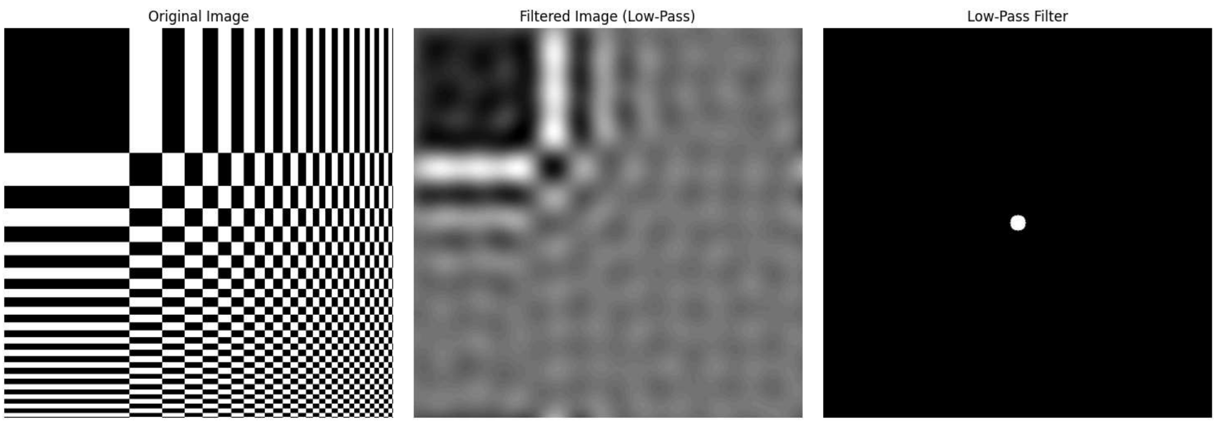
In [34]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

Q1 (a) ILPF

In [60]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import cv2

def ILPF(rows,cols, D_0):

    crow, ccol = rows // 2, cols // 2

    # Create a grid of distances from the center
    x, y = np.ogrid[:rows, :cols]
    distance = np.sqrt((x - crow)**2 + (y - ccol)**2)

    # Create the circular filter
    H_ILPF = np.where(distance <= D_0, 1, 0)

    return  H_ILPF



# Load and process the image
img = mpimg.imread('dynamicCheckerboard.png')

D_0 = 10
u, v = img.shape
H_ILPF = ILPF(u, v, D_0)

# Perform the 2D Fourier Transform
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)

fshift_filtered = fshift * H_ILPF

# Apply the inverse shift and inverse Fourier Transform to get the filtered image b
f_ishift = np.fft.ifftshift(fshift_filtered)
img_back = np.fft.ifft2(f_ishift)
img_back = np.real(img_back)

# Plot the original image, the filtered image, and the filter
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

# Plot the original image
axs[0].imshow(img, cmap='gray')
axs[0].set_title('Original Image')
axs[0].axis('off')

# Plot the filtered image
```

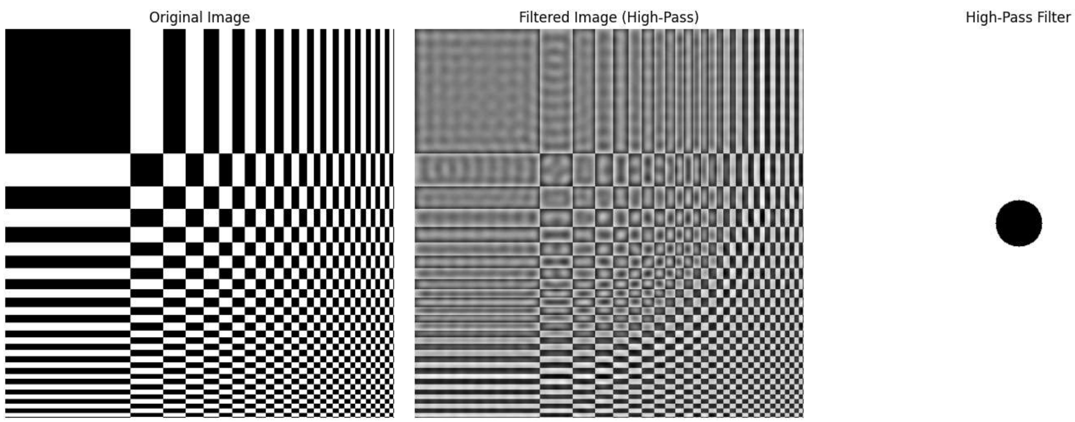
```
axs[1].imshow(img_back, cmap='gray')
axs[1].set_title('Filtered Image (Low-Pass)')
axs[1].axis('off')

