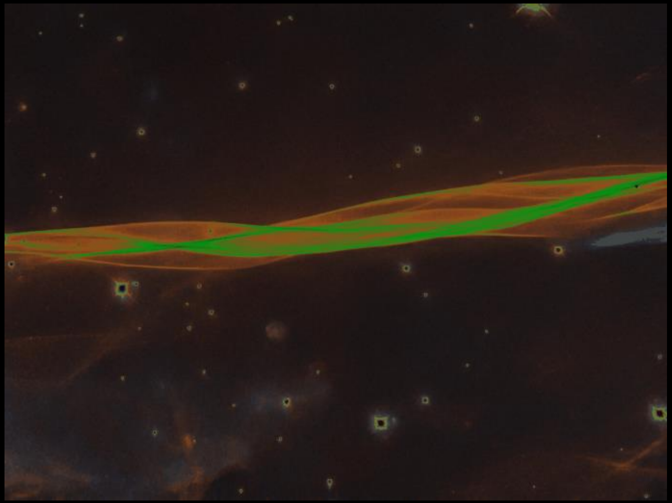


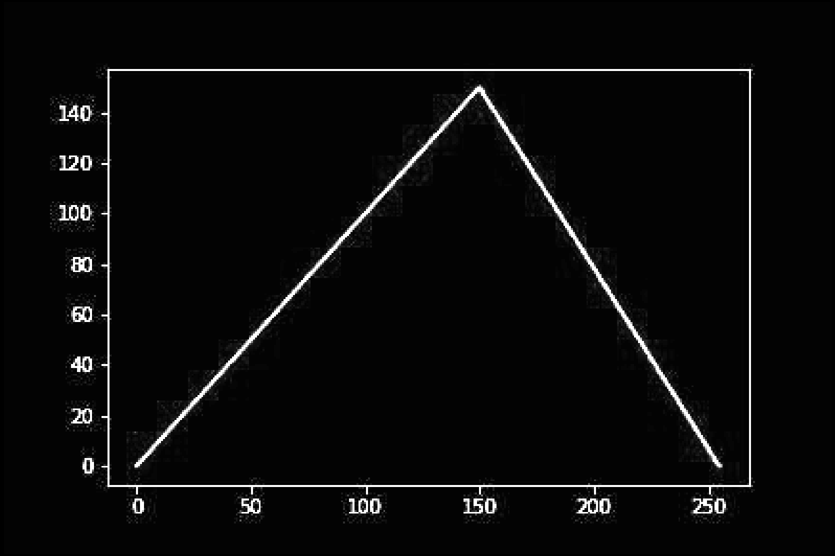


```
for i in range(0,10,1):
    alter_io('dark_galaxy.jpg', 50, i*50, 255)
```

Mimicking GIMP capability in python using three-point interpolated lookup table, this code varies the value channel by modifying the value = 150 pixel elements according to the for loop on the left.

color-curve:~# Aside from the RGB pixel description of an image, one can also use an HSV (hue, saturation, value) description respectively describing pixel color, purity, and intensity. One way to enhance an image is by modifying its value channel which can be described by changes in its io curve.

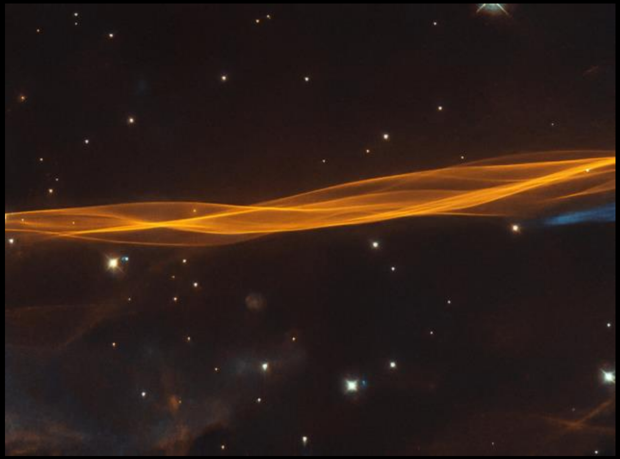


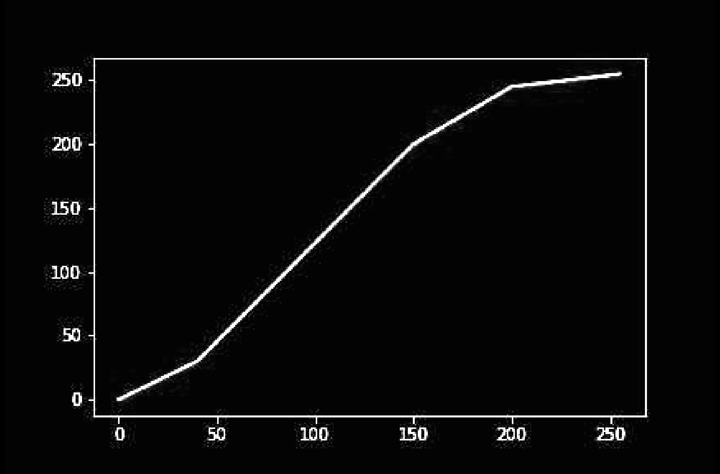


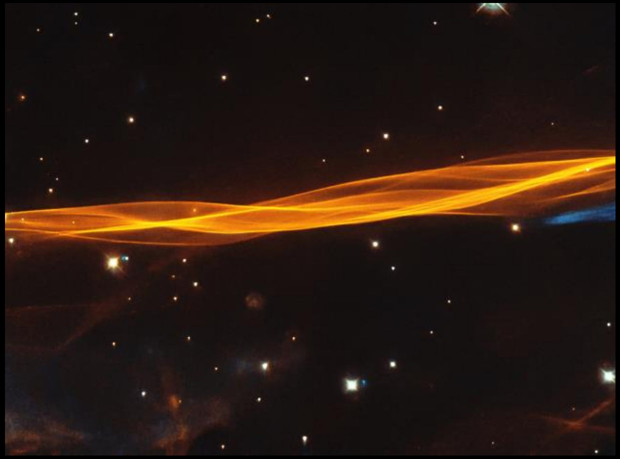
```python
for i in range(0,10,1):
    alter_io('dark_galaxy.jpg', 50, 150, i*50)
```

Mimicking GIMP capability in python using three-point interpolated lookup table, this code varies the value channel by modifying the value = 255 pixel elements according to the for loop on the left.

color-curve:~# Aside from the RGB pixel description of an image, one can also use an HSV (hue, saturation, value) description respectively describing pixel color, purity, and intensity. One way to enhance an image is by modifying its value channel which can be described by changes in its io curve.







```
input_curve = np.array([0, 40, 150, 200, 255])
output_curve = np.array([0, 30, 200, 245, 255])
```

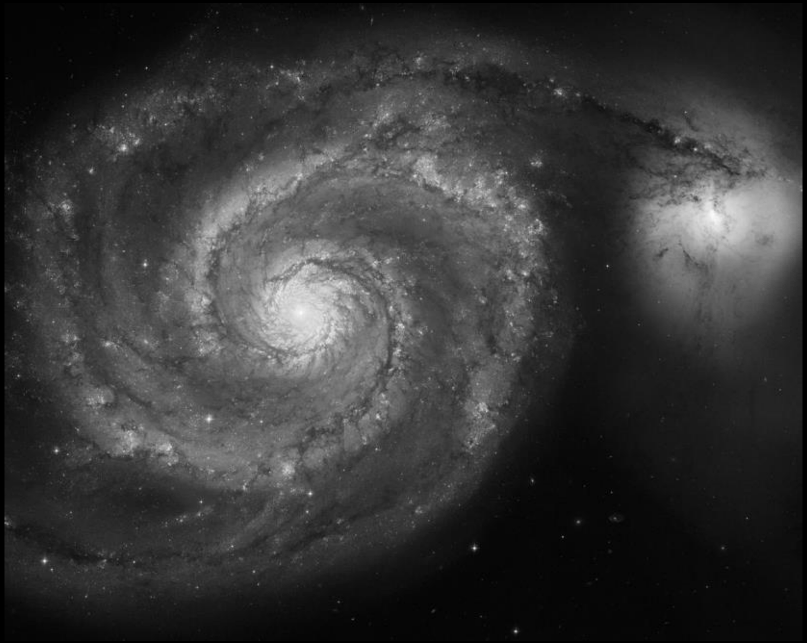
From a series of numerical experimentation, the above five-point interpolation lookup table transformed to the image reveals more feature prominence of the now-crispier galaxy.

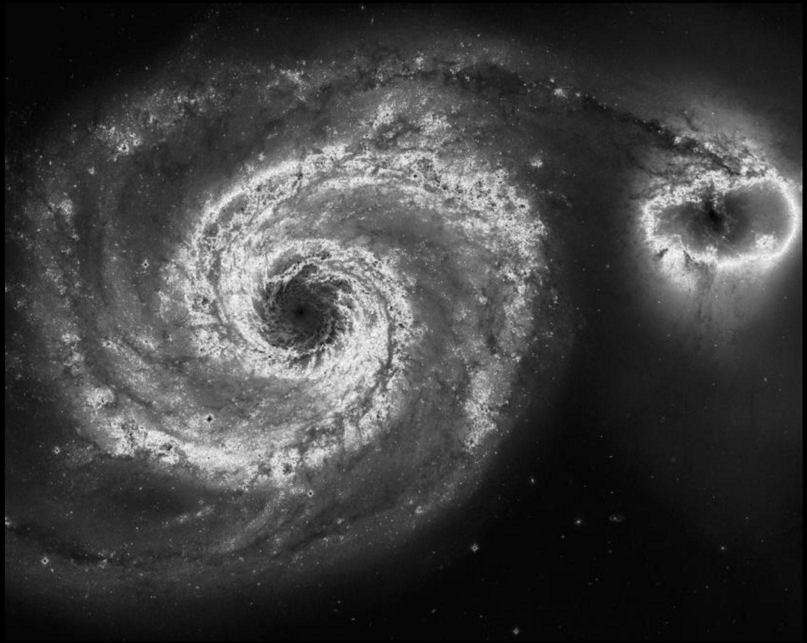
```
Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

color-curve:~# code it yourself!
```

```python
def alter_io(path, point1, point2, point3):
    """
    User defined function to alter the input-ouput color curve of an image

    Parameters:
    path (string): specify the image path
    point1: specify the first point output value of the three-point interpolated io cur
    point2: specify the second point output value of the three-point interpolated io cu
    point3: specify the third point output value of the three-point interpolated io cur

    This mimics GIMP's feature of varying the io curve. Here, however, we only restrict
    to the three-point LUT interpolation

    Author:
    Lyle Kenneth Geraldez
    """
    # Load an image
    img = cv2.imread(path)

    # Define the input and output curves
    input_curve = np.array([0, 50, 150, 255])
    output_curve = np.array([0, point1, point2, point3])
```

```python
    # Create the lookup table using linear interpolation
    lut = np.interp(np.arange(256), input_curve, output_curve).astype(np.uint8)

    # Plot the input and output curves
    fig, ax = plt.subplots()
    ax.plot(input_curve, output_curve, label='Input-Output Curve')

    # Plot the LUT curve
    ax.plot(np.arange(256), lut, label='LUT Curve')
    plt.savefig(f'lut_point1_{point1}.jpg')

    # Apply the lookup table to the image
    img_processed = cv2.LUT(img, lut)

    # Uncomment to save the output image
    # cv2.imwrite(f'io_point1_{point1}.jpg', img_processed)
    # cv2.imwrite(f'io_point2_{point2}.jpg', img_processed)
    # cv2.imwrite(f'io_point3_{point3}.jpg', img_processed)
```

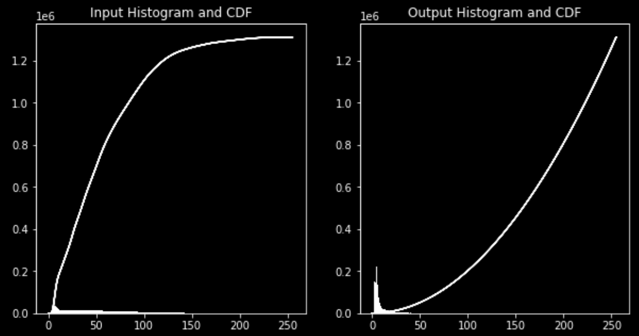
histogram-manipulation:~# Another way of image enhancement is through CDF backprojection. From a given histogram, one can calculate the PDF, and then the normalized CDF of an image. The calculated CDF can be transformed to another CDF shape via a lookup table.



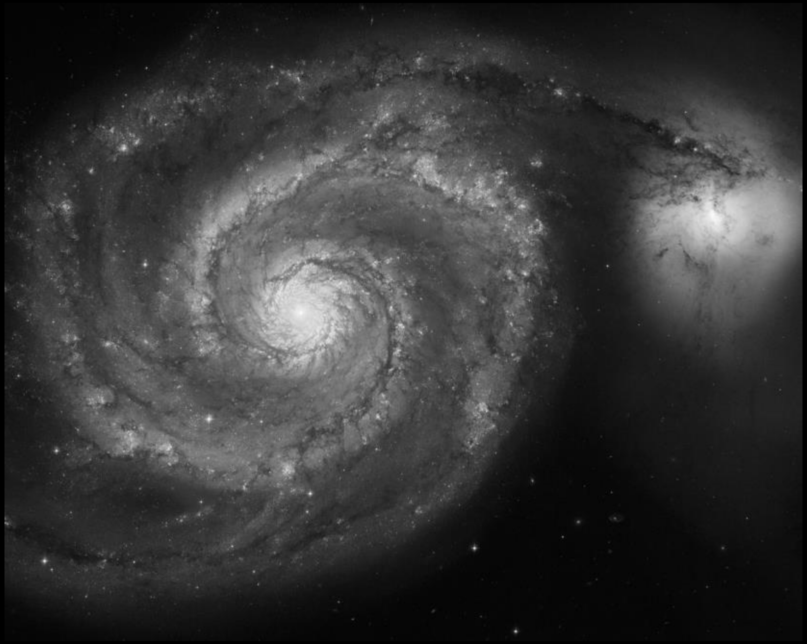


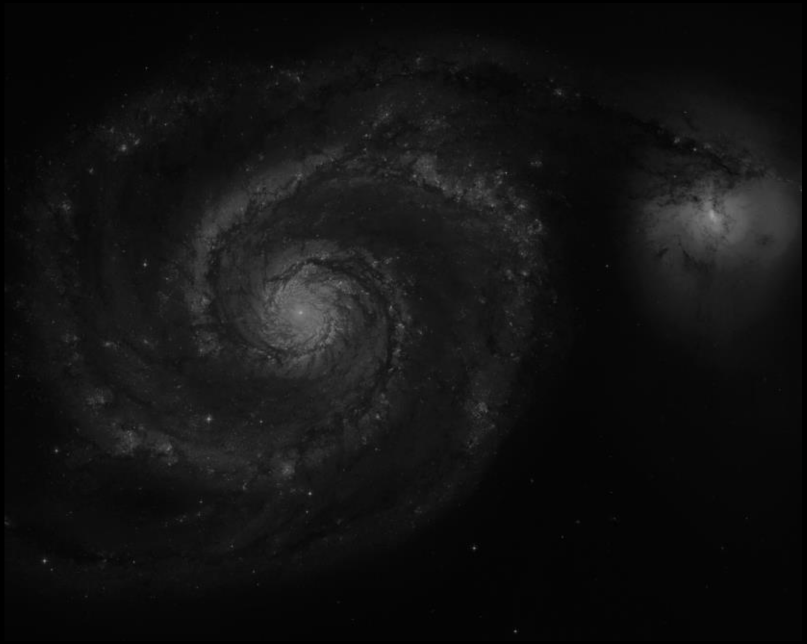
```
x = np.linspace(0, 1, 256)
desired_cdf = x ** 2
modify_cdf('dark_whirlpool.jpg', desired_cdf)
```

User defined function of CDF backprojection takes in the image and the desired CDF as parameters. Here, we use an exponential CDF. Observe that the transformation increased the contrast and transformed the galaxy image to emphasize some features.



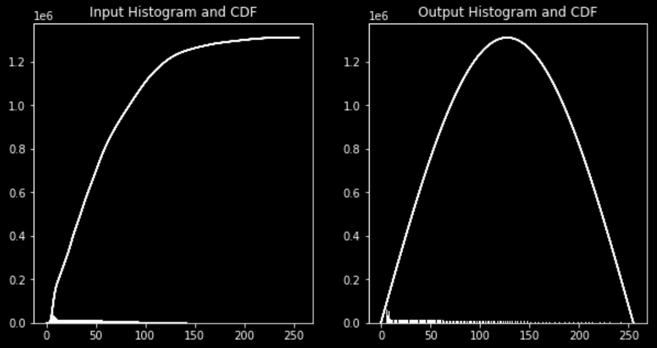
histogram-manipulation:~# Another way of image enhancement is through CDF backprojection. From a given histogram, one can calculate the PDF, and then the normalized CDF of an image. The calculated CDF can be transformed to another CDF shape via a lookup table.



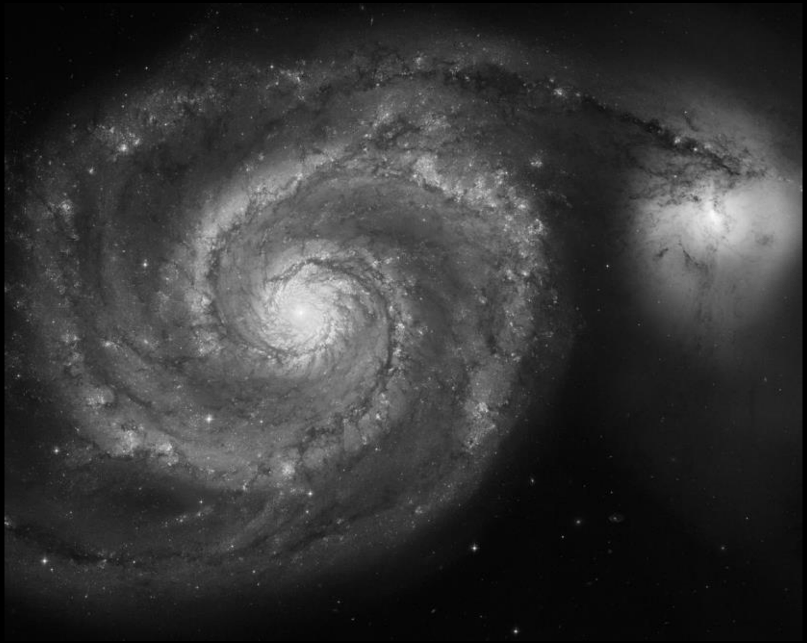


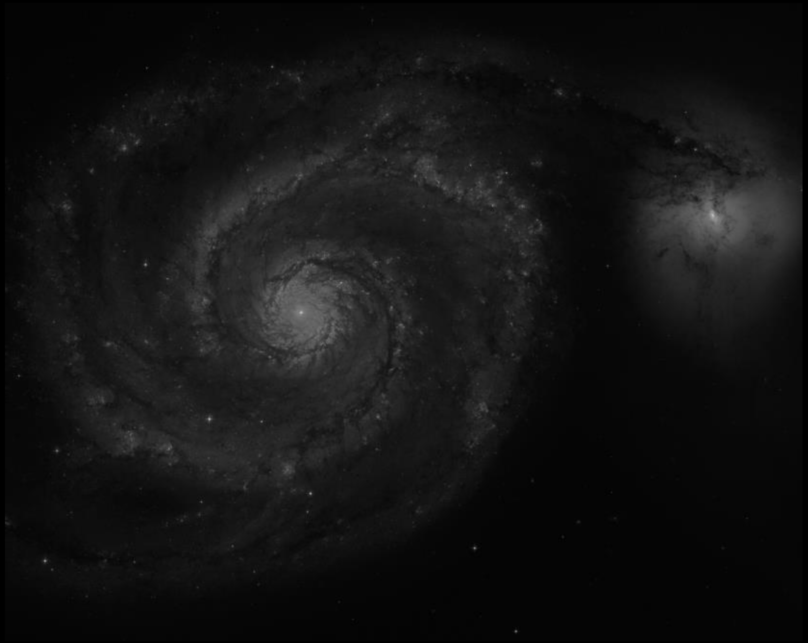
```
x = np.linspace(0, np.pi, 256)
desired_cdf = np.sin(x)
modify_cdf('dark_whirlpool.jpg', desired_cdf)
```

Applying a sinusoidal CDF decreased the intensity and the contrast of the transformed image.

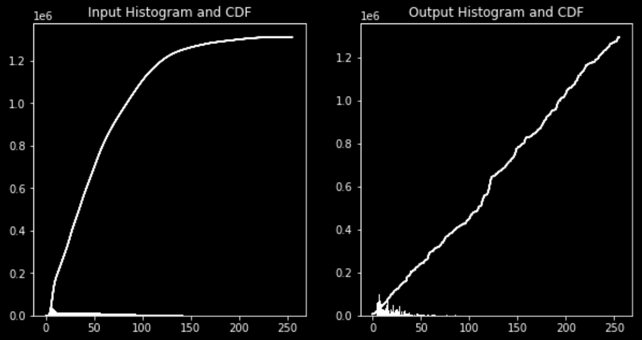


histogram-manipulation:~# Another way of image enhancement is through CDF backprojection. From a given histogram, one can calculate the PDF, and then the normalized CDF of an image. The calculated CDF can be transformed to another CDF shape via a lookup table.





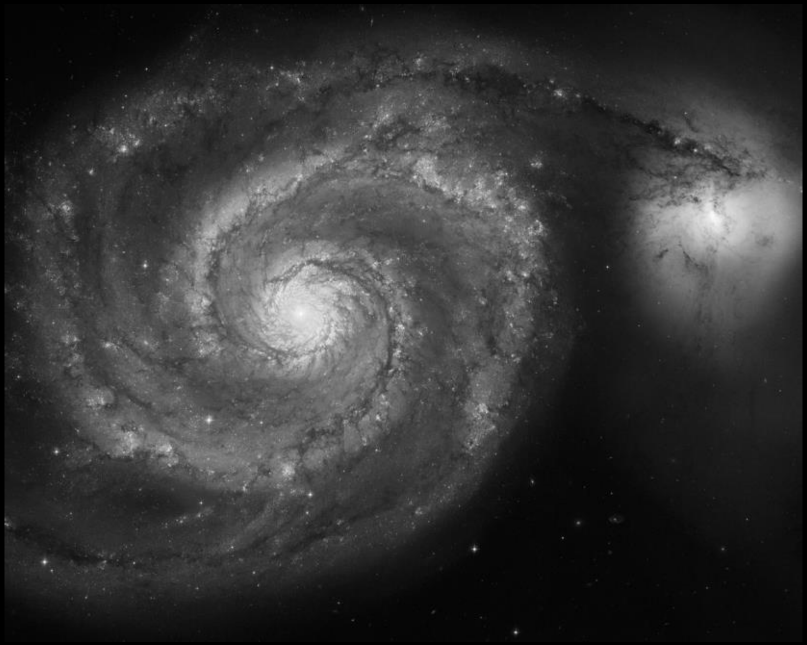
```
np.random.seed(42)
desired_cdf = np.sort(np.random.rand(256))
modify_cdf('dark_whirlpool.jpg', desired_cdf)
```

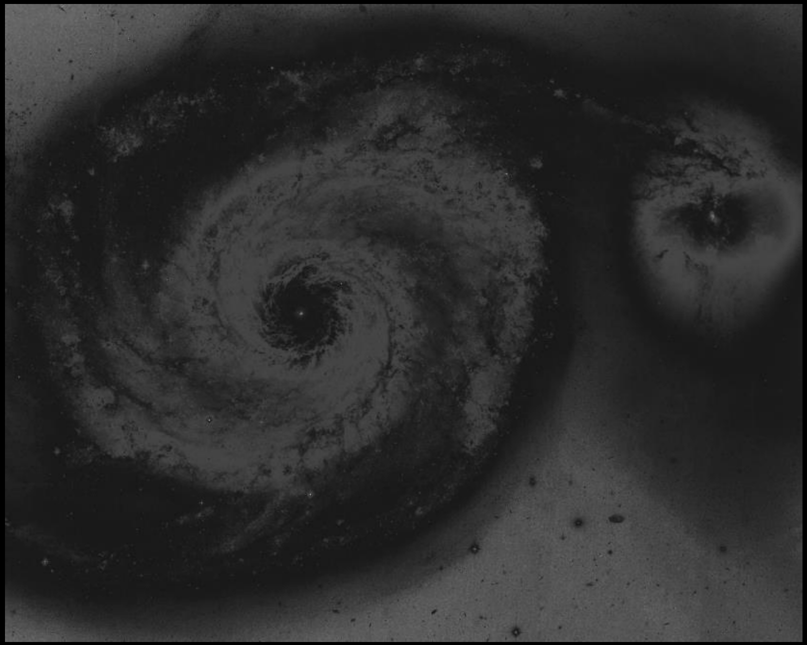


One can also construct linear CDF with randomly-generated perturbation. Qualitatively similar to the sinusoidal image, the intensity and contrast decreased with the applied transformation.

Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

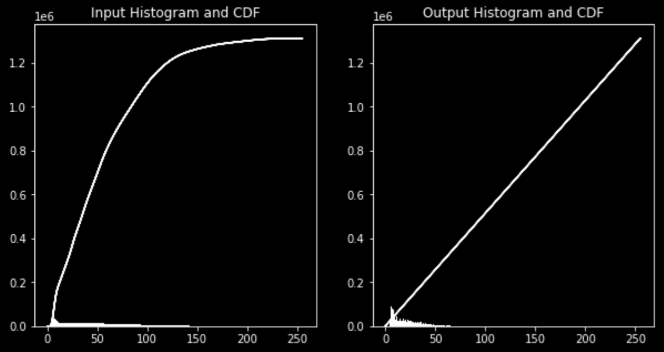
histogram-manipulation:~# Another way of image enhancement is through CDF backprojection. From a given histogram, one can calculate the PDF, and then the normalized CDF of an image. The calculated CDF can be transformed to another CDF shape via a lookup table.



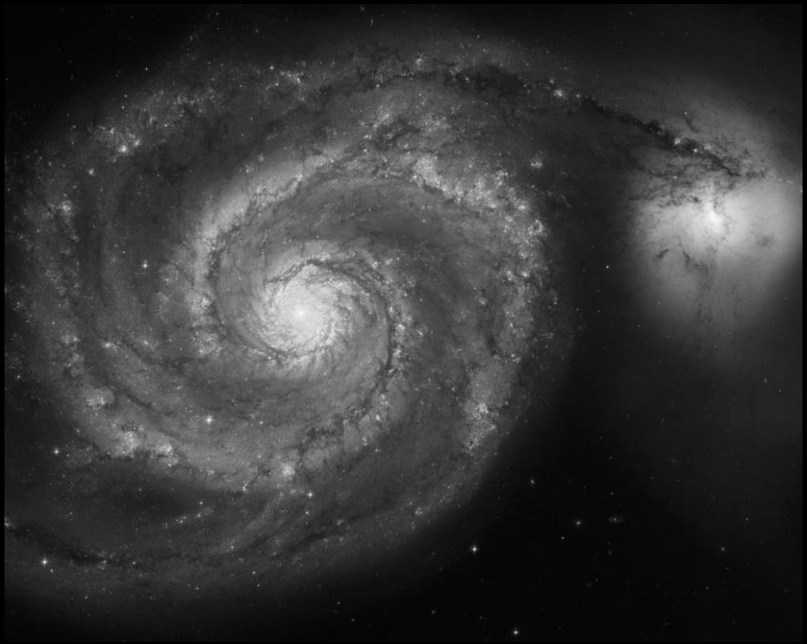


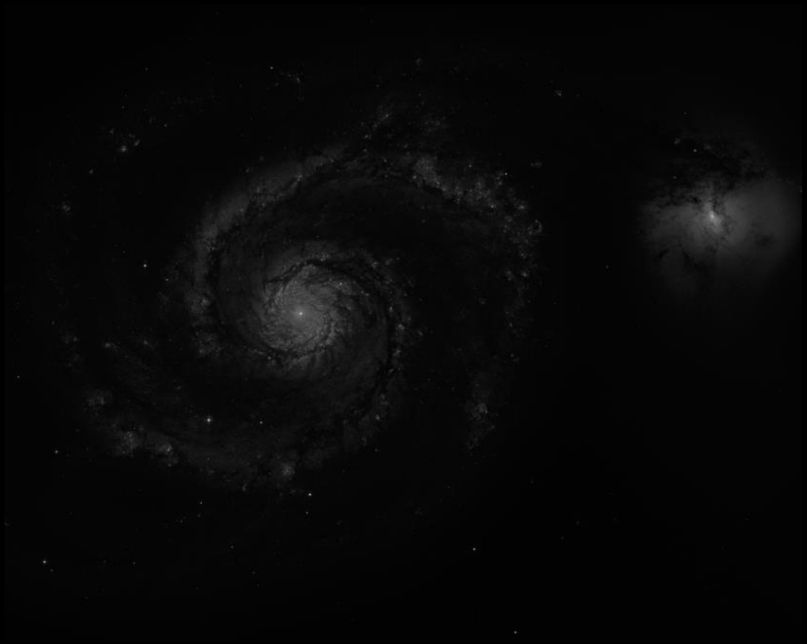
```
modify_cdf('dark_whirlpool.jpg', np.linspace(0, 1, 256))
```

A linear CDF transformation apparently inverted the grayscale image (black became white; white became black).

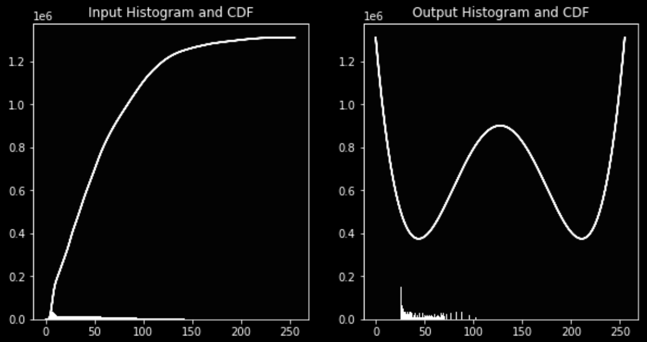


histogram-manipulation:~# Another way of image enhancement is through CDF backprojection. From a given histogram, one can calculate the PDF, and then the normalized CDF of an image. The calculated CDF can be transformed to another CDF shape via a lookup table.





```python
from scipy.special import legendre
x = np.linspace(-1, 1, 256)
p = legendre(4)(x)
desired_cdf = 0.5 * (1 + p)
modify_cdf('dark_whirlpool.jpg', desired_cdf)
```



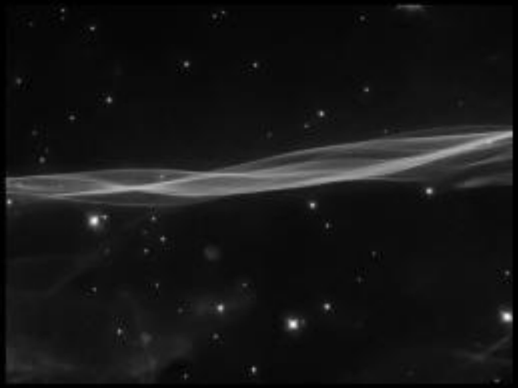
CDF transformation look up table must be properly constructed to fit the desired specific application although no one is stopping us in using a Legendre function CDF transformation for fun (or is it?)

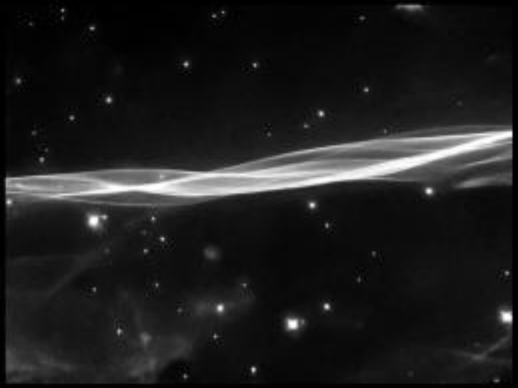
```
Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

histogram-manipulation:~# code it yourself!
```

```python
def modify_cdf(path, desired_cdf):
    """
    User defined function to modify the cdf of an image based on backpropagation
    from a specified lookup table of desired cdf

    Parameters:
    path (int): specify the path of an input image
    desired_cdf (numpy.ndarray): numpy array of a desired cdf

    Author:
    Lyle Kenneth Geraldez
    """
    # Load the input grayscale image
    img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)

    # Compute the input CDF
    hist, bins = np.histogram(img.flatten(), 256, [0,256])
    cdf = hist.cumsum()
    cdf_normalized = cdf / cdf.max()

    # Compute the mapping function using backpropagation
    mapping = np.interp(desired_cdf, cdf_normalized, bins[:-1])
    mapped_img = np.interp(img.flatten(), bins[:-1], mapping).astype(np.uint8)
    mapped_img = mapped_img.reshape(img.shape)

    # Plot the input and output histograms
    fig, ax = plt.subplots(1, 2, figsize=(10, 5))
    ax[0].hist(img.flatten(), 256, [0, 256], color='blue')
    ax[0].plot(cdf_normalized * img.size, color='red')
    ax[0].set_title('Input Histogram and CDF')

    ax[1].hist(mapped_img.flatten(), 256, [0, 256], color='blue')
    ax[1].plot(desired_cdf * img.size, color='red')
    ax[1].set_title('Output Histogram and CDF')

    # Show the images
    plt.show()

    # Save the output image
    cv2.imwrite('modified_cdf.jpg', mapped_img)
```

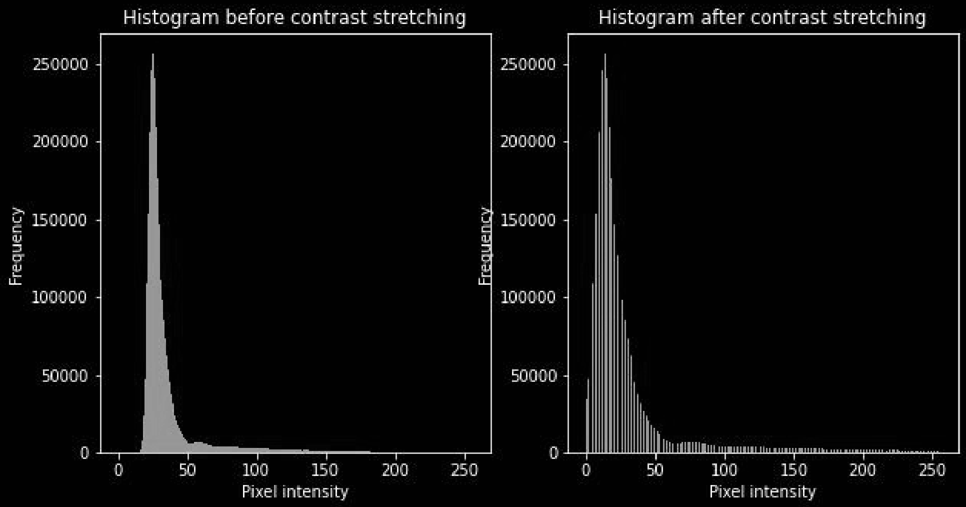
Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

contrast-enhancement:~# One way to increase the contrast of an image is by stretching out the grayscale pixel values such that the darker areas become darker and the lighter areas become lighter. In this new rescaling, each grayscale pixel value becomes transformed to a new one such that the pixel domain stretches from 0 to 255 thus increasing contrast





```
for i in range(1, 10, 1):
    contrast_gray('dark_galaxy.jpg',i,100-i)
```



One way to increase the effects of contrast stretching is to consider an upper and lower percentile limit. This limiting of domain and mapping out the clipped minimum and maximum pixel value to 0 and 255, respectively, increases the contrast effects more which can be noticed by the increase in stretching of histogram (below rendered animation) corresponding to an image with a higher contrast (above rendered animation)

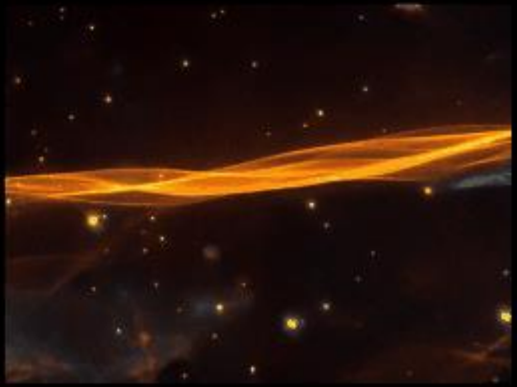
Defined function takes the lower and higher percentile limit as parameters which are varied to generate the animations.

```
Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

contrast-enhancement:~# code it yourself!
```

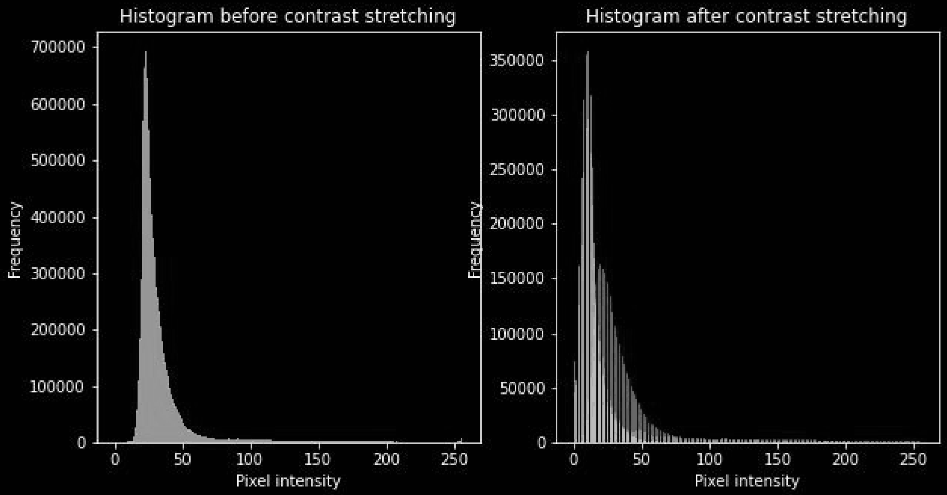
```python
def contrast_gray(path, lower, upper):
    """
    User defined function to apply contrast stretching to an image
    To increase contrast effects, apply percentile rescaling by clipping pixel values

    Parameters:

    path (string): set the path of the input image
    lower (int): set the lower percentile clipping lower pixel values
    upper (int): set the higher percentile clipping higher pixel values

    Author:
    Lyle Kenneth Geraldez
    """

    # Load image
    image = Image.open(path).convert('L')  # Convert to grayscale

    # Calculate minimum and maximum pixel values
    min_val, max_val = np.min(image), np.max(image)

    # Define lower and upper percentile values
    lower_percentile = lower
    upper_percentile = upper

    # Calculate new minimum and maximum pixel values
    new_min_val = np.percentile(image, lower_percentile)
    new_max_val = np.percentile(image, upper_percentile)

    # Apply contrast stretching
    output_image = (image - new_min_val) * (255 / (new_max_val - new_min_val))
    output_image = np.clip(output_image, 0, 255).astype(np.uint8)
```

```python
# Plot histogram before and after contrast stretching
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].hist(np.array(image).ravel(), bins=256, range=(0, 256))
axs[0].set_title('Histogram before contrast stretching')
axs[0].set_xlabel('Pixel intensity')
axs[0].set_ylabel('Frequency')
axs[1].hist(output_image.ravel(), bins=256, range=(0, 256))
axs[1].set_title('Histogram after contrast stretching')
axs[1].set_xlabel('Pixel intensity')
axs[1].set_ylabel('Frequency')
plt.savefig(f'contrast_gray_hist_{lower}.jpg', facecolor='black')

# Display input and output images
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(image, cmap = 'gray')
axs[0].set_title('Input image')
axs[1].imshow(output_image, cmap = 'gray')
axs[1].set_title('Output image')
plt.savefig(f'contrast_gray_{lower}.jpg', facecolor='black')
plt.show()
```

contrast-enhancement:~# The same principle can be applied to each color channel (RGB). Rescaling each of the color channel by mapping the lowest and highest color value to 0 and 255, respectively, similarly stretches the histogram of each color channel. This increase the contrast of a colored image.



```
for i in range(1, 10, 1):
    contrast_rgb('dark_galaxy.jpg', i,100-i)
```



Instead of the grayscale value, the code now applies the transformation to each of the color channel and appropriately apply similar transformation.

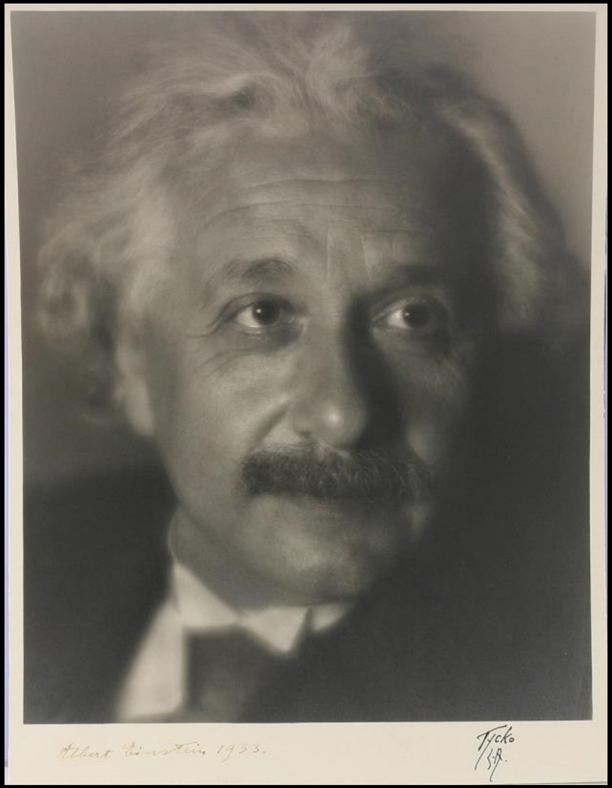
Observe how the image becomes crispier with more contrast as the clipping percentile is varied to increase the stretching effects. Now, each channel histograms get stretches independently.

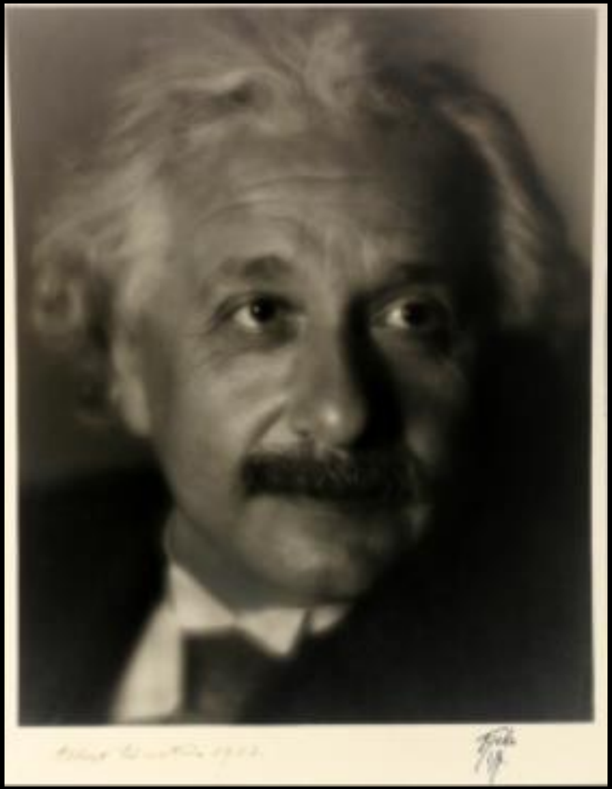
```
Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

contrast-enhancement:~# code it yourself!
```

```python
def contrast_rgb(path, lower, upper):
    """
    User defined function to apply contrast stretching to an image
    To increase contrast effects, apply percentile rescaling by clipping pixel values
    For RGB contrasting, we avoid grayscaling the input image
    Histogram is plotted for each color channel

    Parameters:

    path (string): set the path of the input image
    lower (int): set the lower percentile clipping lower pixel values
    upper (int): set the higher percentile clipping higher pixel values

    Author:
    Lyle Kenneth Geraldez
    """

    # Load image
    image = Image.open(path)

    # Calculate minimum and maximum pixel values
    min_val, max_val = np.min(image), np.max(image)

    # Define lower and upper percentile values
    lower_percentile = lower
    upper_percentile = upper

    # Calculate new minimum and maximum pixel values
    new_min_val = np.percentile(image, lower_percentile)
    new_max_val = np.percentile(image, upper_percentile)

    # Apply contrast stretching
    output_image = (image - new_min_val) * (255 / (new_max_val - new_min_val))
    output_image = np.clip(output_image, 0, 255).astype(np.uint8)
```

```python
# Plot histogram before and after contrast stretching
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].hist(np.array(image).ravel(), bins=256, range=(0, 256))
axs[0].set_title('Histogram before contrast stretching')
axs[0].set_xlabel('Pixel intensity')
axs[0].set_ylabel('Frequency')
for i in range(3):
    axs[1].hist(output_image[:,:,i].ravel(),
    bins=256, range=(0, 256), alpha=0.5, color=['red', 'green', 'blue'][i])
# axs[1].hist(output_image.ravel(), bins=256, range=(0, 256))
axs[1].set_title('Histogram after contrast stretching')
axs[1].set_xlabel('Pixel intensity')
axs[1].set_ylabel('Frequency')
# plt.savefig(f'contrast_rgb_hist_{lower}.png', facecolor='black')

# Display input and output images
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(image, cmap='gray')
axs[0].set_title('Input image')
axs[1].imshow(output_image, cmap='gray')
axs[1].set_title('Output image')
# plt.savefig(f'contrast_rgb_{lower}.png', facecolor='black')
plt.show()
```

fade-restoration:~# One way to restore an image is to apply the discussed contrast stretching to each of the color channel to a faded photograph.

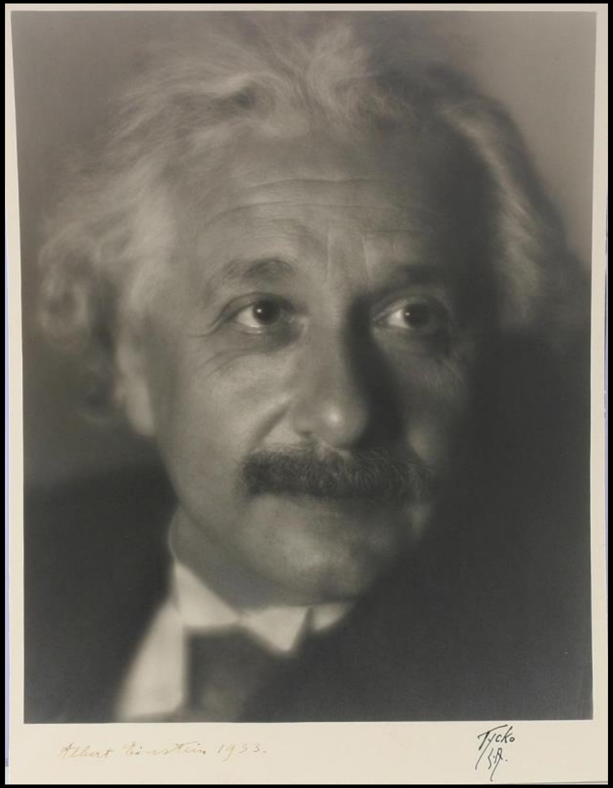


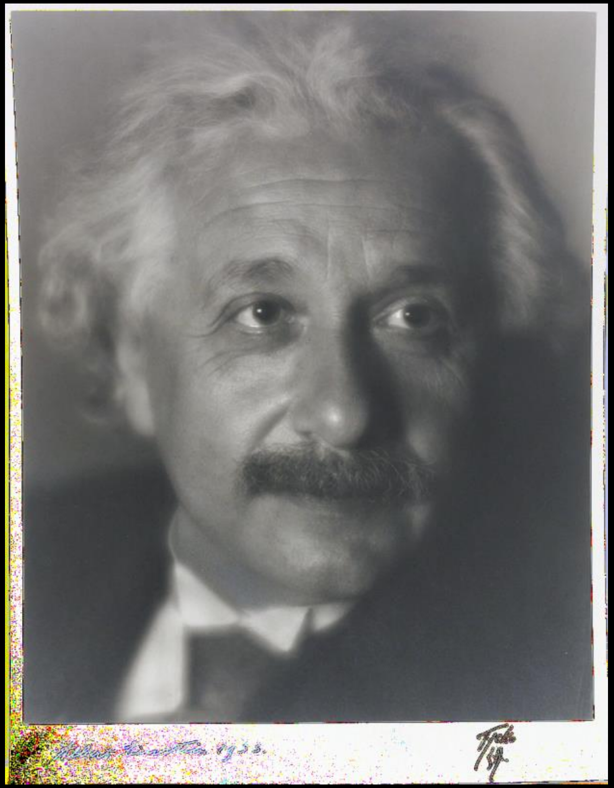


```
contrast_rgb('faded_einstein.jpg', 1,99)
```

Similar code was used from the previously-defined contrast-stretching algorithm with (1,99) percentile clipping. Observe that although several features were made prominent, it still produces a sepia overall theme.

fade-restoration:~# Another method of faded photograph restoration is the gray world algorithm. With the assumption of a neutral gray being the average color, one can rescale each color channel by the corresponding average of each channel across the image. This produces a fade-reversion by re-balancing the image to reduce the appearance of fade-related color perturbation.



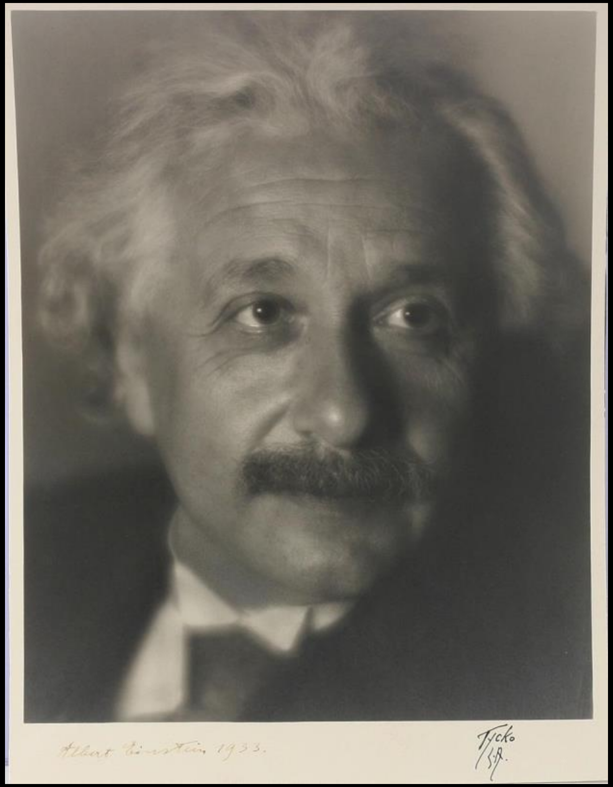


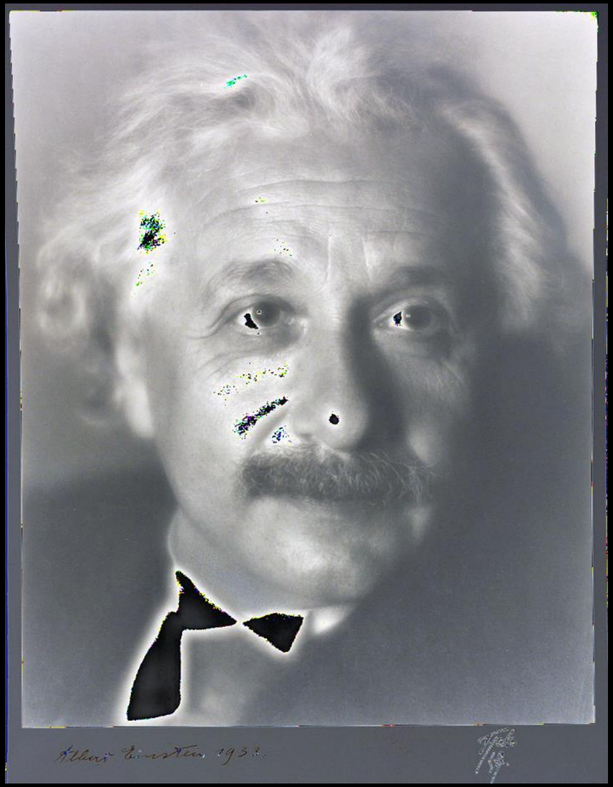
```
gray_world('faded_einstein.jpg')
```

Gray world algorithm function calculates the average of each color channel and rescales each color channel by the respective averages. To scale them back to 0 to 255 domain, a rescaling factor (128) is applied. Observe that the sepia theme disappeared revealing a grayscale version of the image (which, I presume, may have originally been a grayscale image all along)

Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

fade-restoration:~# Instead of rescaling by the average value of the color channel, we can rescale them by the average value of the color only from a known white patch. Here, we choose a small square along Einstein's eyes as white patch.





```
white_patch('faded_einstein.jpg', (480,495,367,383))
```

The function applies the white-patch rescaling which also takes the coordinate sides of the white patch rectangle. Corresponding code slide shows snippet of coordinate extraction from an image. Needless to say, the eye white patch choice failed.

```
Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

fade-restoration:~# code it yourself!
```

```python
def contrast_rgb(path, lower, upper):
    """
    User defined function to apply contrast stretching to an image
    To increase contrast effects, apply percentile rescaling by clipping pixel values
    For RGB contrasting, we avoid grayscaling the input image
    Histogram is plotted for each color channel

    Parameters:

    path (string): set the path of the input image
    lower (int): set the lower percentile clipping lower pixel values
    upper (int): set the higher percentile clipping higher pixel values

    Author:
    Lyle Kenneth Geraldez
    """

    # Load image
    image = Image.open(path)

    # Calculate minimum and maximum pixel values
    min_val, max_val = np.min(image), np.max(image)

    # Define lower and upper percentile values
    lower_percentile = lower
    upper_percentile = upper

    # Calculate new minimum and maximum pixel values
    new_min_val = np.percentile(image, lower_percentile)
    new_max_val = np.percentile(image, upper_percentile)

    # Apply contrast stretching
    output_image = (image - new_min_val) * (255 / (new_max_val - new_min_val))
    output_image = np.clip(output_image, 0, 255).astype(np.uint8)
```

```python
# Plot histogram before and after contrast stretching
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].hist(np.array(image).ravel(), bins=256, range=(0, 256))
axs[0].set_title('Histogram before contrast stretching')
axs[0].set_xlabel('Pixel intensity')
axs[0].set_ylabel('Frequency')
for i in range(3):
    axs[1].hist(output_image[:,:,i].ravel(),
    bins=256, range=(0, 256), alpha=0.5, color=['red', 'green', 'blue'][i])
# axs[1].hist(output_image.ravel(), bins=256, range=(0, 256))
axs[1].set_title('Histogram after contrast stretching')
axs[1].set_xlabel('Pixel intensity')
axs[1].set_ylabel('Frequency')
# plt.savefig(f'contrast_rgb_hist_{lower}.png', facecolor='black')

# Display input and output images
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].imshow(image, cmap='gray')
axs[0].set_title('Input image')
axs[1].imshow(output_image, cmap='gray')
axs[1].set_title('Output image')
# plt.savefig(f'contrast_rgb_{lower}.png', facecolor='black')
plt.show()
```

```
Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

fade-restoration:~# code it yourself!
```

```python
def gray_world(img_path):
    """
    User defined function to apply the gray world algorithm for faded photograph recovery

    Parameters:
    img_path (string): set input image path

    Author:
    Lyle Kenneth Geraldez

    """

    # Load the image
    img = Image.open(img_path)
    # Convert the image to a numpy array
    img_arr = np.array(img)

    # Get the average value of the red, green and blue channel of the image
    Rave = np.mean(img_arr[:,:,0])
    Gave = np.mean(img_arr[:,:,1])
    Bave = np.mean(img_arr[:,:,2])

    # To get the white balanced image, divide each RGB channel by the respective averages
    img_arr[:,:,0] = img_arr[:,:,0] / Rave * 128
    img_arr[:,:,1] = img_arr[:,:,1] / Gave * 128
    img_arr[:,:,2] = img_arr[:,:,2] / Bave * 128

    # Convert the numpy array back to an image
    balanced_img = Image.fromarray(np.uint8(img_arr))
    return balanced_img
```

```
Geraldez-LK-2019-11336 157-WFY-FX-1 ~ (digital-eyes)

fade-restoration:~# code it yourself!
```

```python
def white_patch(img_path, white_patch_region):
    """
    User defined function to apply the white patch algorithm for faded photograph recovery

    Parameters:
    img_path (string): set input image path
    white_patch_region (tuple): set four numbers in the following format
        (top, bottom, left, right) coordinates of the white patch region


    Author:
    Lyle Kenneth Geraldez
    """

    # Load the image
    img = Image.open(img_path)

    # Convert the image to a numpy array
    img_arr = np.array(img)

    # Get a region from the image which is known to be white
    white_patch_arr = img_arr[white_patch_region[0]:white_patch_region[1],
                              white_patch_region[2]:white_patch_region[3]]

    # Average the RGB of the white pixels to get Rw, Gw, Bw, separately
    Rw = np.mean(white_patch_arr[:,:,0])
    Gw = np.mean(white_patch_arr[:,:,1])
    Bw = np.mean(white_patch_arr[:,:,2])

    # Divide each channel of the whole original image with the respective white averages
    img_arr[:,:,0] = img_arr[:,:,0] / Rw * 255
    img_arr[:,:,1] = img_arr[:,:,1] / Gw * 255
    img_arr[:,:,2] = img_arr[:,:,2] / Bw * 255

    # Convert the numpy array back to an image
    balanced_img = Image.fromarray(np.uint8(img_arr))
    return balanced_img
```

```python
import cv2

# Define the mouse event handler function
def mouse_event_handler(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:  # if left mouse button is clicked
        print(f'Coordinates: ({x}, {y})')

# Load the image using OpenCV
img = cv2.imread('faded_einstein.jpg')

# Create a window to display the image
cv2.namedWindow('image')

# Set the mouse event handler function to the window
cv2.setMouseCallback('image', mouse_event_handler)

# Display the image in the window
cv2.imshow('image', img)
cv2.waitKey(0)  # Wait for a key to be pressed to close the window
cv2.destroyAllWindows()  # Close all windows
```

technical-correctness:~# The report objectives were properly achieved (and then some) with concrete results in the form of produced images and rendered animations. For some methodology, the images were seen to be qualitatively enhanced by the appropriate transformation methods outlines on the manual. Moreover, methods where these enhancements fail lies not in the code implementation but that of appropriateness to a specific situation (such as in the white-patch algorithm or even the hypothesized sepia-originated faded photograph of Einstein's image). Nevertheless, the required images were all rendered, compiled, and documented such that the validity and correctness can be verified. **35/35**

quality-of-presentation:~# The temporal dimension is a powerful coordinate to make use when presentation information especially in the field of imaging. In contrast with presenting a series of disconnected snapshots, rendering them into a moving animated movie provides an intuitive essence which can be directly extract to supplement what a particular code, scheme, algorithm, or method is being tackle. However, I may be fooling myself if I don't tell you that I did this mainly because it was super cool! At the heart of my trademarked black-and-neon themed presentation, images were well integrated seamlessly. The code snippet (mainly, function calling) was attached to each simulation. The called function defined are fully shown on a separate slide for transparency and ease of access although one can find a fully-explicit code running on the respective directories at the GitHub repository. **35/35**

self-reflection:~# The report itself was constructed in a more narrative manner to include some aspect of self-reflection although the journey itself throughout the entire activity are being properly summarized in this page. There are some steps wherein some of the results were that of pure exploration (like applying a Legendre CDF to an image). The references for the used image was also properly cited. **28/30**

initiative:~# The simulation results and some extended discussions related to other applied fields such as holography and digital wave propagation were included on the report. The written codes were all functionized for ease of calling and replication of results. These were properly commented and documented by an appropriate descriptive docstring. Together with the rendering of the images, this constitute different aspects of the report that went over the bare minimum objectives. **10/10**

Total: 108/100

references:~#

+ dark_galaxy.jpg: "Featured Image: Star Clusters in M51" (2016). [Image]. Retrieved from https://aasnova.org/2016/06/20/featured-image-star-clusters-in-m51/

+ dark_whirlpool.jpg: "NGC 5195" (2009). [Image]. Retrieved from https://en.wikipedia.org/wiki/NGC_5195

+ faded_einstein.jpg: "Einstein, Albert" (Unknown). [Image]. Retrieved from https://einstein.manhattanrarebooks.com/pages/books/20/albert-einstein/photograph-signed