图像变换与去噪

实验目的

1. 试对附件中的图片施加仿射变换、透视变换，输出相应的图片。
2. 试滤除图片中的噪声信号。

实验环境

- OpenCV2
- NumPy
- MatPlotLib

In [2]:
```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
```

实验原理

二维平面上点$(x,y)^T$的齐次坐标表示为$(x,y,1)^T$

仿射变换（Affine Transformation）

仿射变换是一种保持平行性的几何变换，可以通过矩阵乘法表示。它包含旋转、缩放、平移和剪切等操作。仿射变换保留了直线性和平行性，但不保留距离和角度。

数学上，仿射变换可以表示为以下形式：

$$\mathbf{T}(\mathbf{x}) = \mathbf{A} \cdot \mathbf{x} + \mathbf{b}$$

其中：

- $\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}$ 表示二维空间中的点。
- $\mathbf{A} = \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix}$ 是一个 2×2 矩阵，定义了旋转、缩放和剪切。
- $\mathbf{b} = \begin{pmatrix} b_x \\ b_y \end{pmatrix}$ 是一个平移向量，定义了变换中的平移部分。

用齐次坐标也可以表示为 $\begin{pmatrix} t_{11} & t_{12} & b_x \\ t_{21} & t_{22} & b_y \\ 0 & 0 & 1 \end{pmatrix}$

透视变换（Perspective Transformation）

透视变换，也就是射影变换，是一种更一般的变换，适用于三维物体在二维平面上的投影。它会导致直线在图像上收敛，类似于人类的视角感知，常用于处理图像中的视角变化。

透视变换可以用一个 3×3 的矩阵$\mathbf{H} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix}$来表示：

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \mathbf{H} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

最终的坐标 $(x', y')^T$ 是通过归一化计算得到的：

$$x' = \frac{x'}{w'}, \quad y' = \frac{y'}{w'}$$

傅里叶变换（Fourier Transform）

傅里叶变换是一种将时域或空间域的信号转换为频域表示的数学变换。它在信号处理、图像处理等领域中广泛应用，主要用于分析信号的频率成分。

二维傅里叶变换（适用于图像）将空间域中的图像 $f(x, y)$ 转换为频域表示 $F(u, v)$，其数学形式如下：

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-j2\pi(ux+vy)} \, dx \, dy$$

傅里叶变换的物理意义在于，它将图像中的每个像素值分解为不同的频率成分，可以用于检测图像中的边缘、去除噪声、图像压缩等任务。

实验方法

仿射变换

PIL提供的`transform`方法可以直接将变换矩阵作用于图像，传入`method=Transform.AFFINE`的同时传入变换矩阵的逆矩阵的6个参数即可。

In [3]:
```python
def affine_trans(pic: Image.Image,
                 T: np.ndarray, b: np.ndarray) -> Image.Image:
    inv_T = np.linalg.inv(T)
    inv_b = -b
    data = np.concatenate((inv_T, inv_b.reshape(2, 1)), axis=1).flatten()
    pic = pic.transform(pic.size, Image.AFFINE, data, Image.BICUBIC)
    return pic
```

透视变换

同上。传入`method=Transform.PERSPECTIVE`的同时传入变换矩阵的逆矩阵的前8个参数即可。

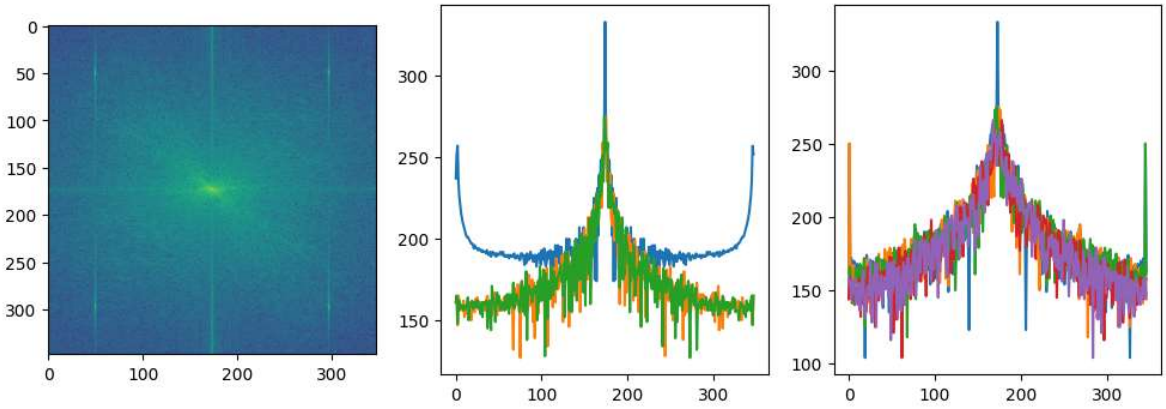
In [4]:
```
def perspective_trans(pic: Image.Image, H: np.ndarray) -> Image.Image:
    inv_H = np.linalg.inv(H)
    data = inv_H.flatten()[:8]
    pic = pic.transform(pic.size, Image.PERSPECTIVE, data, Image.BICUBIC)
    return pic
```

去噪

观察到图像噪声为有规律的条纹，考虑到其为固定频率的噪声。先对图像进行傅里叶变换得到频谱，画出$f_y = 0$附近和$f_x = 0$附近切片的曲线：

In [5]:
```
img = np.array(Image.open('lena.png').convert('L'))
freq = np.fft.fft2(img)
freq = np.fft.fftshift(freq)
freq = 20*np.log(np.abs(freq) + 1)
freq = freq.astype(np.uint16)
fig, ax = plt.subplots(1, 3, figsize=(12, 4))
# f_y = 0
ax[1].plot(freq[:, freq.shape[1]//2])
ax[1].plot(freq[:, freq.shape[1]//2+1])
ax[1].plot(freq[:, freq.shape[1]//2-1])
# f_x = 0
ax[2].plot(freq[freq.shape[0]//2, :])
ax[2].plot(freq[freq.shape[0]//2+1, :])
ax[2].plot(freq[freq.shape[0]//2-1, :])
ax[2].plot(freq[freq.shape[0]//2+2, :])
ax[2].plot(freq[freq.shape[0]//2-2, :])
freq = Image.fromarray(freq)
ax[0].imshow(freq)
plt.show
```

Out[5]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



发现噪声主要由三部分组成：

1. $f_y = -123, 124$且$|f_x| > 50$的位置有四条带状噪声；
2. $f_y = 0$且$|f_x| > 50$的位置有两条带状噪声；
3. $f_x = 0$附近且$|f_y| > 170$的位置有两个点状噪声。 可以通过掩码覆盖这些位置，然后从周围采样进行插值填充来消除这些噪声。

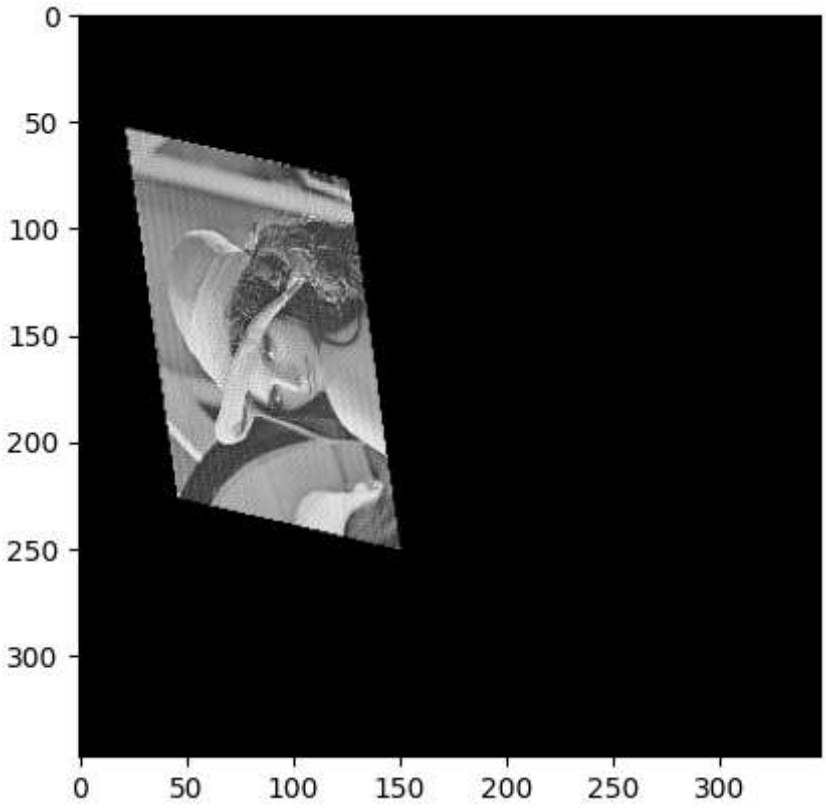
实验过程

仿射变换

利用矩阵 $\begin{pmatrix} 0.07 & 0.3 & 100 \\ 0.5 & 0.07 & 50 \\ 0 & 0 & 1 \end{pmatrix}$ 进行仿射变换。

In [6]:
```
T = np.array([[0.07, 0.3], [0.5, 0.07]])
b = np.array([100, 50])
pic = Image.open('lena.png').convert('L')
pic = affine_trans(pic, T, b)
plt.imshow(pic, cmap='gray')
```

Out[6]:
```
<matplotlib.image.AxesImage at 0xff4437613650>
```



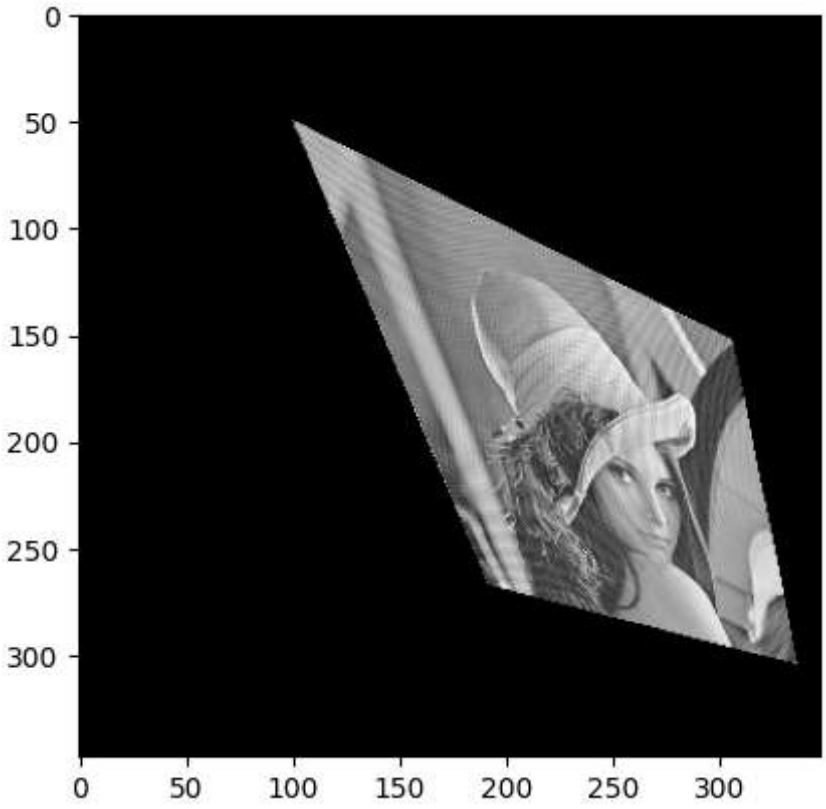
仿射变换保持了直线的平行性。

透视变换

利用矩阵 $\begin{pmatrix} 1 & 0.5 & 100 \\ 0.5 & 1 & 50 \\ 0.001 & 0.001 & 1 \end{pmatrix}$ 进行透视变换。

In [7]:
```
H = np.array([[1, 0.5, 100], [0.5, 1, 50], [0.001, 0.001, 1]])
pic = Image.open('lena.png').convert('L')
pic = perspective_trans(pic, H)
plt.imshow(pic, cmap='gray')
```

Out[7]:
```
<matplotlib.image.AxesImage at 0xff443774a660>
```



透视变换下直线的平行性已不再保持。

去噪

写出根据掩码在频谱图上插值的函数。

In [8]:
```python
import numpy as np
from scipy.interpolate import griddata

def interpolate(freq: np.ndarray, mask: np.ndarray) -> np.ndarray:
    """
    Interpolates the masked regions in a frequency array using neigh
    boring non-masked pixels.

    Args:
        freq (np.ndarray): The frequency array.
        mask (np.ndarray): A binary mask array where 1 represents the
        masked regions to interpolate.

    Returns:
        np.ndarray: The interpolated frequency array.
    """
    # Get the coordinates of the masked (noisy) and non-masked (valid) regions
    masked_coords = np.argwhere(mask == 1)  # Coordinates where mask is 1 (maske
    valid_coords = np.argwhere(mask == 0)   # Coordinates where mask is 0 (valia

    # Get frequency values at the non-masked (valid) coordinates
    valid_values = freq[mask == 0]

    # Use griddata to interpolate the masked regions
    interpolated_values = griddata(valid_coords, valid_values,
                                   masked_coords, method='nearest')
```

```
    # Handle NaN values (if any) by using nearest-neighbor interpolation as a fa
    nan_mask = np.isnan(interpolated_values)
    if np.any(nan_mask):
        interpolated_values[nan_mask] = griddata(valid_coords, valid_values,
                                                 masked_coords[nan_mask], method

    # Create a copy of the frequency array and fill in the interpolated values i
    freq_interp = freq.copy()
    for i, (x, y) in enumerate(masked_coords):
        freq_interp[x, y] = interpolated_values[i]

    return freq_interp
```

进行去噪。

In [9]:

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.fft import fft2, ifft2, fftshift, ifftshift
from PIL import Image

# Load the image and convert to grayscale
img = Image.open('lena.png').convert('L')
img = np.array(img)

# Perform 2D FFT
freq = fftshift(fft2(img))
x_c, y_c = img.shape[0] // 2, img.shape[1] // 2

# Create a mask where 1 indicates noise and 0 indicates valid data
mask = np.ones_like(img)
w_b = 1 # Width of the band
y_1 = -123
x_1 = 50
y_2 = 124
x_2 = -50
x_3 = -50
x_4 = 50
y_5 = -170
y_6 = 170

mask = np.zeros_like(img, dtype=np.uint8)
mask[x_c+x_1:, y_c+y_1-w_b:y_c+y_1+w_b] = 1
mask[:x_c-x_1, y_c+y_1-w_b:y_c+y_1+w_b] = 1
mask[x_c+x_2:, y_c+y_2-w_b:y_c+y_2+w_b] = 1
mask[:x_c+x_2, y_c+y_2-w_b:y_c+y_2+w_b] = 1
mask[:x_c+x_3, y_c] = 1
mask[x_c+x_4:, y_c] = 1
mask[x_c-w_b+1:x_c+w_b+1, :y_c+y_5] = 1
mask[x_c-w_b-4:x_c+w_b, y_c+y_6:] = 1

# Apply interpolation to the masked regions
freq_interp = interpolate(freq, mask)

# Perform inverse FFT to get the filtered image
filtered_freq = ifftshift(freq_interp)
img_filtered = ifft2(filtered_freq).real

# Clip values and convert to uint8
img_filtered = np.clip(img_filtered, 0, 255).astype(np.uint8)
```

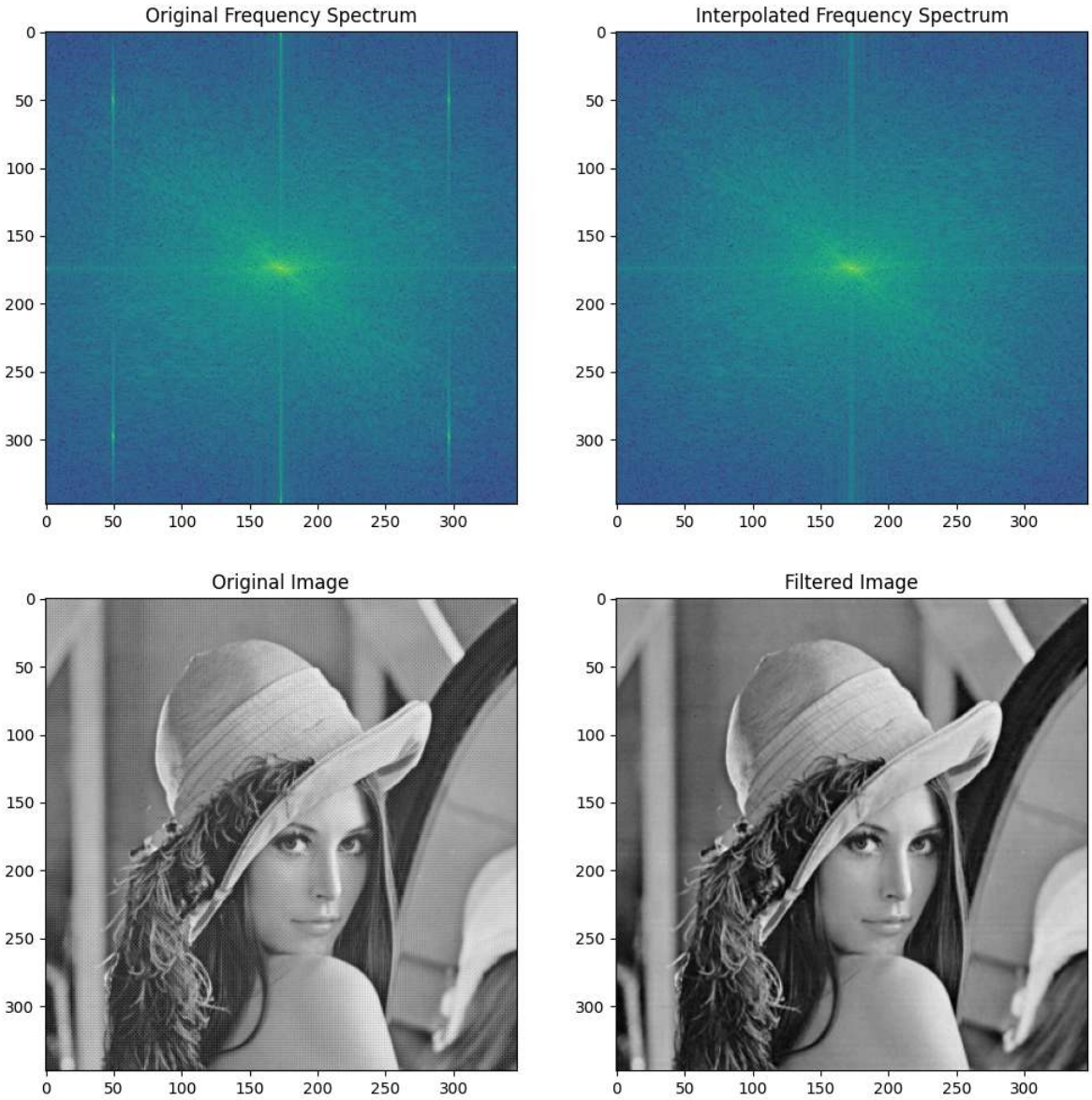
```python
# 2x2 Subplots
fig, axs = plt.subplots(2, 2, figsize=(12, 12))

# Original frequency spectrum
axs[0, 0].imshow(np.log(1 + np.abs(freq)))
axs[0, 0].set_title('Original Frequency Spectrum')

# Interpolated frequency spectrum
axs[0, 1].imshow(np.log(1 + np.abs(freq_interp)))
axs[0, 1].set_title('Interpolated Frequency Spectrum')

# Original and filtered image
axs[1, 0].imshow(img, cmap='gray')
axs[1, 0].set_title('Original Image')
axs[1, 1].imshow(img_filtered, cmap='gray')
axs[1, 1].set_title('Filtered Image')

plt.show()
```



去除3组噪声频率之后，可见图像的噪声基本已经去除干净。