# Plot the low-pass filter
axs[2].imshow(H_ILPF, cmap='gray')
axs[2].set_title('Low-Pass Filter')
axs[2].axis('off')

plt.tight_layout()
plt.show()
```



Q1 (b) IHPF

In [61]:
```
def IHPF(rows, cols, D_0):
    H_IHPF = 1 - ILPF(rows,cols, D_0)
    return H_IHPF

# Load and process the image
img = mpimg.imread('dynamicCheckerboard.png')
D_0 = 30
u, v = img.shape
H_IHPF = IHPF(u,v,D_0 )

# Perform the 2D Fourier Transform
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)

# Apply the High-Pass Filter
fshift_filtered = fshift * H_IHPF

# Apply the inverse shift and inverse Fourier Transform to get the filtered image b
f_ishift = np.fft.ifftshift(fshift_filtered)
img_back = np.fft.ifft2(f_ishift)
img_back = np.real(img_back)


# Plot the original image, the filtered image, and the filter
plt.figure(figsize=(15, 5))

# Original Image
```

```
plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Filtered Image
plt.subplot(1, 3, 2)
plt.imshow(img_back, cmap='gray')
plt.title('Filtered Image (High-Pass)')
plt.axis('off')

# High-Pass Filter
plt.subplot(1, 3, 3)
plt.imshow(H_IHPF, cmap='gray')
plt.title('High-Pass Filter')
plt.axis('off')

plt.tight_layout()
plt.show()
```

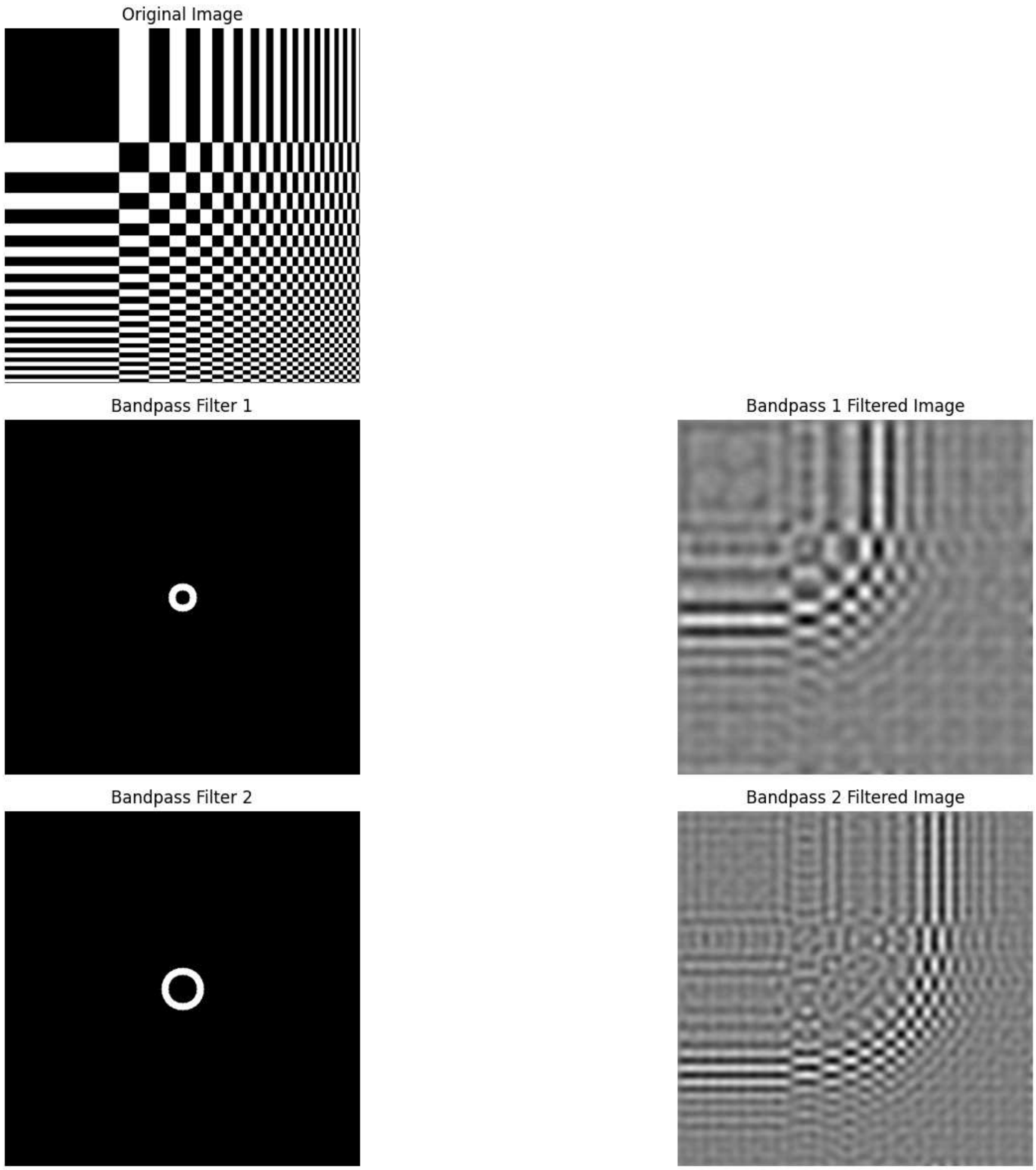


Q1 (c) IBPF

In [62]:
```
def IBPF(rows, cols, D_l, D_h):
    H_IBPF = ILPF(rows,cols, D_h)*IHPF(rows, cols, D_l)
    return H_IBPF

