gradient-domain image processing

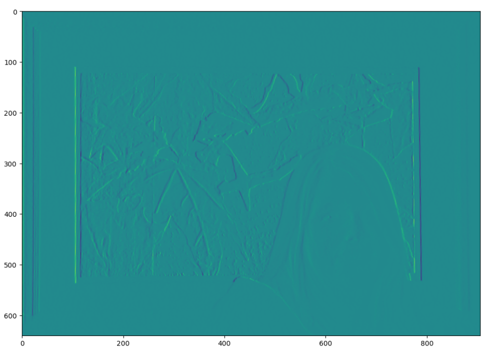
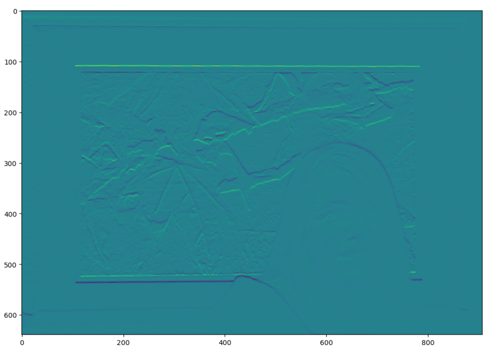
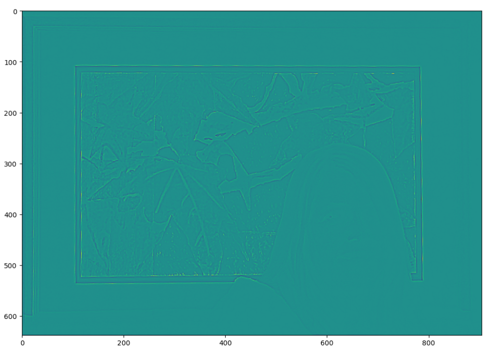
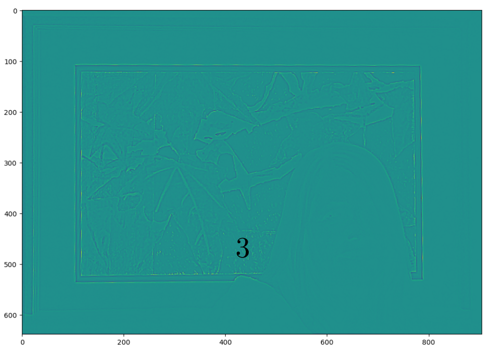
October 25, 2022

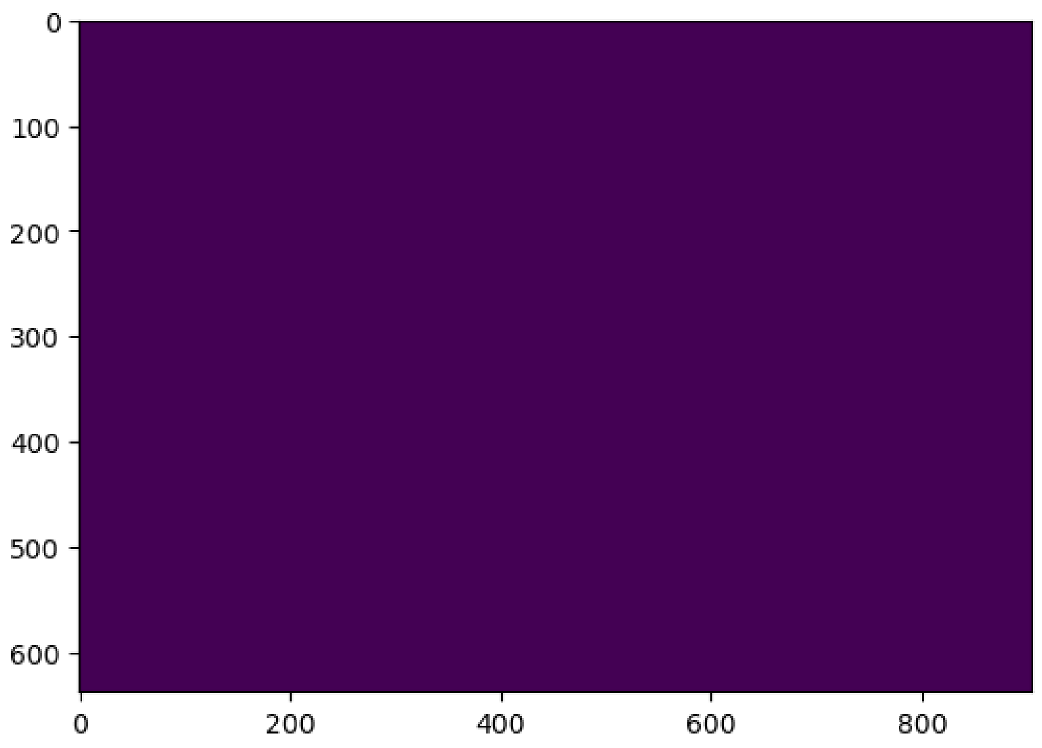
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
from scipy.interpolate import interpn
from scipy.signal import convolve2d
from tqdm import tqdm
```

```
/home/aramesh/anaconda3/envs/comp-photo/lib/python3.10/site-
packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and
<1.23.0 is required for this version of SciPy (detected version 1.23.1
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

[2]:
```
# load images
museum_amb = './data/museum/museum_ambient.png'
museum_flash = './data/museum/museum_flash.png'

museum_amb = cv2.imread(museum_amb,-1)[:,:,:3][:,:,::-1]
museum_flash = cv2.imread(museum_flash,-1)[:,:,:3][:,:,::-1]

# normalize to 0-1
# museum_amb_norm = (museum_amb - np.min(museum_amb)) / (np.max(museum_amb) -␣
 ↪np.min(museum_amb))
# museum_flash_norm = (museum_flash - np.min(museum_flash)) / (np.
 ↪max(museum_flash) - np.min(museum_flash))
museum_amb_norm = museum_amb / 255
museum_flash_norm = museum_flash / 255
```

[3]:
```
def gradient(img) :

    temp = img[:,:].copy()
    row = np.zeros((1,temp.shape[1]))
    col = np.zeros((temp.shape[0]+2,1))
    temp2 = np.vstack((row,np.vstack((temp, row))))
    temp2 = np.hstack((col,np.hstack((temp2, col))))
    img = temp2.copy()
    # we remove the last row and the last column corresponding to img_x at the␣
 ↪end
    # we remove the last col and the last column corresponding to img_y at the␣
 ↪end

    img_x = np.diff(img,n=1,axis=1) # change along row (i.e. diff bw columns)
    img_y = np.diff(img,n=1,axis=0) # change along column (i.e. diff bw rows)
#     print(img_x.shape, img_y.shape)
#     img_x, img_y = img_x[:-1,:], img_y[:,:-1]
#     print(img_x.shape, img_y.shape)
    return img_x, img_y # vector field

def divergence(u,v) :
    u_x = np.diff(u,n=1,axis=1) # change along row (i.e. diff bw columns)
    v_y = np.diff(v,n=1,axis=0) # change along column (i.e. diff bw rows)
    u_x, v_y = u_x[1:-1,:], v_y[:,1:-1]

    out = u_x + v_y
    return out # scalar field

def laplacian(img) :
    kernel = np.array([[0,1,0],[1,-4,1],[0,1,0]])
    out = convolve2d(img, kernel, mode='same', boundary='fill', fillvalue=0)
    return out # scalar field

# testing functions above

museum_amb_norm_gradient_x,museum_amb_norm_gradient_y  =␣
 ↪gradient(museum_amb_norm[:,:,0])
museum_amb_norm_lhs =␣
 ↪divergence(museum_amb_norm_gradient_x,museum_amb_norm_gradient_y )
museum_amb_norm_rhs = laplacian(museum_amb_norm[:,:,0])

fig, ax = plt.subplots(5, figsize=(50,50))
ax[0].imshow(museum_amb_norm[:,:,0])
ax[1].imshow(museum_amb_norm_gradient_x)
ax[2].imshow(museum_amb_norm_gradient_y)
ax[3].imshow(museum_amb_norm_lhs)
ax[4].imshow(museum_amb_norm_rhs)
plt.show()
plt.imshow(np.around(museum_amb_norm_lhs,5) == np.around(museum_amb_norm_rhs,5))
plt.show()
print("np.allclose(museum_amb_norm_lhs, museum_amb_norm_rhs) = ",np.
 ↪allclose(museum_amb_norm_lhs, museum_amb_norm_rhs))
```



```
np.allclose(museum_amb_norm_lhs, museum_amb_norm_rhs) =  True
```

[35]:
```
# Differentiate then re-integrate an image

# Poisson Solver

def conjugate_gradient_descent(I_init, B, I_star_boundary, eps, N, D) :

    I_star = B * I_init + (1-B) * I_star_boundary
    r = B * (D - laplacian(I_star))
    d = r.copy()
    delta_new = np.sum(r*r) # <r,r>
    n = 0

    while np.sqrt(np.sum(r*r)) > eps and n < N :

        if n % 200 == 0 : print(n,np.sqrt(np.sum(r*r)))
        q = laplacian(d)
        eta = delta_new / np.sum(d*q) # n = delta_new / <d,q>

        I_star = I_star + B * (eta * d)
        r = B * (r - eta * q)

        delta_old = delta_new.copy()
        delta_new = np.sum(r*r)

        beta = delta_new / delta_old
        d = r + beta * d

        n = n + 1

    return I_star

def poisson_solver(lap_img, img, N = 1000, eps = 0.001) :

    I_init = np.zeros(img.shape)

    boundary_size = 1
    B = np.ones(img.shape)
    B[0:boundary_size,:], B[-boundary_size:-1,:], B[:,0:boundary_size], B[:
 ↪,-boundary_size:-1] = 0,0,0,0

    I_star_boundary = np.zeros(img.shape)
    I_star_boundary[0:boundary_size,:] = img[0:boundary_size,:].copy()
    I_star_boundary[-boundary_size:-1,:] = img[-boundary_size:-1,:].copy()
    I_star_boundary[:,0:boundary_size] = img[:,0:boundary_size].copy()
    I_star_boundary[:,-boundary_size:-1] = img[:,-boundary_size:-1].copy()

    I_star = conjugate_gradient_descent(I_init, B, I_star_boundary, eps, N,␣
 ↪lap_img)

    return I_star


# Laplacian (Divergence of gradient field)

# lap_img_r = laplacian(museum_amb_norm[:,:,0])
# lap_img_g = laplacian(museum_amb_norm[:,:,1])
# lap_img_b = laplacian(museum_amb_norm[:,:,2])

# I_star_r = poisson_solver(lap_img_r, museum_amb_norm[:,:,0])
# I_star_g = poisson_solver(lap_img_g, museum_amb_norm[:,:,1])
# I_star_b = poisson_solver(lap_img_b, museum_amb_norm[:,:,2])

# I_star = np.concatenate((np.expand_dims(I_star_r,2),np.
 ↪stack((I_star_g,I_star_b),axis=-1)),axis=2)

# fig, ax = plt.subplots(2, figsize = (20,20))
# ax[0].imshow(museum_amb_norm)
# ax[1].imshow(I_star)

# plt.show()
```

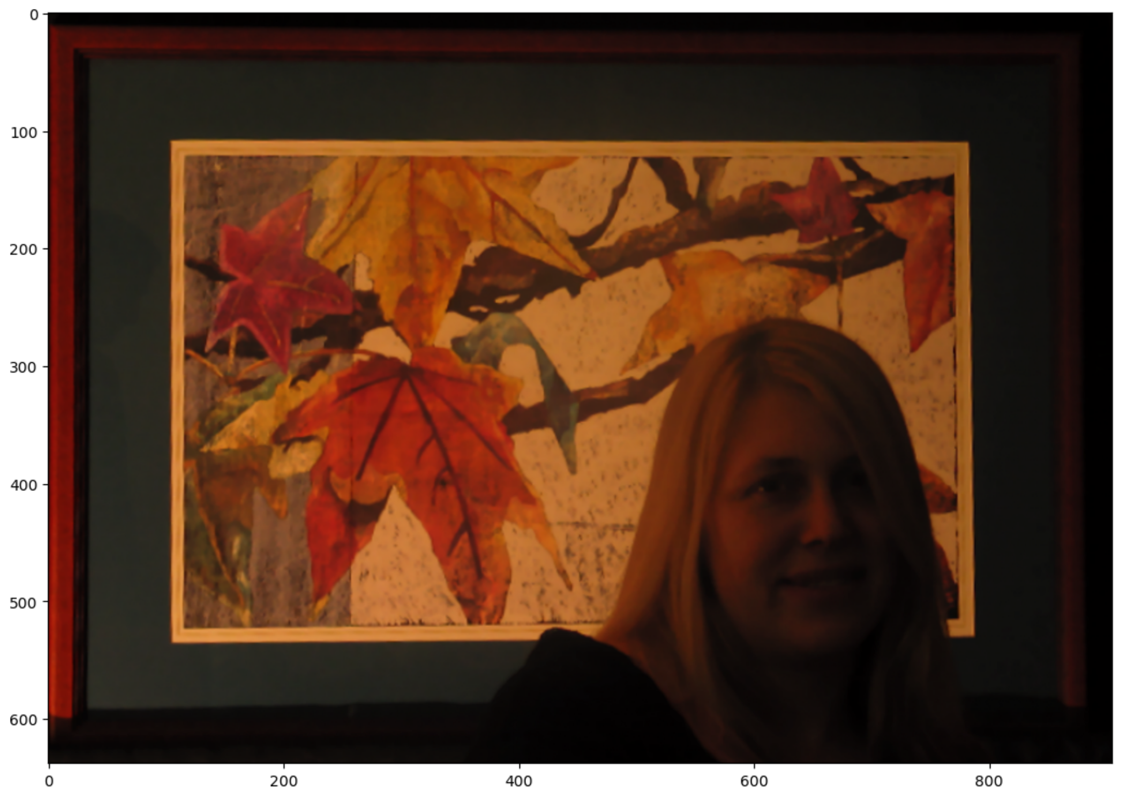
[38]:
```
# Creating fused gradient field

def fuse_gradient_field(img_amb_norm, img_flash_norm, sigma, tau_s) :

    a_grad_x, a_grad_y = gradient(img_amb_norm)
    phi_dash_grad_x, phi_dash_grad_y = gradient(img_flash_norm)

    a_grad_x_org = a_grad_x.copy()
    a_grad_y_org = a_grad_y.copy()
    phi_dash_grad_x_org = phi_dash_grad_x.copy()
    phi_dash_grad_y_org = phi_dash_grad_y.copy()

    a_grad_x, phi_dash_grad_x = a_grad_x[1:,:], phi_dash_grad_x[1:,:]
    a_grad_y, phi_dash_grad_y = a_grad_y[:,1:], phi_dash_grad_y[:,1:]

    M_num = phi_dash_grad_x * a_grad_x + phi_dash_grad_y * a_grad_y
    M_den = np.sqrt(phi_dash_grad_x**2 + phi_dash_grad_y**2) * np.
 ↪sqrt(a_grad_x**2 + a_grad_y**2)
    M = M_num / (M_den+0.00001)
#     plt.imshow(M)
#     plt.show()

#     sigma = 40
#     tau_s = 0.9
    ws = np.zeros(a_grad_x.shape)
    ws[:-1,:-1] = np.tanh(sigma * (img_flash_norm-tau_s))
    ws = (ws - np.min(ws)) / (np.max(ws) - np.min(ws))
#     print(ws.shape, a_grad_x.shape, M.shape, phi_dash_grad_x.shape, a_grad_y.
 ↪shape, phi_dash_grad_y.shape)
    phi_star_grad_x = np.zeros(a_grad_x_org.shape)
    phi_star_grad_y = np.zeros(a_grad_y_org.shape)
    phi_star_grad_x[1:,:] = ws * a_grad_x + (1-ws) * (M * phi_dash_grad_x +␣
 ↪(1-M) * a_grad_x)
    phi_star_grad_y[:,1:] = ws * a_grad_y + (1-ws) * (M * phi_dash_grad_y +␣
 ↪(1-M) * a_grad_y)

    div = divergence(phi_star_grad_x,phi_star_grad_y)

    return div

sigma = 40
```

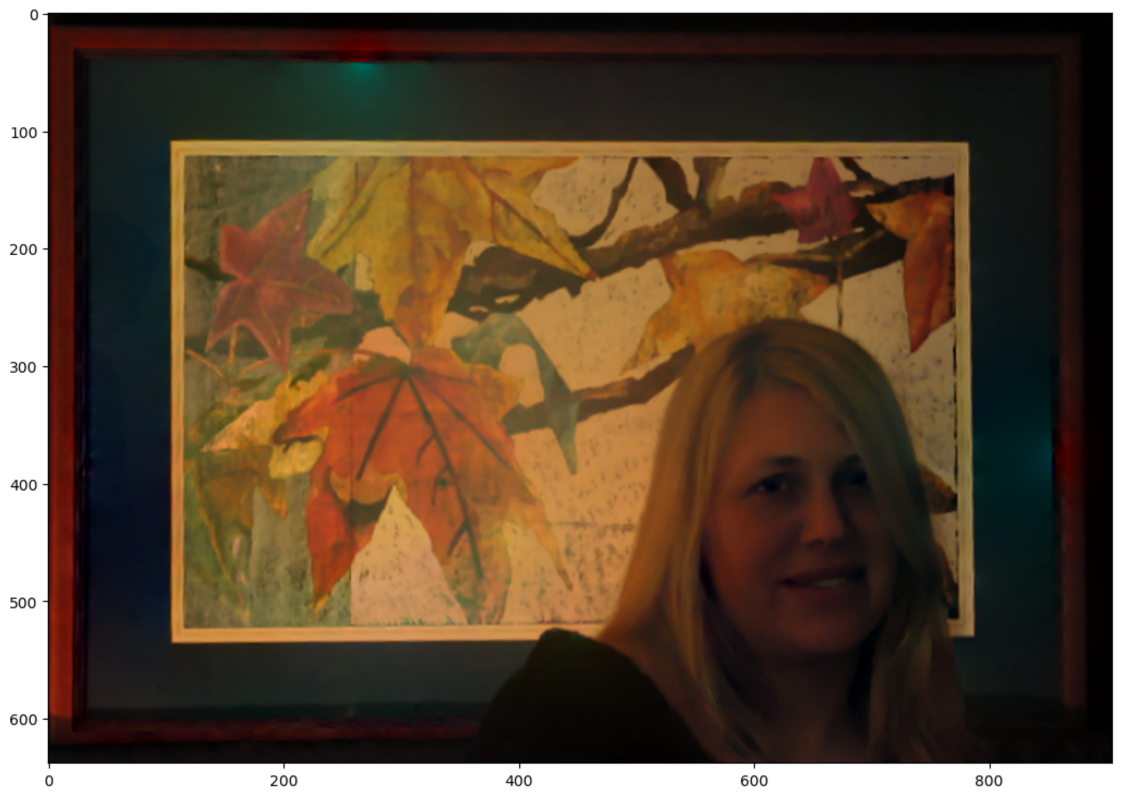
```
tau = 0.7
fused_image_r_div = fuse_gradient_field(museum_amb_norm[:,:,0],␣
 ↪museum_flash_norm[:,:,0], sigma, tau)
fused_image_g_div = fuse_gradient_field(museum_amb_norm[:,:,1],␣
 ↪museum_flash_norm[:,:,1], sigma, tau)
fused_image_b_div = fuse_gradient_field(museum_amb_norm[:,:,2],␣
 ↪museum_flash_norm[:,:,2], sigma, tau)

I_star_r = poisson_solver(fused_image_r_div, museum_amb_norm[:,:,0], N=5000,␣
 ↪eps=0.0001)
I_star_g = poisson_solver(fused_image_g_div, museum_amb_norm[:,:,1], N=5000,␣
 ↪eps=0.0001)
I_star_b = poisson_solver(fused_image_b_div, museum_amb_norm[:,:,2], N=5000,␣
 ↪eps=0.0001)

I_star = np.concatenate((np.expand_dims(I_star_r,2),np.
 ↪stack((I_star_g,I_star_b),axis=-1)),axis=2)

fig, ax = plt.subplots(2, figsize = (20,20))

ax[0].imshow(museum_amb_norm)
ax[1].imshow(I_star)

plt.show()
```

```
0 22.2254092831278
200 0.12270651688354539
400 0.10631794992179094
600 0.03653641182678293
800 0.015282583806361586
1000 0.005541985751994589
1200 0.001544945398920914
1400 0.00021016430601727298
0 15.58808710040839
200 0.10334098486855126
400 0.07923207086238547
600 0.026450632438439727
800 0.011515362061476098
1000 0.004136915146855653
1200 0.0013647960453020457
1400 0.00024004845182718466
0 11.611914397060135
200 0.06346618294283832
400 0.04498522244146792
600 0.017759121943766203
800 0.006093216765906235
1000 0.0020492192482132214
```

```
1200 0.0008824832813596229
1400 0.0001425353718196643
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

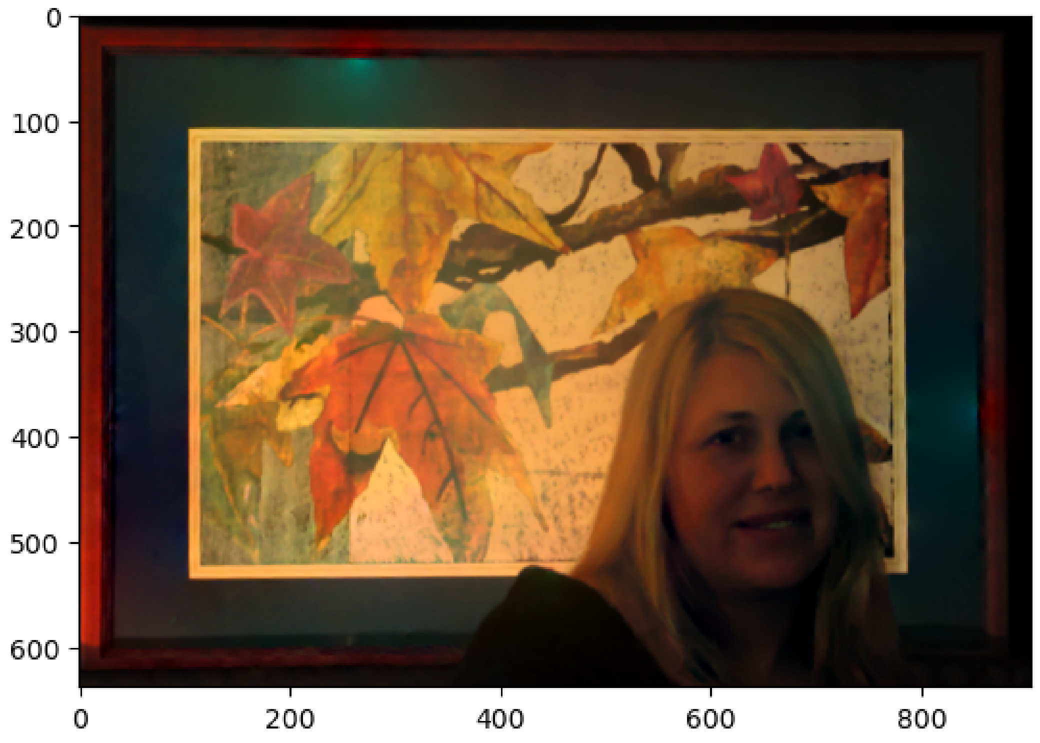

0
100
200
300
400
500
600
0
200
400
600
800


0
100
200
300
400
500
600
0
200
400
600
800

[39]:
```
plt.imsave('outputs/outputs_report_part2/boundary_condition_variation/
  ↪flashremoved_sigma{}_tau{}_boundary_ambflashmean.png'.format(sigma,tau),np.
  ↪clip(I_star,0,1))
plt.imsave('outputs/outputs_report_part2/boundary_condition_variation/
  ↪flashremoved_sigma{}_tau{}_scaled125percent_boundary_ambflashmean.png'.
  ↪format(sigma,tau),np.clip(I_star*1.25,0,1))
```

[190]:
```
plt.imshow(np.clip(I_star*1.25,0,1))
```

[190]:
```
<matplotlib.image.AxesImage at 0x7f426cc4e200>
```



[145]:
```
museum_amb_norm_gradient_x.shape
```

[145]:
```
(640, 906)
```

[146]:
```
museum_amb_norm_gradient_y.shape
```

[146]:
```
(639, 907)
```

[ ]:
```

```