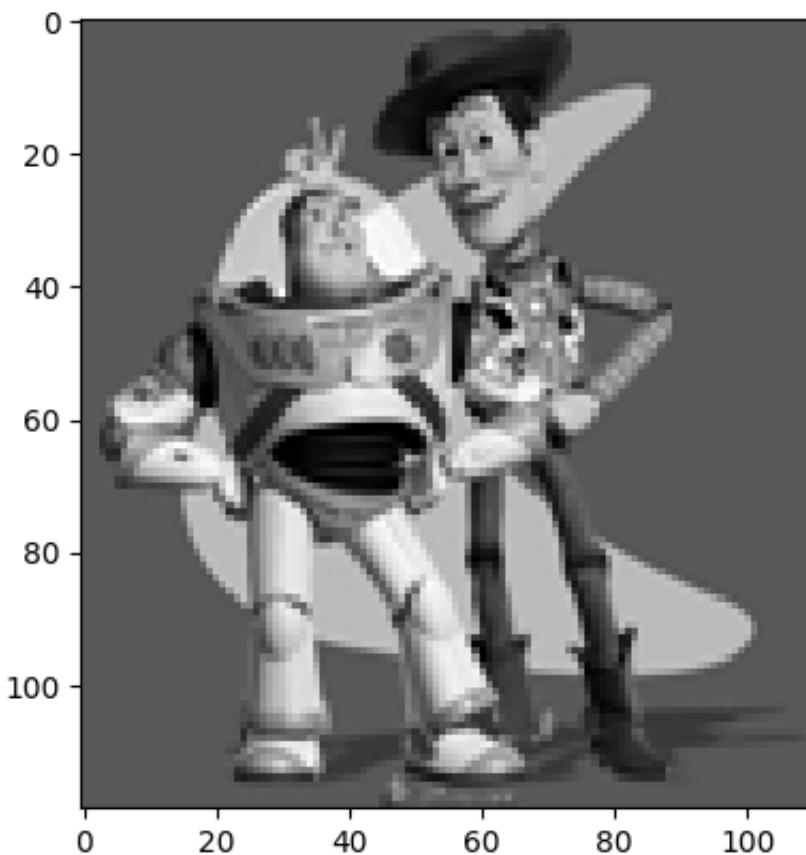


```
In [5]: import cv2
import scipy
import matplotlib.pyplot as plt
import numpy as np
```

Toy problem

```
In [3]: img = cv2.imread('toy/toy_problem.png') #read image
plt.imshow(img)
plt.show()
print(img.shape)
```



(119, 110, 3)

```
In [4]: def toy_problem(s):
    # Get image dimensions
    h, w = s.shape

    # Create im2ind matrix
    im2ind = np.reshape(np.arange(0, h*w), [h, w])

    # Initialize sparse matrix A and vector b
```

```
A = scipy.sparse.lil_matrix((2*h*w + 1, h*w))
b = np.zeros(2*h*w + 1)
# Populate A and b with constraints for gradient objectives
e = 0 # equation counter
for i in range(h):
    for j in range(w):
        # Objective 1
        if j > 0:
            e += 1
            A[e, im2ind[i, j]] = 1
            A[e, im2ind[i, j-1]] = -1
            b[e] = s[i, j] - s[i, j-1]

        # Objective 2
        if i > 0:
            e += 1
            A[e, im2ind[i, j]] = 1
            A[e, im2ind[i-1, j]] = -1
            b[e] = s[i, j] - s[i-1, j]

    # objective 3
    e += 1
    A[e, im2ind[0, 0]] = 1
    b[e] = s[0, 0]

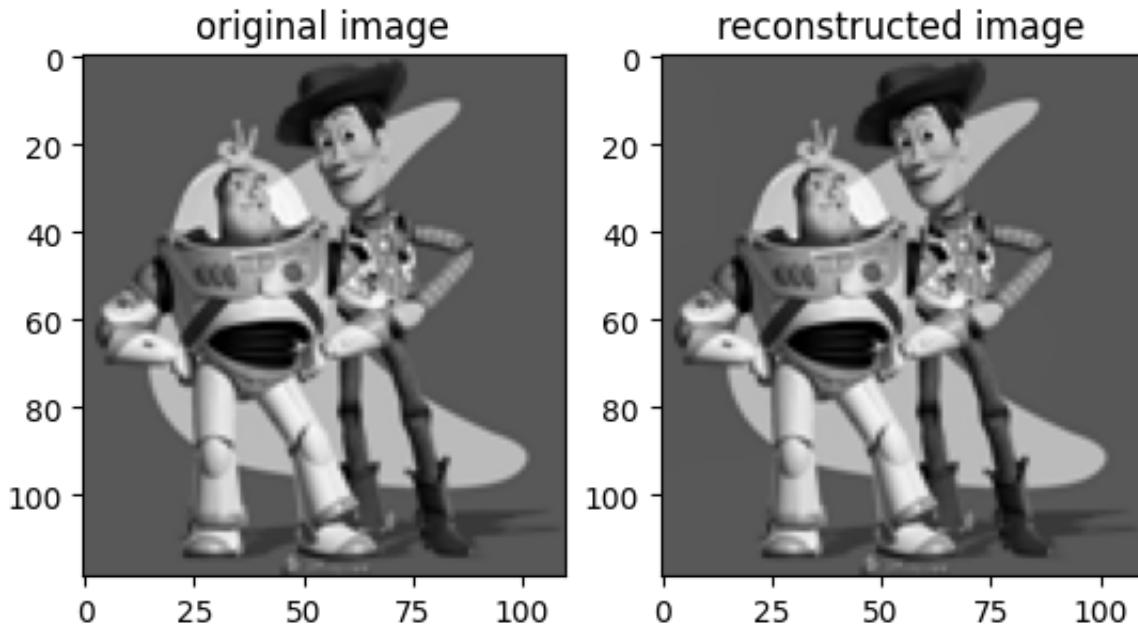
    # Solve for variables v
    v = scipy.sparse.linalg.lsqr(A, b)[0]

    # Create output image using solved values
    output_image = np.reshape(v, [h, w])
    # print(im2ind)
    # print(v)
    # print(b)

return output_image
```

```
In [5]: result_img = np.dstack((toy_problem(img[:, :, 0].astype(float)), toy_problem(img,
fig, axes = plt.subplots(1,2)
axes[0].set_title("original image")
axes[0].imshow(img)
axes[1].set_title("reconstructed image")
axes[1].imshow(result_img.astype(np.uint8))
```

```
Out[5]: <matplotlib.image.AxesImage at 0x165b77970>
```



Copy one source image into other target image

```
In [6]: tree_img = cv2.imread('alpha/tree_image.jpg')
rgb_tree_img = cv2.cvtColor(tree_img, cv2.COLOR_BGR2RGB)
tree_img = cv2.resize(tree_img, (1280,853))
rgb_tree_img = cv2.resize(rgb_tree_img, (1280,853))
gray_tree_img = cv2.cvtColor(tree_img, cv2.COLOR_BGR2GRAY)
```

```
In [7]: moon_img = cv2.imread('alpha/moon.jpg')
shift_filter = np.array([
    [1,0,0],
    [0,0,0],
    [0,0,0]
])
gray_moon_img = cv2.cvtColor(moon_img, cv2.COLOR_BGR2GRAY)

from scipy import ndimage

# Apply thresholding to the grayscale image
# _, thresh = cv2.threshold(gray_moon_img, 100, 180, cv2.THRESH_BINARY)

# Create a mask from the thresholded image
# mask = np.zeros_like(thresh, dtype=np.uint8)
# mask[thresh > 0] = 255

#shift image top left corner
shifted_B_img = ndimage.convolve(moon_img[:, :, 0], shift_filter, mode = 'constant')
shifted_G_img = ndimage.convolve(moon_img[:, :, 1], shift_filter, mode = 'constant')
shifted_R_img = ndimage.convolve(moon_img[:, :, 2], shift_filter, mode = 'constant')
```

```

for i in range(300):
    shifted_B_img = ndimage.convolve(shifted_B_img, shift_filter, mode = 'constant')
    shifted_G_img = ndimage.convolve(shifted_G_img, shift_filter, mode = 'constant')
    shifted_R_img = ndimage.convolve(shifted_R_img, shift_filter, mode = 'constant')

# shifted_image =(np.stack((shifted_B_img,shifted_G_img,shifted_R_img))) .astype(np.uint8)

```

In [8]: `mask = np.zeros((gray_moon_img.shape), dtype= np.uint8)
cv2.circle(mask, (480,118), 85, 1, -1, cv2.LINE_AA)`

```

# Apply the mask to the original image
# masked_image = cv2.bitwise_and(moon_img, moon_img, mask=mask)
# print(masked_image.shape)

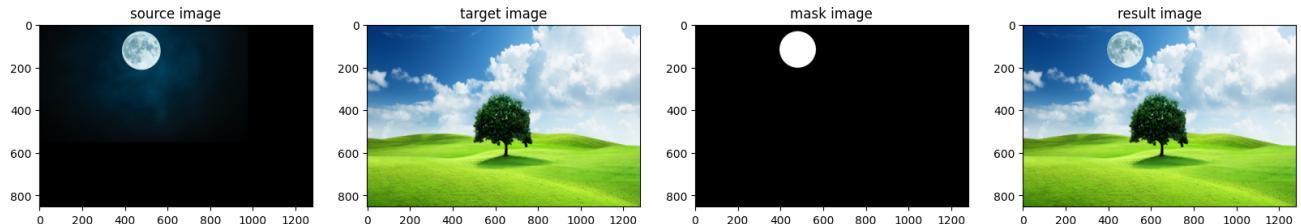
```

Out[8]: `array([[0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 ...,
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)`

In [9]: `shifted_image =(np.dstack((shifted_R_img,shifted_G_img,shifted_B_img))) .astype(np.uint8)`

In [10]: `result_B = np.multiply(tree_img[:, :, 0], (1-mask)) + np.multiply(shifted_B_img, mask)
result_G = np.multiply(tree_img[:, :, 1], (1-mask)) + np.multiply(shifted_G_img, mask)
result_R = np.multiply(tree_img[:, :, 2], (1-mask)) + np.multiply(shifted_R_img, mask)
result =(np.dstack((result_R,result_G,result_B))).astype(np.uint8)
fig, axes = plt.subplots(1,4, figsize= (20,20))
axes[0].set_title('source image')
axes[0].imshow(shifted_image)
axes[1].set_title('target image')
axes[1].imshow(rgb_tree_img)
axes[2].set_title('mask image')
axes[2].imshow(mask, cmap = 'gray')
axes[3].set_title('result image')
axes[3].imshow(result)`

Out[10]: `<matplotlib.image.AxesImage at 0x16a03ed40>`



Alpha blending

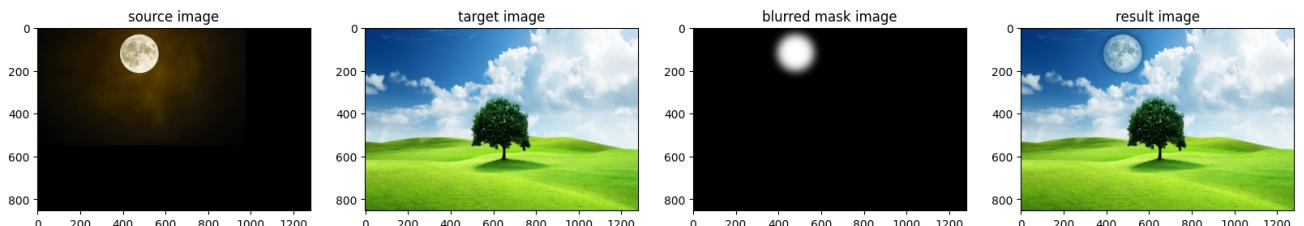
```
In [11]: # blur_mask = cv2.GaussianBlur(mask, (3, 3), 10)
# print(blur_mask)
# plt.imshow(blur_mask, cmap = 'gray')
# blur_mask = cv2.GaussianBlur(mask, (3, 3), 0)
# blur_mask = blur_mask.astype('float32')
from skimage import filters

blur_mask = filters.gaussian(mask, sigma=20)*255
```

```
In [12]: def alpha_blend(s, t, m):
    result = []
    for i in range(2, -1, -1):
        result.append(np.multiply(t[:, :, i], (1 - m)) + np.multiply(s[:, :, i], m))
    return (np.dstack(result)).astype(np.uint8)

shifted_image = (np.dstack((shifted_B_img, shifted_G_img, shifted_R_img))).astype(np.uint8)
fn_result = alpha_blend(shifted_image, tree_img, blur_mask)
fig, axes = plt.subplots(1, 4, figsize=(20, 20))
axes[0].set_title('source image')
axes[0].imshow(shifted_image)
axes[1].set_title('target image')
axes[1].imshow(rgb_tree_img)
axes[2].set_title('blurred mask image')
axes[2].imshow(blur_mask, cmap='gray')
axes[3].set_title('result image')
axes[3].imshow(fn_result)
```

Out[12]: <matplotlib.image.AxesImage at 0x17841a830>



Gradient domain blending

```
In [165...]: s = cv2.imread('gradient1/source_01_newsource.png')
t = cv2.imread('gradient1/target_01.jpg')
m = cv2.imread('gradient1/target_01_mask.png')
```

```
In [14]: def gradient_domain_editing(S, T, mask):
    h, w = T.shape # Assuming S, T, and mask are grayscale images with shape (h, w)
```

```
# Create variable index mapping
im2ind = np.arange(h * w).reshape(h, w)

# Initialize sparse matrix A and vector b
A = scipy.sparse.lil_matrix((2 * (h * w) + 1, h * w), dtype=np.float64)
b = np.zeros(2 * (h * w) + 1, dtype=np.float64)

# Equation counter
e = 0
for y in range(h):
    for x in range(w):
        if x + 1 < w:
            # Horizontal gradient
            A[e, im2ind[y, x]] = 1
            A[e, im2ind[y, x + 1]] = -1
            if mask[y, x]: # check if in R range
                b[e] = S[y, x] - S[y, x + 1]
            else: # boundary
                b[e] = T[y, x] - T[y, x + 1]
        e += 1

        if y + 1 < h:
            # Vertical gradient
            A[e, im2ind[y, x]] = 1
            A[e, im2ind[y + 1, x]] = -1
            if mask[y, x]:
                b[e] = S[y, x] - S[y + 1, x]
            else:
                b[e] = T[y, x] - T[y + 1, x]
        e += 1

A[e, im2ind[0, 0]] = 1
b[e] = T[0, 0]

# Solve the least squares problem
v = scipy.sparse.linalg.lsqr(A.tocsr(), b)[0]

# Reshape the solution to the image shape
N = np.reshape(v, (h, w))

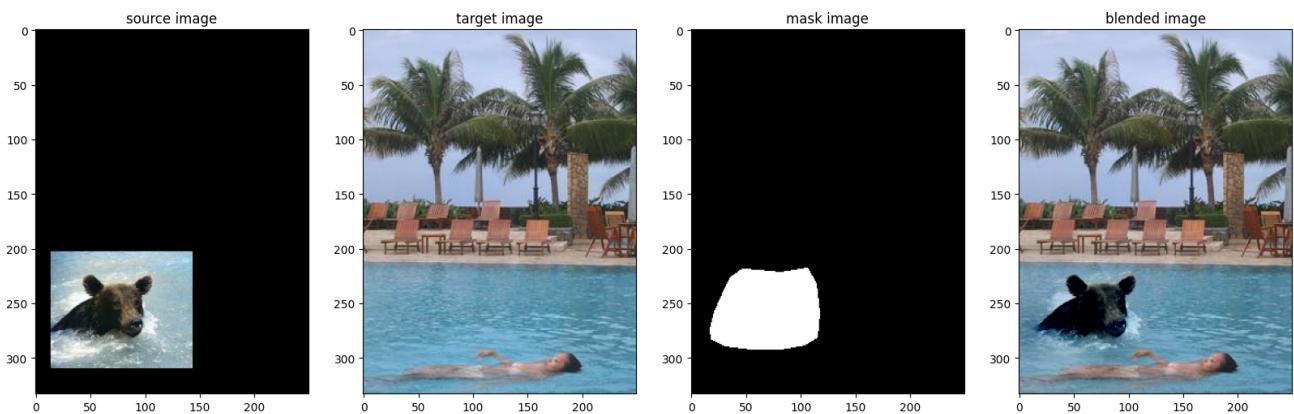
return np.clip(N, 0, 255).astype(np.uint8)

def generate_gradient_image(s, t, m):
    N_B = gradient_domain_editing(s[:, :, 0].astype(np.float64), t[:, :, 0].astype(np.float64))
    N_G = gradient_domain_editing(s[:, :, 1].astype(np.float64), t[:, :, 1].astype(np.float64))
    N_R = gradient_domain_editing(s[:, :, 2].astype(np.float64), t[:, :, 2].astype(np.float64))
    N = np.dstack((N_R, N_G, N_B))
    return N
```

Example 1

```
In [166...]: N = generate_gradient_image(s,t,m)
fig, axes = plt.subplots(1,4, figsize= (20,20) )
axes[0].set_title('source image')
axes[0].imshow(cv2.cvtColor(s, cv2.COLOR_BGR2RGB))
axes[1].set_title('target image')
axes[1].imshow(cv2.cvtColor(t, cv2.COLOR_BGR2RGB))
axes[2].set_title('mask image')
axes[2].imshow(cv2.cvtColor(m, cv2.COLOR_BGR2RGB))
axes[3].set_title('blended image')
axes[3].imshow(N)
```

Out[166...]: <matplotlib.image.AxesImage at 0x2e80e0910>



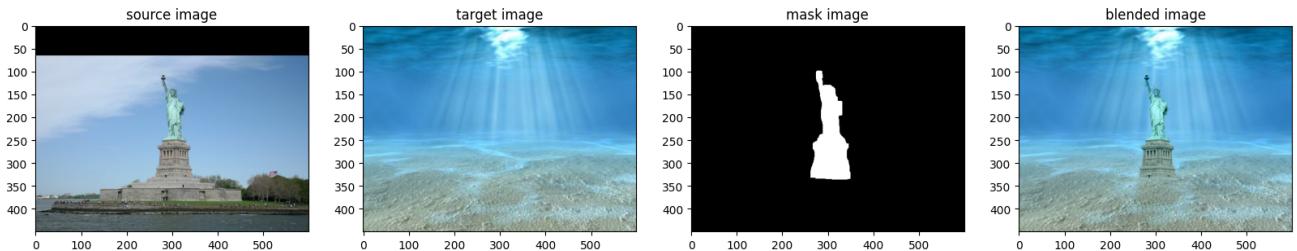
Example 2

```
In [16]: s = cv2.imread('gradient2/source1.jpg')
t = cv2.imread('gradient2/target1.jpg')
m = cv2.imread('gradient2/mask1.png')
y_max, x_max = t.shape[:-1]
y_min, x_min = 0, 0
x_range = x_max - x_min
y_range = y_max - y_min
offset = (0, 66)
M = np.float32([[1, 0, offset[0]], [0, 1, offset[1]]])
s = cv2.warpAffine(s, M, (x_range, y_range))

N = generate_gradient_image(s,t,m)
fig, axes = plt.subplots(1,4, figsize= (20,20) )
axes[0].set_title('source image')
axes[0].imshow(cv2.cvtColor(s, cv2.COLOR_BGR2RGB))
axes[1].set_title('target image')
axes[1].imshow(cv2.cvtColor(t, cv2.COLOR_BGR2RGB))
axes[2].set_title('mask image')
axes[2].imshow(cv2.cvtColor(m, cv2.COLOR_BGR2RGB))
```

```
axes[3].set_title('blended image')
axes[3].imshow(N)
```

Out[16]: <matplotlib.image.AxesImage at 0x177f4c7c0>

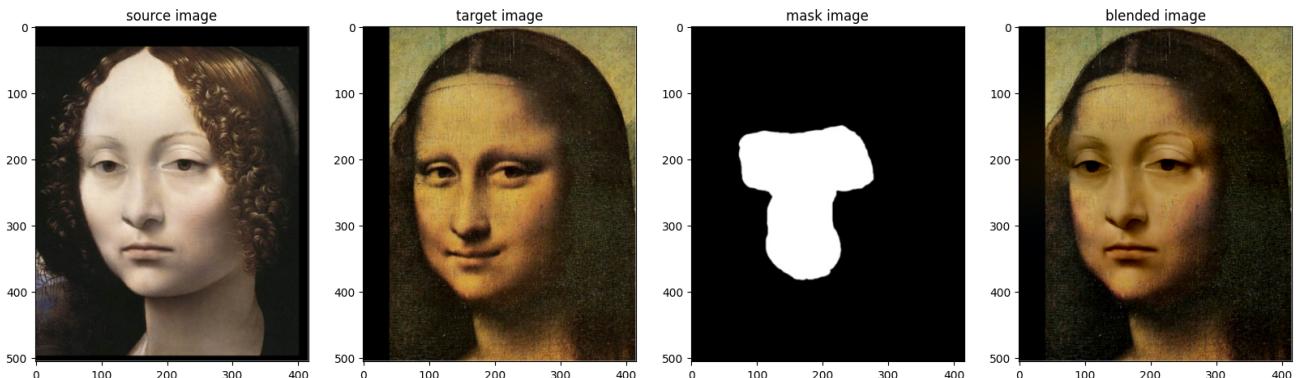


Example 3

```
In [139...]: s = cv2.imread('test1/source.png')
t = cv2.imread('test1/target.png')
m = cv2.imread('test1/mask.png')

N = generate_gradient_image(s,t,m)
fig, axes = plt.subplots(1,4, figsize=(20,20))
axes[0].set_title('source image')
axes[0].imshow(cv2.cvtColor(s, cv2.COLOR_BGR2RGB))
axes[1].set_title('target image')
axes[1].imshow(cv2.cvtColor(t, cv2.COLOR_BGR2RGB))
axes[2].set_title('mask image')
axes[2].imshow(cv2.cvtColor(m, cv2.COLOR_BGR2RGB))
axes[3].set_title('blended image')
axes[3].imshow(N)
```

Out[139...]: <matplotlib.image.AxesImage at 0x2d3e20760>



Laplacian image blending

```
In [22]: def down_sampling(img):
    h, w, c = img.shape
    half_img = np.zeros((h//2, w//2, c))
```

```
for ch in range(c):
    h_y, h_x = 0, 0
    for y in range(0, h, 2):
        h_x = 0
        for x in range(0, w, 2):
            if h_y < h//2 and h_x < w//2:
                half_img[h_y, h_x, ch] = img[y, x, ch]
            h_x += 1
        h_y += 1
return half_img.astype(np.uint8)

def up_sampling(img):
    h, w, c = img.shape
    twice_img = np.zeros((h*2, w*2, c))
    for ch in range(c):
        t_y, t_x = 0, 0
        for y in range(h):
            t_x = 0
            for x in range(w):
                twice_img[t_y: t_y + 2, t_x : t_x + 2, ch] = img[y, x, ch]
                t_x += 2
            t_y += 2
    return twice_img.astype(np.uint8)
```

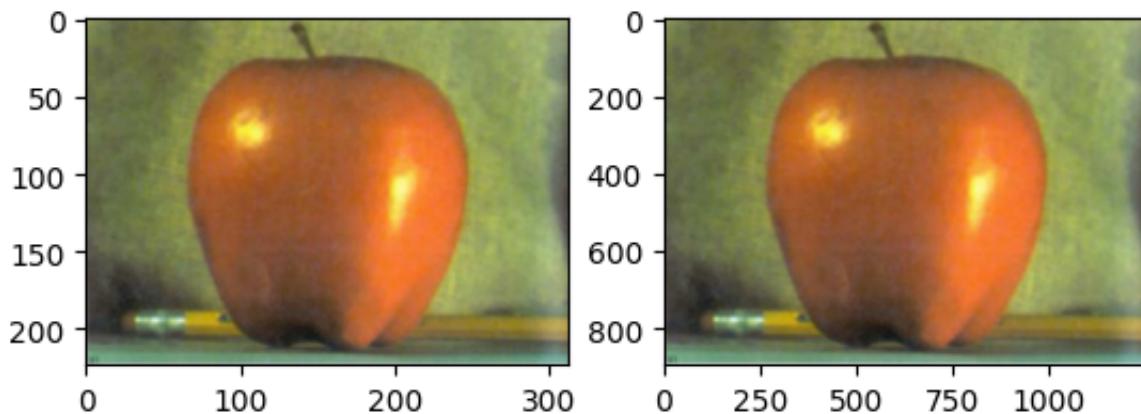
In [23]:

```
def convert_rgb(img):
    result = []
    for i in range(2, -1, -1):
        result.append(img[:, :, i])
    return np.dstack(result)
```

In [24]:

```
#Test
s = cv2.imread('lap/apple.png')
t = cv2.imread('lap/orange.png')
m = cv2.imread('lap/mask.png')
fig, axes = plt.subplots(1, 2)
axes[0].imshow(convert_rgb(down_sampling(s)))
axes[1].imshow(convert_rgb(up_sampling(s)))
```

Out[24]: <matplotlib.image.AxesImage at 0x28d644520>



```
In [25]: def gaussian_one_layer_pyramid(img):
    filtered_img = cv2.GaussianBlur(img, (3,3), 5)
    down_sampled_img = down_sampling(filtered_img)
    return down_sampled_img
```

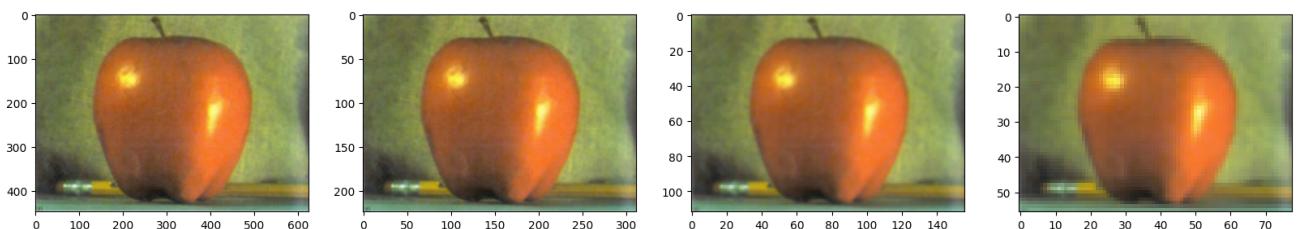
```
In [26]: def gaussian_pyramid(img, num = 3) -> list:
    gpy = []
    gpy.append(img)
    gaussian_img = img.copy()

    for i in range(num):
        gaussian_img = gaussian_one_layer_pyramid(gaussian_img)
        gpy.append(gaussian_img)
    return gpy
```

```
In [27]: #showing gaussian pyramid
gpy = gaussian_pyramid(s)
fig, axes = plt.subplots(1,4, figsize=(20,20))

axes[0].imshow(convert_rgb(gpy[0]))
print(gpy[0].shape)
axes[1].imshow(convert_rgb(gpy[1]))
print(gpy[1].shape)
axes[2].imshow(convert_rgb(gpy[2]))
print(gpy[2].shape)
axes[3].imshow(convert_rgb(gpy[3]))
print(gpy[3].shape)
```

```
(448, 624, 3)
(224, 312, 3)
(112, 156, 3)
(56, 78, 3)
```



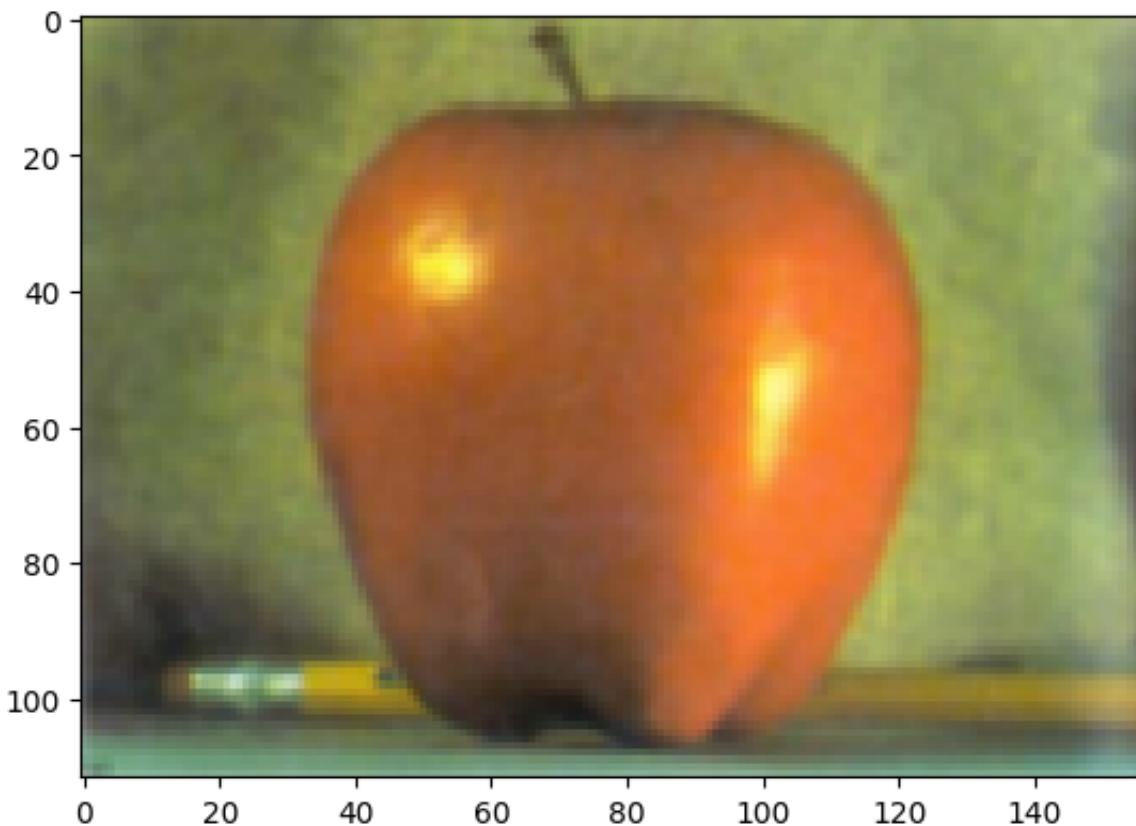
```
In [28]: def laplacian_pyramid(img):
    gpy = gaussian_pyramid(img)
    length = len(gpy)
    lpy = []
    for i in range(length):
        if i + 1 < length:
            up_sampled = up_sampling(gpy[i + 1]) #upsample i + 1 gaussian py
            if (gpy[i].shape != up_sampled).any():
                up_sampled = np.resize(up_sampled, gpy[i].shape)
            lpy.append((gpy[i] - up_sampled).astype(np.uint8)) # subtract fr
    return lpy # one less than gaussian pyramid
```

```
In [29]: def mask_list(m):
    m_list = []
    m_list.append(m)
    down_m = m
    for i in range(3):
        down_m = down_sampling(down_m)
        m_list.append(down_m)
    return m_list
```

```
In [36]: from skimage import filters
def alpha_blending(s, t, m):
    result = []
    t = t.astype(np.float64)
    m = filters.gaussian(m.astype(bool), sigma = 20)
    for i in range(2, -1, -1):
        result.append(np.multiply(t[:, :, i], (1 - m[:, :, i])) + np.multiply(s[
    return (np.dstack(result)).astype(np.uint8)
```

```
In [37]: lpy = laplacian_pyramid(s)
plt.imshow(convert_rgb(lpy[2] + up_sampling(gpy[3])))
```

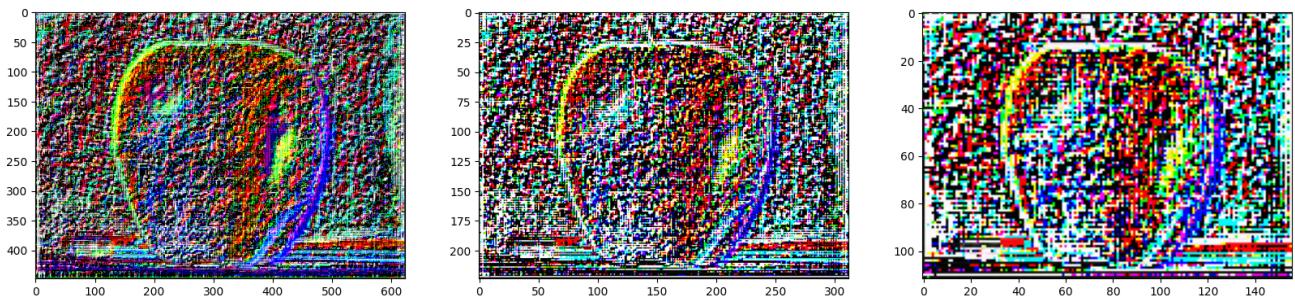
```
Out[37]: <matplotlib.image.AxesImage at 0x28d1710c0>
```



```
In [38]: #showing laplacian pyramid
fig, axes = plt.subplots(1,3, figsize=(20,20))

axes[0].imshow(lpy[0])
axes[1].imshow(lpy[1])
axes[2].imshow(lpy[2])
```

Out[38]: <matplotlib.image.AxesImage at 0x28ea3bbe0>

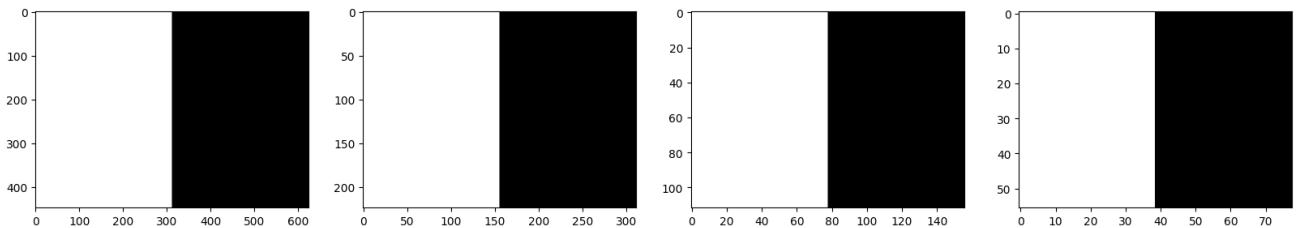
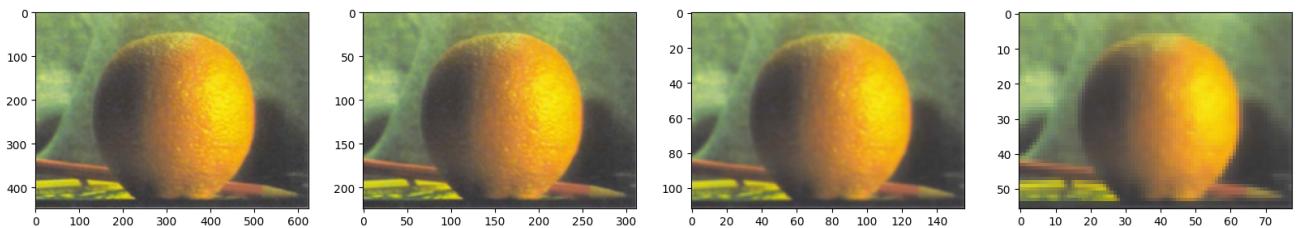
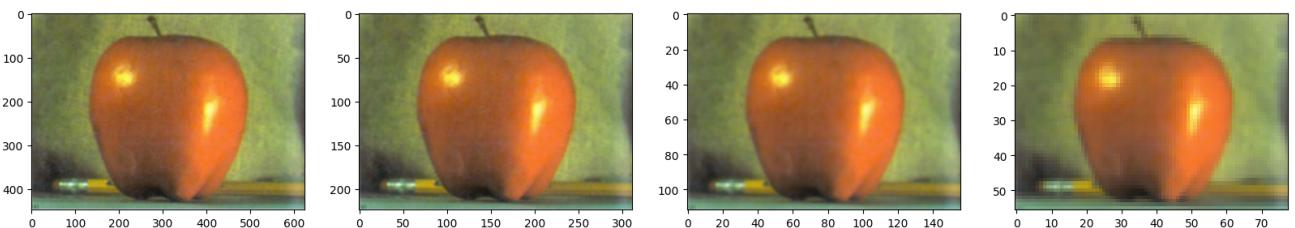


```
In [39]: gps = gaussian_pyramid(s)
gpt = gaussian_pyramid(t)
m_list = mask_list(m)
```

```
In [40]: fig, axes = plt.subplots(3,4, figsize=(20,20))
```

```
axes[0,0].imshow(convert_rgb(gps[0]))
axes[0,1].imshow(convert_rgb(gps[1]))
axes[0,2].imshow(convert_rgb(gps[2]))
axes[0,3].imshow(convert_rgb(gps[3]))
axes[1,0].imshow(convert_rgb(gpt[0]))
axes[1,1].imshow(convert_rgb(gpt[1]))
axes[1,2].imshow(convert_rgb(gpt[2]))
axes[1,3].imshow(convert_rgb(gpt[3]))
axes[2,0].imshow(m_list[0],cmap = 'gray')
axes[2,1].imshow(m_list[1],cmap = 'gray')
axes[2,2].imshow(m_list[2],cmap = 'gray')
axes[2,3].imshow(m_list[3],cmap = 'gray')
```

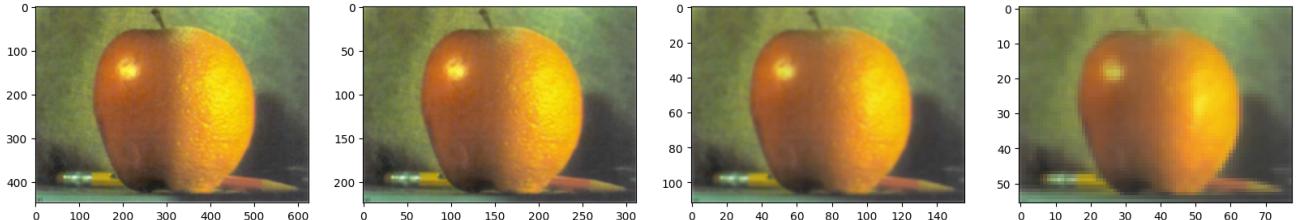
Out[40]: <matplotlib.image.AxesImage at 0x2a01bb520>



```
In [41]: gpb = []
for i in range(len(gps)):
    gpb.append(alpha_blending(gps[i], gpt[i], m_list[i]))
```

```
fig, axes = plt.subplots(1,4, figsize=(20,20))
axes[0].imshow(gpb[0].astype(np.uint8))
axes[1].imshow(gpb[1].astype(np.uint8))
axes[2].imshow(gpb[2].astype(np.uint8))
axes[3].imshow(gpb[3].astype(np.uint8))
```

Out[41]: <matplotlib.image.AxesImage at 0x2a2420910>



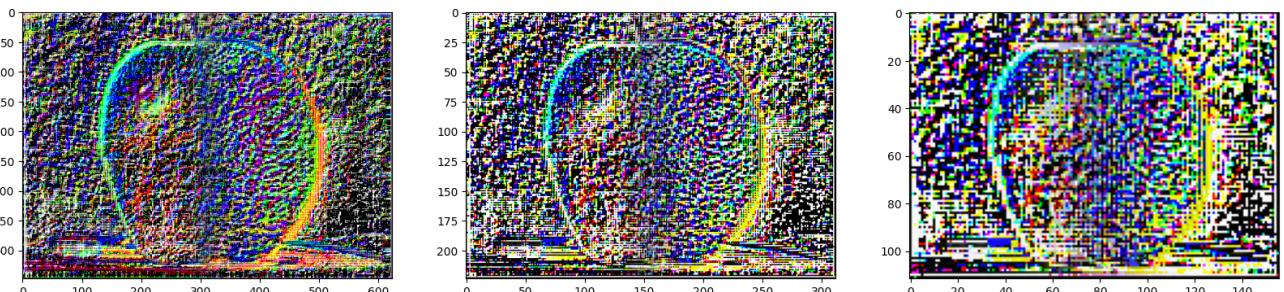
```
In [42]: lps = laplacian_pyramid(s)
lpt = laplacian_pyramid(t)
lm_list = mask_list(m)
```

```
lpb = []
for i in range(len(lps)):
    lpb.append(alpha_blending(lps[i], lpt[i], lm_list[i]))

fig, axes = plt.subplots(1,3, figsize=(20,20))

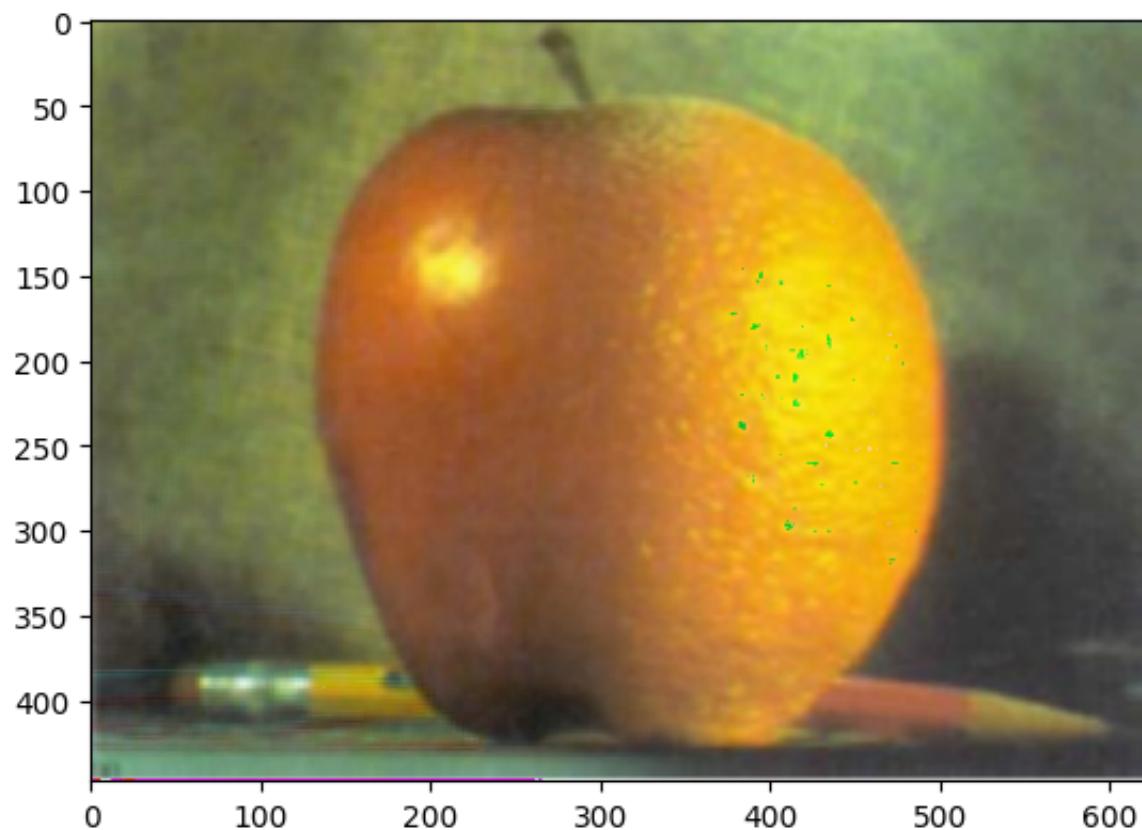
axes[0].imshow(lpb[0].astype(np.uint8))
axes[1].imshow(lpb[1].astype(np.uint8))
axes[2].imshow(lpb[2].astype(np.uint8))
```

Out[43]: <matplotlib.image.AxesImage at 0x2a2504850>



```
In [52]: for i in range(3, 0, -1):
    gpb[i - 1] = lpt[i - 1] + up_sampling(alpha_blending((gps[i]), (gpt[i])),
plt.imshow(gpb[0]))
```

Out[52]: <matplotlib.image.AxesImage at 0x28ed6c5e0>



In []: *# image source:
#In-class notes and google image search*