def filtering_operation(img, H_IBPF):

    # Perform the 2D Fourier Transform
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)

    # Apply the band-pass filter in the frequency domain
    fshift_filtered = fshift * H_IBPF

    # Apply the inverse shift and inverse Fourier Transform to get the filtered ima
    f_ishift = np.fft.ifftshift(fshift_filtered)
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.real(img_back)

    return img_back
```

```python
# Load and process the image
img = mpimg.imread('dynamicCheckerboard.png')
u, v = img.shape

# Apply Bandpass Filter 1
h_bandpass_1 = 20  # Outer radius (low-pass effect)
l_bandpass_1 = 10  # Inner radius (high-pass effect)

H_IBPF_1 = IBPF(u, v, l_bandpass_1, h_bandpass_1)
filtered_img_1 = filtering_operation(img, H_IBPF_1)

# Apply Bandpass Filter 2
h_bandpass_2 = 30  # Outer radius (low-pass effect)
l_bandpass_2 = 20  # Inner radius (high-pass effect)
H_IBPF_2 = IBPF(u,v, l_bandpass_2, h_bandpass_2)
filtered_img_2 = filtering_operation(img, H_IBPF_2)

# Convert images to uint8 (normalized to 255) for display
img_uint8 = cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
filtered_img_1_uint8 = cv2.normalize(filtered_img_1, None, 0, 255, cv2.NORM_MINMAX)
filtered_img_2_uint8 = cv2.normalize(filtered_img_2, None, 0, 255, cv2.NORM_MINMAX)

# Plot the original image, filters, and the corresponding filtered images
plt.figure(figsize=(15, 12))

# Plot the original image
plt.subplot(3, 2, 1)
plt.imshow(img_uint8, cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Plot Bandpass Filter 1
plt.subplot(3, 2, 3)
plt.imshow(H_IBPF_1, cmap='gray')
plt.title('Bandpass Filter 1')
plt.axis('off')

# Plot the filtered image with Bandpass Filter 1
plt.subplot(3, 2, 4)
plt.imshow(filtered_img_1_uint8, cmap='gray')
plt.title('Bandpass 1 Filtered Image')
plt.axis('off')

# Plot Bandpass Filter 2
plt.subplot(3, 2, 5)
plt.imshow(H_IBPF_2, cmap='gray')
plt.title('Bandpass Filter 2')
plt.axis('off')

