Create Any Image Using Only Sine Functions - 2D Fourier Transform in Python

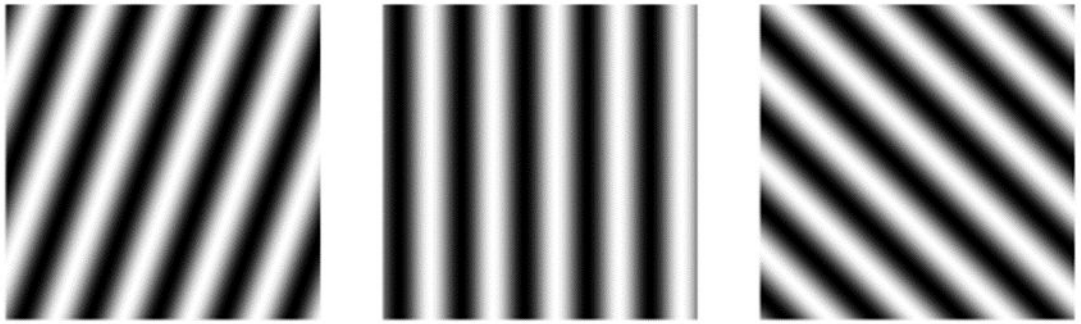
J022 RISHI GANDHI
BTECH DATA SCIENCE

What Are Sinusoidal Gratings?

A sinusoidal grating is a two-dimensional representation in which the amplitude varies sinusoidally along a certain direction. All the examples below are sinusoidal gratings having a different orientation:

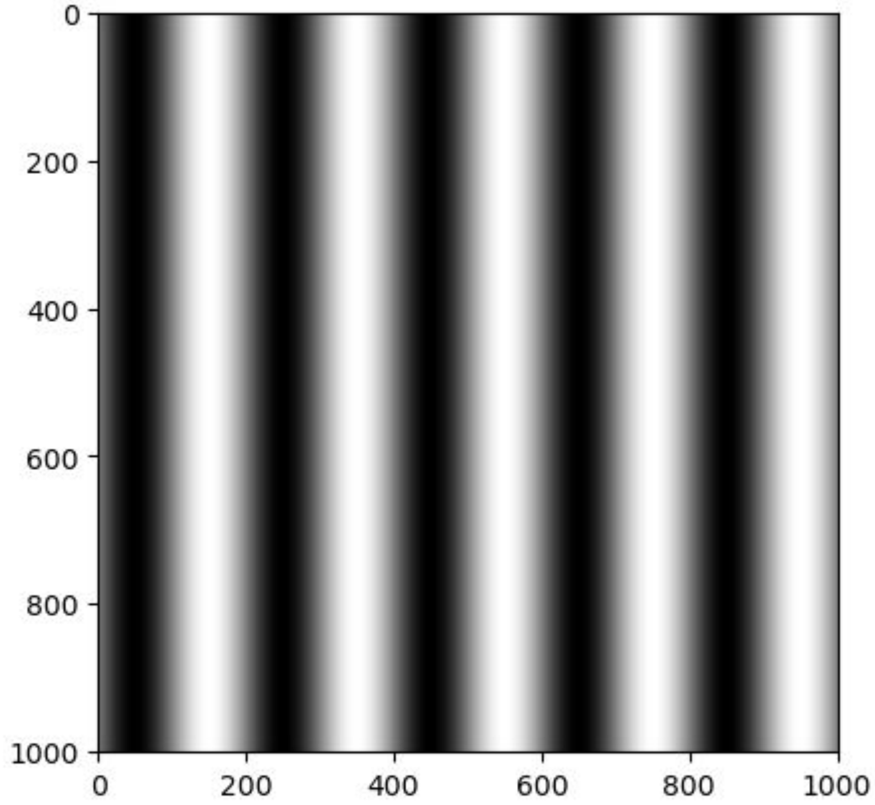
The parameters that describe a sinusoidal grating are:

1. wavelength or frequency
2. amplitude
3. orientation
4. Phase



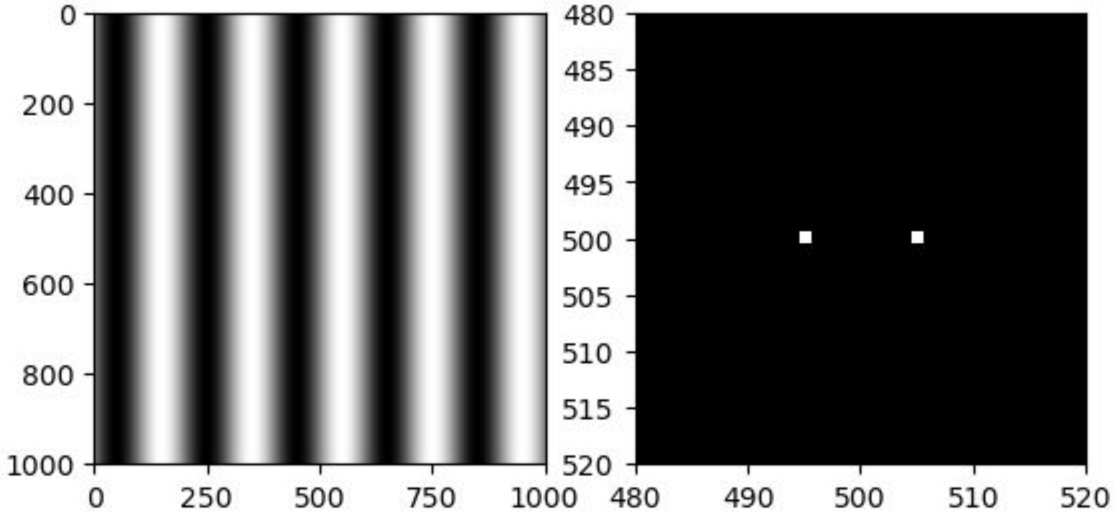
Creating Sinusoidal Gratings using NumPy in Python

```
# gratings.py

import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-500, 501, 1)

X, Y = np.meshgrid(x, x)

wavelength = 200
grating = np.sin(2 * np.pi * X / wavelength)

plt.set_cmap("gray")
plt.imshow(grating)
plt.show()
```



The Fourier Transform

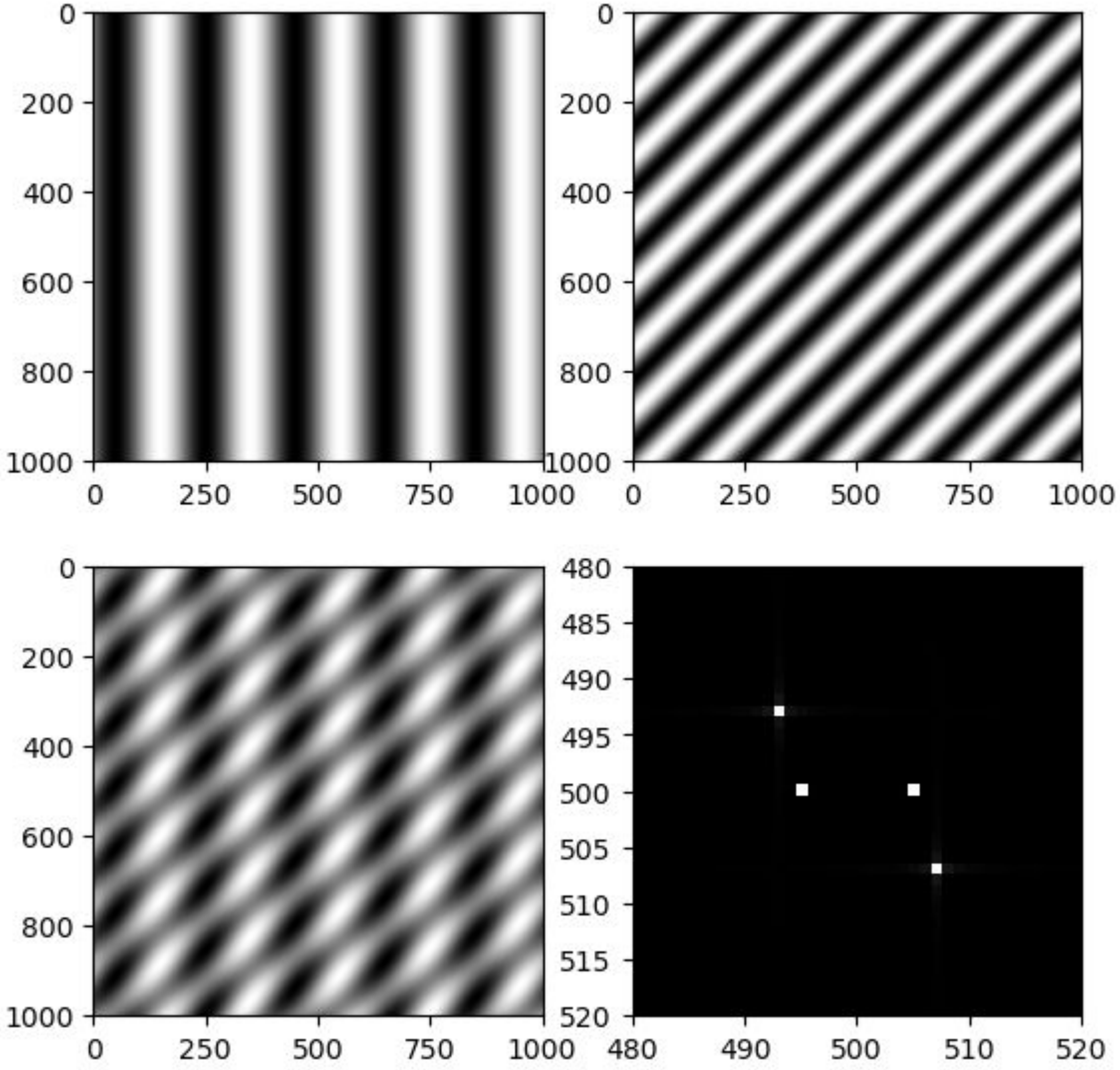
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-500, 501, 1)

X, Y = np.meshgrid(x, x)

wavelength = 200
angle = 0
grating = np.sin(
    2*np.pi*(X*np.cos(angle) + Y*np.sin(angle)) / wavelength
)

plt.set_cmap("gray")

plt.subplot(121)
plt.imshow(grating)

# Calculate Fourier transform of grating
ft = np.fft.ifftshift(grating)
ft = np.fft.fft2(ft)
ft = np.fft.fftshift(ft)

plt.subplot(122)
plt.imshow(abs(ft))
plt.xlim([480, 520])
plt.ylim([520, 480])  # Note, order is reversed for y
plt.show()
```



Take the two sinusoidal gratings you created and work out their Fourier transform using Python’s NumPy.

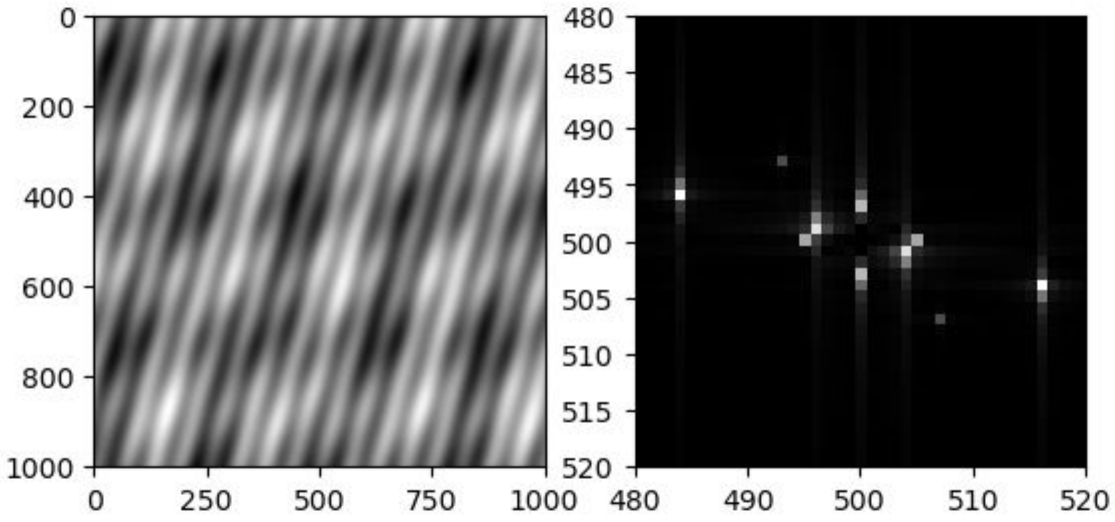
Adding More Than One Grating

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-500, 501, 1)

X, Y = np.meshgrid(x, x)

wavelength_1 = 200
angle_1 = 0
grating_1 = np.sin(
    2*np.pi*(X*np.cos(angle_1) + Y*np.sin(angle_1)) / wavelength_1
)
wavelength_2 = 100
angle_2 = np.pi/4
grating_2 = np.sin(
    2*np.pi*(X*np.cos(angle_2) + Y*np.sin(angle_2)) / wavelength_2
)

plt.set_cmap("gray")
plt.subplot(121)
plt.imshow(grating_1)
plt.subplot(122)
plt.imshow(grating_2)
plt.show()

gratings = grating_1 + grating_2

# Calculate Fourier transform of the sum of the two gratings
ft = np.fft.ifftshift(gratings)
ft = np.fft.fft2(ft)
ft = np.fft.fftshift(ft)

plt.figure()
plt.subplot(121)
plt.imshow(gratings)

plt.subplot(122)
plt.imshow(abs(ft))
plt.xlim([480, 520])
plt.ylim([520, 480])  # Note, order is reversed for y
plt.show()
```



Adding more sinusoidal gratings

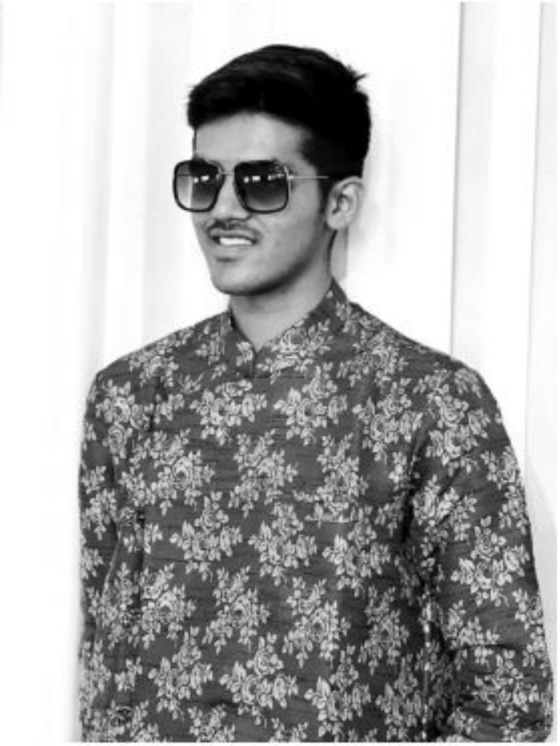
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-500, 501, 1)

X, Y = np.meshgrid(x, x)

amplitudes = 0.5, 0.25, 1, 0.75, 1
wavelengths = 200, 100, 250, 300, 60
angles = 0, np.pi / 4, np.pi / 9, np.pi / 2, np.pi / 12

gratings = np.zeros(X.shape)
for amp, w_len, angle in zip(amplitudes, wavelengths, angles):
    gratings += amp * np.sin(
        2*np.pi*(X*np.cos(angle) + Y*np.sin(angle)) / w_len
    )

# Calculate Fourier transform of the sum of the gratings
ft = np.fft.ifftshift(gratings)
ft = np.fft.fft2(ft)
ft = np.fft.fftshift(ft)

plt.set_cmap("gray")
plt.subplot(121)
plt.imshow(gratings)

plt.subplot(122)
plt.imshow(abs(ft))
plt.xlim([480, 520])
plt.ylim([520, 480])  # Note, order is reversed for y
plt.show()
```



The image on the left shows all five gratings superimposed. The Fourier transform on the right shows the individual terms as pairs of dots. The amplitude of the dots represents the amplitudes of the gratings, too.

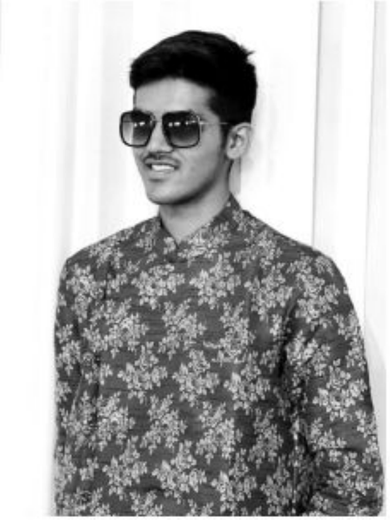
Reading The Image and Converting To Grayscale

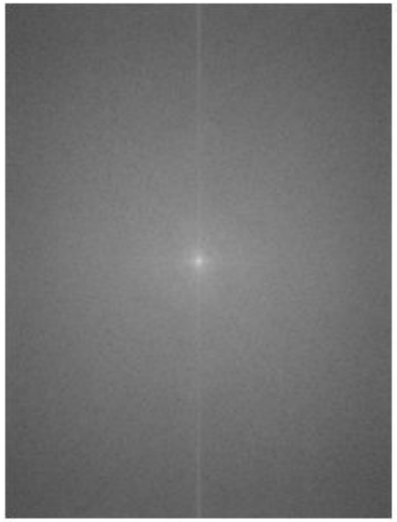
```
# fourier_synthesis.py

import matplotlib.pyplot as plt

image_filename = "Rishi.jpg"

# Read and process image
image = plt.imread(image_filename)
image = image[:, :, :3].mean(axis=2)  # Convert to grayscale
print(image.shape)

plt.set_cmap("gray")

plt.imshow(image)
plt.axis("off")
plt.show()
```



Calculating the 2D Fourier Transform of The Image

```python
# fourier_synthesis.py

import numpy as np
import matplotlib.pyplot as plt

image_filename = "Rishi.jpg"

def calculate_2dft(input):
    ft = np.fft.ifftshift(input)
    ft = np.fft.fft2(ft)
    return np.fft.fftshift(ft)

# Read and process image
image = plt.imread(image_filename)
image = image[:, :, :3].mean(axis=2)  # Convert to grayscale

plt.set_cmap("gray")

ft = calculate_2dft(image)

plt.subplot(121)
plt.imshow(image)
plt.axis("off")
plt.subplot(122)
plt.imshow(np.log(abs(ft)))
plt.axis("off")
plt.show()
```





Now there are lots of dots that have non-zero values in the Fourier transform. Instead of five pairs of dots representing five sinusoidal gratings, you now have thousands of pairs of dots. This means that there are thousands of sinusoidal gratings present in the image. Each pair of dots represents a sinusoidal grating with a specific frequency, amplitude, orientation, and phase.

The Inverse Fourier Transform

- Finding All The Pairs of Points in The 2D Fourier Transform
- Sorting The Coordinates in Order of Distance From The Centre
- Finding The Second Symmetrical Point in Each Pair
- Using the 2D Fourier Transform in Python to Reconstruct The Image

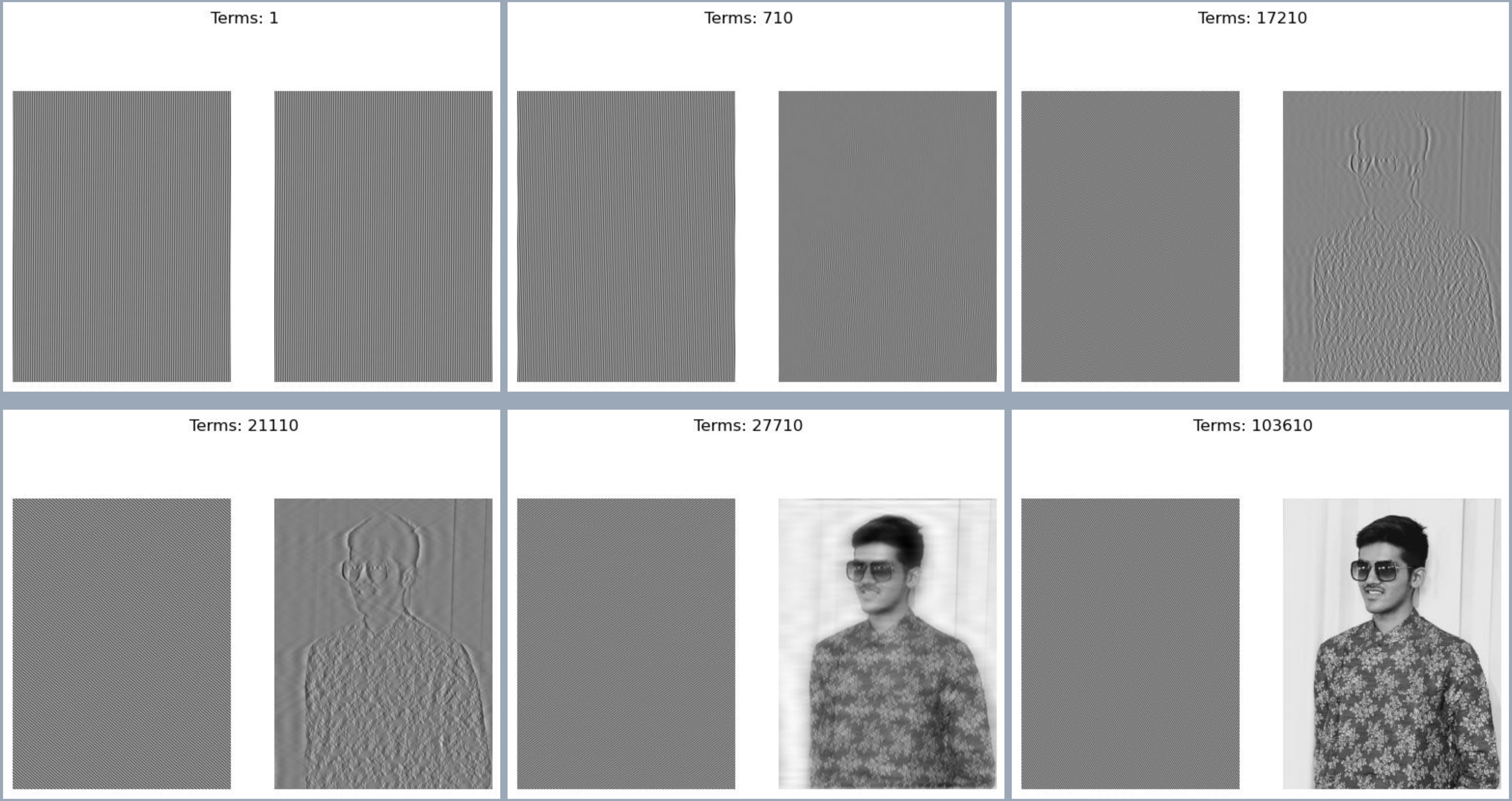
  - The main algorithm, consisting of the four steps listed above, works its way through the whole Fourier transform, retrieving sinusoidal gratings and reconstructing the final image. The comments in the code signpost the link between these steps and the corresponding sections in the code.

```python
# fourier_synthesis.py
import time
import numpy as np
import matplotlib.pyplot as plt

image_filename = "Rishi.jpg"
start = time.time()

def calculate_2dft(input):
    ft = np.fft.ifftshift(input)
    ft = np.fft.fft2(ft)
    return np.fft.fftshift(ft)

def calculate_2dift(input):
    ift = np.fft.ifftshift(input)
    ift = np.fft.ifft2(ift)
    ift = np.fft.fftshift(ift)
    return ift.real

def calculate_distance_from_centre(coords, centre):
    # Distance from centre is √(x^2 + y^2)
    return np.sqrt(
        (coords[0] - centre) ** 2 + (coords[1] - centre) ** 2
    )

def find_symmetric_coordinates(coords, centre):
    return (centre + (centre - coords[0]),
            centre + (centre - coords[1]))

def display_plots(individual_grating, reconstruction, idx):
    plt.subplot(121)
    plt.imshow(individual_grating)
    plt.axis("off")
    plt.subplot(122)
    plt.imshow(reconstruction)
    plt.axis("off")
    plt.suptitle(f"Terms: {idx}")
    plt.pause(0.01)

# Read and process image
image = plt.imread(image_filename)
image = image[:, :, :3].mean(axis=2)  # Convert to grayscale
```

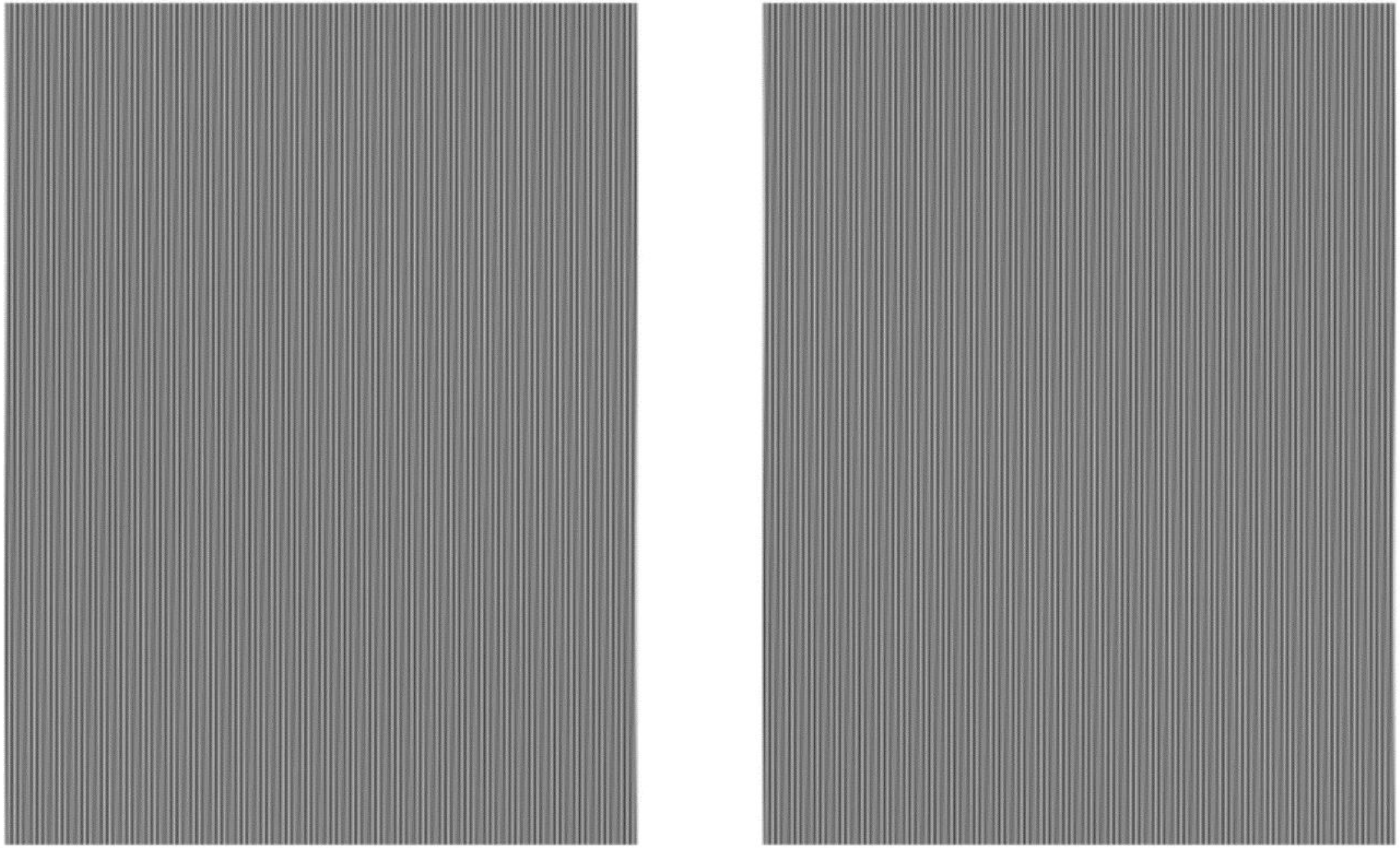
```python
# Array dimensions (array is square) and centre pixel
array_size = len(image)
centre = int((array_size - 1) / 2)

# Get all coordinate pairs in the left half of the array,
# including the column at the centre of the array (which
# includes the centre pixel)
coords_left_half = (
    (x, y) for x in range(array_size) for y in range(centre+1)
)

# Sort points based on distance from centre
coords_left_half = sorted(
    coords_left_half,
    key=lambda x: calculate_distance_from_centre(x, centre)
)

plt.set_cmap("gray")

ft = calculate_2dft(image)

# Show grayscale image and its Fourier transform
plt.subplot(121)
plt.imshow(image)
plt.axis("off")
plt.subplot(122)
plt.imshow(np.log(abs(ft)))
plt.axis("off")
plt.pause(2)

# Reconstruct image
fig = plt.figure()
# Step 1
# Set up empty arrays for final image and
# individual gratings
rec_image = np.zeros(image.shape)
individual_grating = np.zeros(
    image.shape, dtype="complex"
)
idx = 0

# All steps are displayed until display_all_until value
display_all_until = 10
# After this, skip which steps to display using the
# display_step value
display_step = 100
# Work out index of next step to display
next_display = display_all_until + display_step
```


```python
# All steps are displayed until display_all_until value
display_all_until = 10
# After this, skip which steps to display using the
# display_step value
display_step = 100
# Work out index of next step to display
next_display = display_all_until + display_step
```


```python
# Step 2
for coords in coords_left_half:
    # Central column: only include if points in top half of
    # the central column
    if not (coords[1] == centre and coords[0] > centre):
        idx += 1
        symm_coords = find_symmetric_coordinates(
            coords, centre
        )
        # Step 3
        # Copy values from Fourier transform into
        # individual_grating for the pair of points in
        # current iteration
        individual_grating[coords] = ft[coords]
        individual_grating[symm_coords] = ft[symm_coords]

        # Step 4
        # Calculate inverse Fourier transform to give the
        # reconstructed grating. Add this reconstructed
        # grating to the reconstructed image
        rec_grating = calculate_2dift(individual_grating)
        rec_image += rec_grating

        # Clear individual_grating array, ready for
        # next iteration
        individual_grating[coords] = 0
        individual_grating[symm_coords] = 0

        # Don't display every step
        if idx < display_all_until or idx == next_display:
            if idx > display_all_until:
                next_display += display_step
                # Accelerate animation the further the
                # iteration runs by increasing
                # display_step
                display_step += 100
            display_plots(rec_grating, rec_image, idx)

plt.show()
print("Time taken :",time.time()-start)
```


Terms: 1
Terms: 710
Terms: 17210
Terms: 21110
Terms: 27710
Terms: 103610

Terms: 1



Thank You