# Plot the filtered image with Bandpass Filter 2
plt.subplot(3, 2, 6)
plt.imshow(filtered_img_2_uint8, cmap='gray')
plt.title('Bandpass 2 Filtered Image')
plt.axis('off')
```

```
plt.tight_layout()
plt.show()
```



Conclusion Q1

(a) ILPF (Low Pass Filter): The ILPF at a cut-off frequency of 10 effectively smoothed the image by removing high-frequency components. This led to a significant reduction in sharp edges and fine details, highlighting the low-frequency information in the image, such as large patterns and shapes.

(b) IHPF (High Pass Filter): The high pass filter at a cut-off frequency of 30 removed most of the low-frequency content, resulting in an image dominated by high-frequency components. This made the edges and fine details prominent, while the smoother regions of the image were significantly suppressed.

(c) IBPF (Band Pass Filters): The band pass filters, with lower and higher cut-offs (10, 20) and (20, 30), isolated specific frequency ranges. As a result:

```
The IBPF (10, 20) emphasized medium frequency components, retaining
some detail while suppressing both very fine and very broad
patterns.
The IBPF (20, 30) further refined this effect by allowing a
narrower range of frequencies, thus filtering out more details but
focusing on intermediate structures within the image.
```

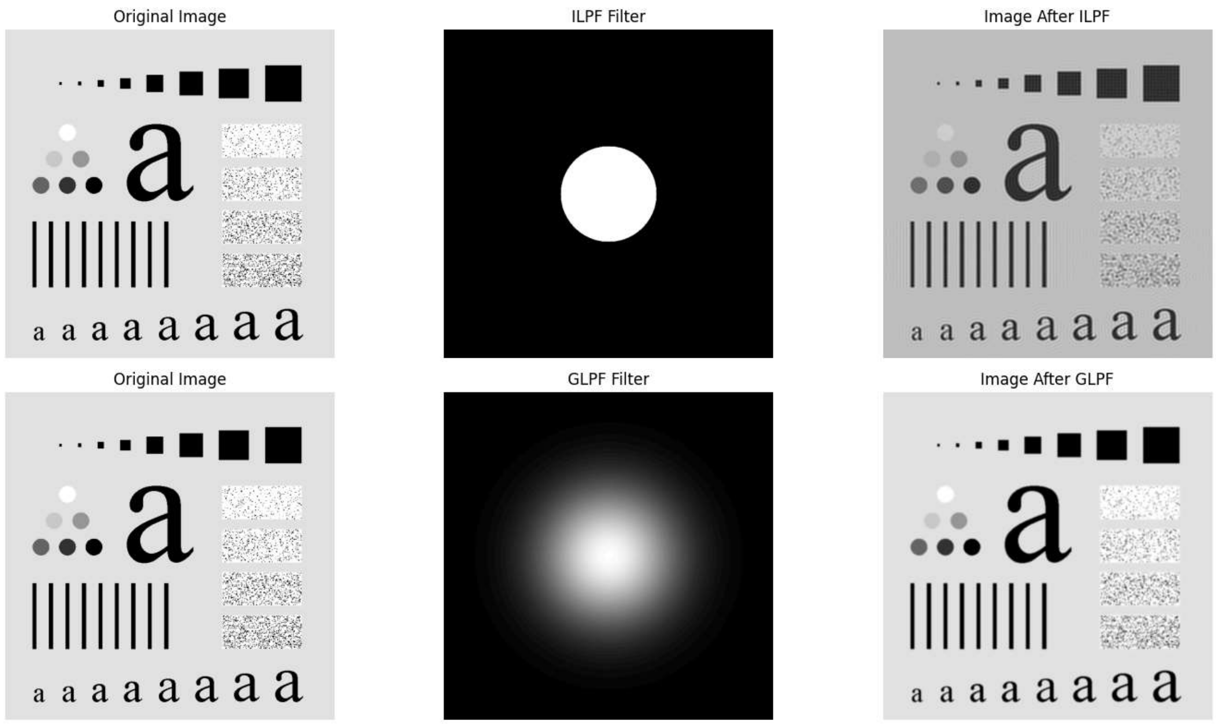
Q2 ILPF and GLPF

In [63]:

```
# Gaussian Low-Pass Filter (GLPF)
def GLPF(rows, cols, D_0):
    x = np.arange(0, rows)
    y = np.arange(0, cols)
    crow, ccol = rows // 2, cols // 2
    X, Y = np.meshgrid(x, y, indexing='ij')
    D_uv = np.sqrt((X - crow)**2 + (Y - ccol)**2)
    H_GLPF = np.exp(-D_uv**2 / (2 * D_0**2))
    return H_GLPF


# Function to apply Fourier Transform and filter
def apply_filter(img, filter_mask):
    # Perform the 2D Fourier Transform
    f = np.fft.fft2(img)
    fshift = np.fft.fftshift(f)

    # Apply the filter in the frequency domain
    fshift_filtered = fshift * filter_mask

    # Perform the inverse Fourier transform
    f_ishift = np.fft.ifftshift(fshift_filtered)
    img_back = np.fft.ifft2(f_ishift)

    # Return the real part of the image after filtering
    return np.real(img_back)

# Example usage
def ILPF_GLPF(img, d):
    # Get the dimensions of the image
    rows, cols = img.shape

    # Get the ILPF and GLPF filters
    H_ILPF = ILPF(rows, cols, d)
```

```python
    H_GLPF = GLPF(rows, cols, d)

    # Apply the ILPF and GLPF filters
    img_back_ILPF = apply_filter(img, H_ILPF)
    img_back_GLPF = apply_filter(img, H_GLPF)

    return img_back_ILPF, img_back_GLPF, H_ILPF, H_GLPF

# Load and preprocess the image
img = mpimg.imread('characters.tif')


# Apply the ILPF and GLPF filters
img_back_ILPF, img_back_GLPF, H_ILPF, H_GLPF = ILPF_GLPF(img, 100)

# Save the filtered images
cv2.imwrite('img_after_ILPF.png', img_back_ILPF)
cv2.imwrite('img_after_GLPF.png', img_back_GLPF)

# Visualize the results
plt.figure(figsize=(15, 8))

# Original Image
plt.subplot(2, 3, 1)
plt.title("Original Image")
plt.imshow(img, cmap='gray')
plt.axis('off')

# ILPF Filter
plt.subplot(2, 3, 2)
plt.title("ILPF Filter")
plt.imshow(H_ILPF, cmap='gray')
plt.axis('off')

# Image After ILPF
plt.subplot(2, 3, 3)
plt.title("Image After ILPF")
plt.imshow(img_back_ILPF, cmap='gray')
plt.axis('off')

# Original Image
plt.subplot(2, 3, 4)
plt.title("Original Image")
plt.imshow(img, cmap='gray')
plt.axis('off')

# GLPF Filter
plt.subplot(2, 3, 5)
plt.title("GLPF Filter")
plt.imshow(H_GLPF, cmap='gray')
plt.axis('off')

# Image After GLPF
plt.subplot(2, 3, 6)
plt.title("Image After GLPF")
plt.imshow(img_back_GLPF, cmap='gray')
```

```
plt.axis('off')

plt.tight_layout()
plt.show()
```



Conclusion Q2

The **Ideal Low-Pass Filter (ILPF)** demonstrates significant blurring and ringing effects in the spatial domain. This occurs because the inverse Fourier transform of the ILPF's box filter H(ω)) results in a sinc function, which inherently contains oscillations (side lobes). When convolved with the image, this sinc function introduces the characteristic ringing artifacts, particularly around sharp transitions or edges in the image. This behavior is a direct consequence of the abrupt cut-off in the frequency domain, which translates to oscillations in the spatial domain.

In contrast, the **Gaussian Low-Pass Filter (GLPF)** behaves more smoothly. The inverse Fourier transform of the GLPF is also Gaussian, which has no side lobes, thus eliminating the ringing effect. This results in a much smoother and visually appealing output, with a gradual attenuation of frequencies instead of an abrupt cut-off. The lack of oscillations makes GLPF more effective in avoiding unwanted artifacts, while still achieving blurring, making it ideal for applications where a natural, softer appearance is desired without the side effects of ringing.

In [ ]: