Bilateral Filtering-custom-image

October 26, 2022

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import cv2
from scipy.interpolate import interpn
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}"
```

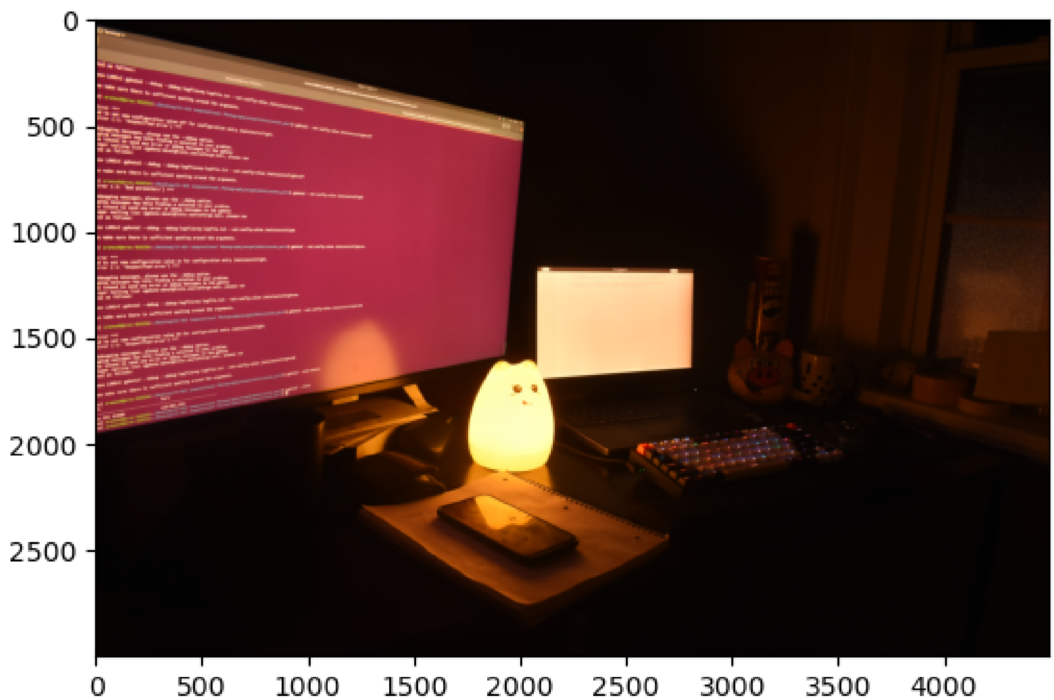
[2]:
```python
# load images
lamp_amb = './data/custom_part1/DSC_1080.JPG'
lamp_flash = './data/custom_part1/DSC_1081.JPG'

lamp_amb = cv2.imread(lamp_amb,-1)[:,:,::-1]
lamp_flash = cv2.imread(lamp_flash,-1)[:,:,::-1]

# normalize to 0-1
lamp_amb_norm = (lamp_amb - np.min(lamp_amb)) / (np.max(lamp_amb) - np.
 ↪min(lamp_amb))
lamp_flash_norm = (lamp_flash - np.min(lamp_flash)) / (np.max(lamp_flash) - np.
 ↪min(lamp_flash))
```

[3]:
```python
def bilateral_filtering(img1, img2, i_j, sigma_r, sigma_s, kernel_size=[0,0]) :

    J_j_list = []

    for j in tqdm(range(NB_segments+1)):
        G_j = (1/(sigma_r * np.sqrt(2 * np.pi))) * np.exp((-1/(2 * sigma_r**2))␣
 ↪* (img2-i_j[j])**2 )
        K_j = cv2.GaussianBlur(G_j, kernel_size, sigmaX=sigma_s,␣
 ↪sigmaY=sigma_s) # normalization factor.
        H_j = np.multiply(G_j,img1)
        H_star_j = cv2.GaussianBlur(H_j,kernel_size, sigmaX=sigma_s,␣
 ↪sigmaY=sigma_s)
        J_j = H_star_j / K_j
```

```
        J_j[K_j == 0] = 1 # to tackle cases where den is zero.
        J_j_list.append(J_j)

    J_j_list = np.array(J_j_list)
    points_rows =  np.arange(img1.shape[0])
    points_cols =  np.arange(img1.shape[1])
    pts_needed = []
    for i in range(img1.shape[0]) :
        for j in range(img1.shape[1]) :
            pts_needed.append([img1[i,j], i, j])

    J = interpn([i_j, points_rows, points_cols], J_j_list, pts_needed)
    J = J.reshape(img1.shape[0], img1.shape[1])

    return J
```

[6]:
```
# setting hyper params for bilateral filtering
lmda = 0.01
sigma_r = 0.05
sigma_s = 40
minI = np.min(lamp_amb_norm) - lmda
maxI = np.max(lamp_amb_norm) + lmda
NB_segments = np.ceil((maxI - minI)/sigma_r).astype('int')
i_j = [minI + j * (maxI - minI)/NB_segments for j in range(NB_segments+1)]
kernel_size = [11,11]

J_r = bilateral_filtering(lamp_amb_norm[:,:,0], lamp_amb_norm[:,:,0], i_j,␣
 ↪sigma_r, sigma_s,kernel_size)
J_g = bilateral_filtering(lamp_amb_norm[:,:,1], lamp_amb_norm[:,:,1], i_j,␣
 ↪sigma_r, sigma_s,kernel_size)
J_b = bilateral_filtering(lamp_amb_norm[:,:,2], lamp_amb_norm[:,:,2], i_j,␣
 ↪sigma_r, sigma_s,kernel_size)
J = np.concatenate((np.expand_dims(J_r,-1),np.stack((J_g,J_b),-1)),-1)
plt.imsave('outputs/outputs_report_part3/custom_part1_J_sr_{}_ss_{}_ks_{}_Abase.
 ↪png'.format(sigma_r, sigma_s, kernel_size[0]), np.clip(J, 0,1))
plt.imshow(np.clip(J, 0,1))

A_base = J.copy()
```

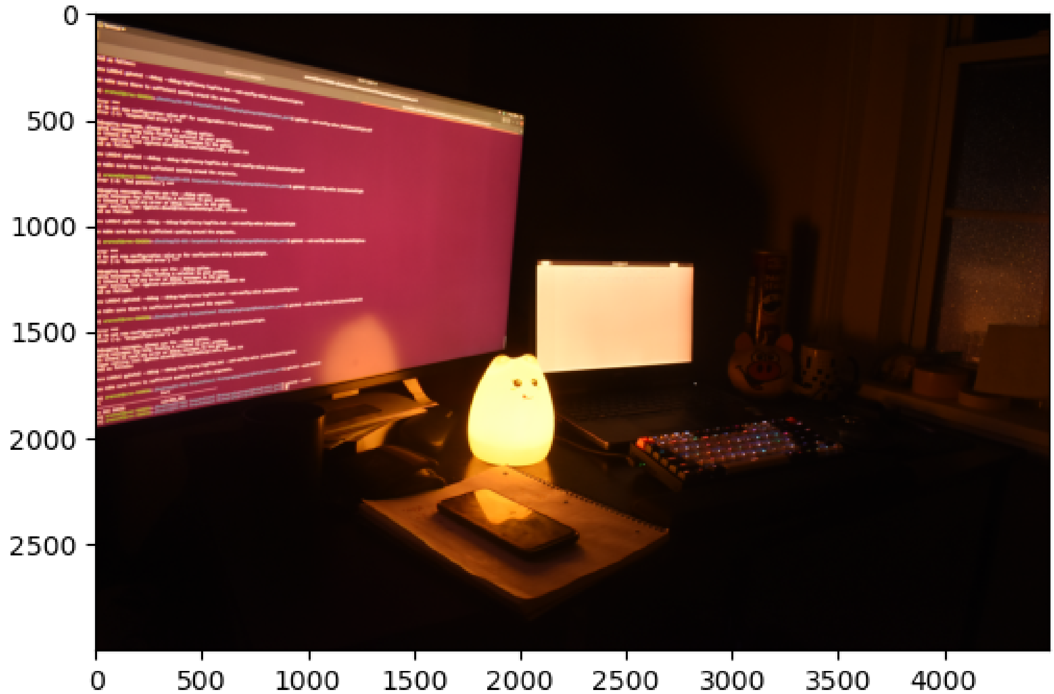
```
100%|                        | 22/22 [00:09<00:00,  2.24it/s]
100%|                        | 22/22 [00:08<00:00,  2.68it/s]
100%|                        | 22/22 [00:08<00:00,  2.69it/s]
```



[7]:
```python
lmda = 0.01
sigma_r = 0.05
sigma_s = 100
minI = np.min(lamp_flash_norm) - lmda
maxI = np.max(lamp_flash_norm) + lmda
NB_segments = np.ceil((maxI - minI)/sigma_r).astype('int')
i_j = [minI + j * (maxI - minI)/(NB_segments) for j in range(NB_segments+1)]
kernel_size = [5,5]

J_r = bilateral_filtering(lamp_amb_norm[:,:,0], lamp_flash_norm[:,:,0], i_j,␣
 ↪sigma_r, sigma_s,kernel_size)
J_g = bilateral_filtering(lamp_amb_norm[:,:,1], lamp_flash_norm[:,:,1], i_j,␣
 ↪sigma_r, sigma_s,kernel_size)
J_b = bilateral_filtering(lamp_amb_norm[:,:,2], lamp_flash_norm[:,:,2], i_j,␣
 ↪sigma_r, sigma_s,kernel_size)
J = np.concatenate((np.expand_dims(J_r,-1),np.stack((J_g,J_b),-1)),-1)
plt.imsave('outputs/outputs_report_part3/custom_part1_J_sr_{}_ss_{}_ks_{}_ANr.
 ↪png'.format(sigma_r, sigma_s, kernel_size[0]), np.clip(J, 0,1))
plt.imshow(np.clip(J, 0,1))

A_NR = J.copy()
```

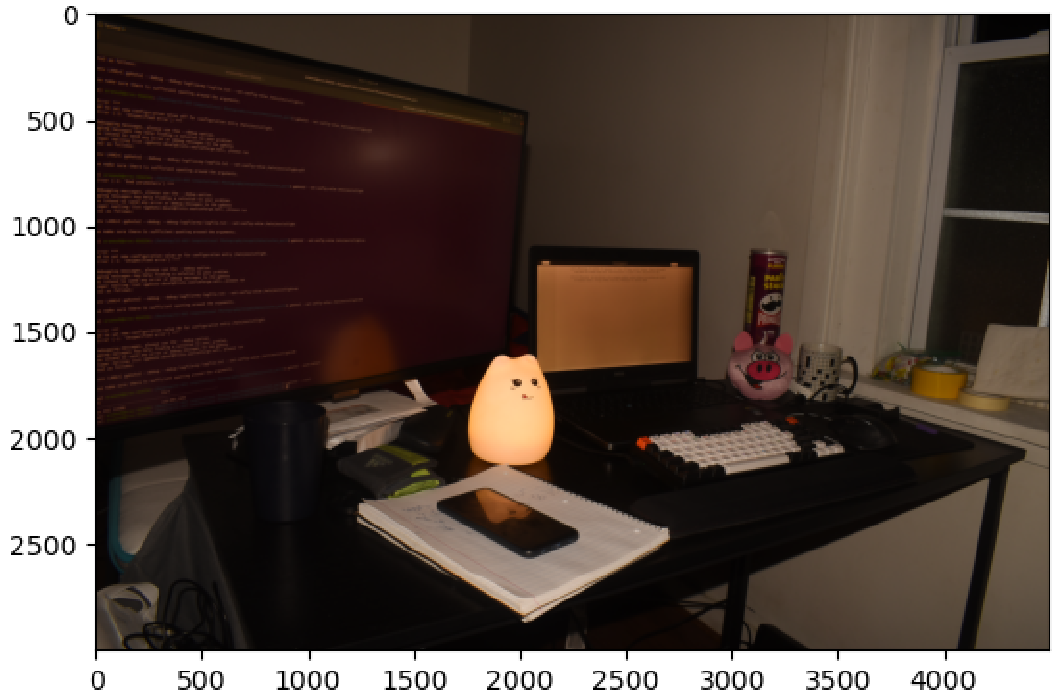
```
100%|                              | 22/22 [00:06<00:00,  3.41it/s]
100%|                              | 22/22 [00:08<00:00,  2.66it/s]
```

```
100%|                    | 22/22 [00:08<00:00,  2.71it/s]
```



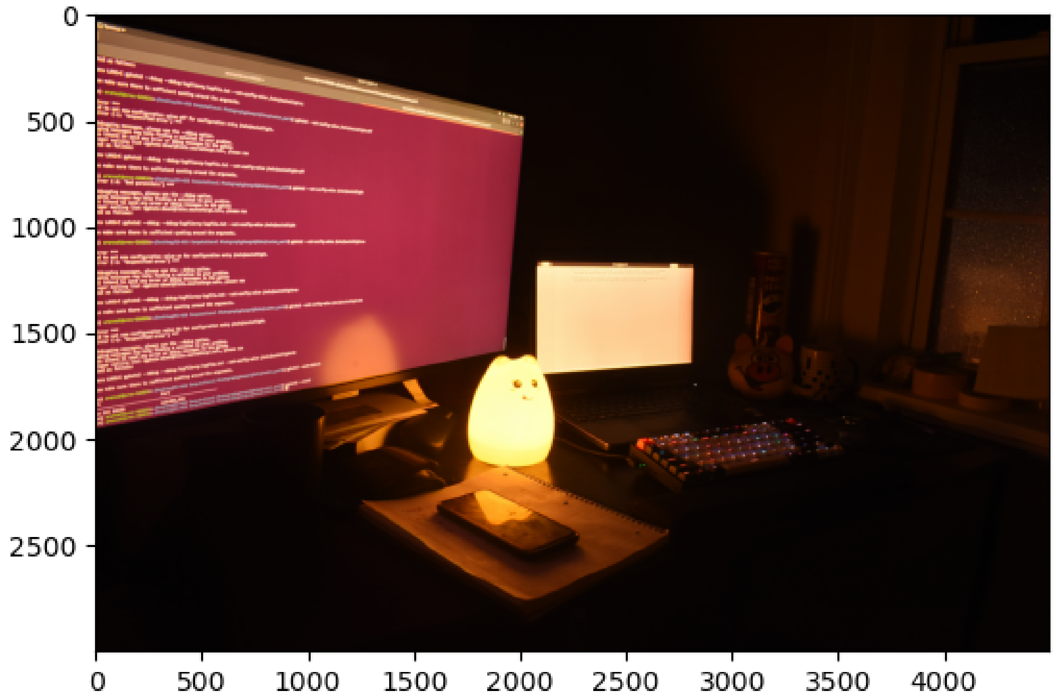
[8]:
```python
# Calculate Fbase
lmda = 0.01
sigma_r = 0.05
sigma_s = 40
minI = np.min(lamp_flash_norm) - lmda
maxI = np.max(lamp_flash_norm) + lmda
NB_segments = np.ceil((maxI - minI)/sigma_r).astype('int')
i_j = [minI + j * (maxI - minI)/NB_segments for j in range(NB_segments+1)]
kernel_size = [11,11]

J_r = bilateral_filtering(lamp_flash_norm[:,:,0], lamp_flash_norm[:,:,0], i_j,␣
 ↪sigma_r, sigma_s,kernel_size)
J_g = bilateral_filtering(lamp_flash_norm[:,:,1], lamp_flash_norm[:,:,1], i_j,␣
 ↪sigma_r, sigma_s,kernel_size)
J_b = bilateral_filtering(lamp_flash_norm[:,:,2], lamp_flash_norm[:,:,2], i_j,␣
 ↪sigma_r, sigma_s,kernel_size)
J = np.concatenate((np.expand_dims(J_r,-1),np.stack((J_g,J_b),-1)),-1)
plt.imsave('outputs/outputs_report_part3/custom_part1_J_sr_{}_ss_{}_ks_{}_Fbase.
 ↪png'.format(sigma_r, sigma_s, kernel_size[0]), np.clip(J, 0,1))
plt.imshow(np.clip(J, 0,1))

F_base = J.copy()
```

```
100%|                              | 22/22 [00:10<00:00,  2.08it/s]
100%|                              | 22/22 [00:10<00:00,  2.09it/s]
100%|                              | 22/22 [00:10<00:00,  2.19it/s]
```



[9]:
```python
# Apply detail transfer
eps = 0.001
A_detail = A_NR * (lamp_flash_norm + eps) / (F_base + eps)
plt.imsave('outputs/outputs_report_part3/
↪custom_part1_J_sr_{}_ss_{}_ks_{}_Adetail.png'.format(sigma_r, sigma_s,␣
↪kernel_size[0]), np.clip(A_detail, 0,1))
plt.imshow(np.clip(A_detail, 0,1))
```

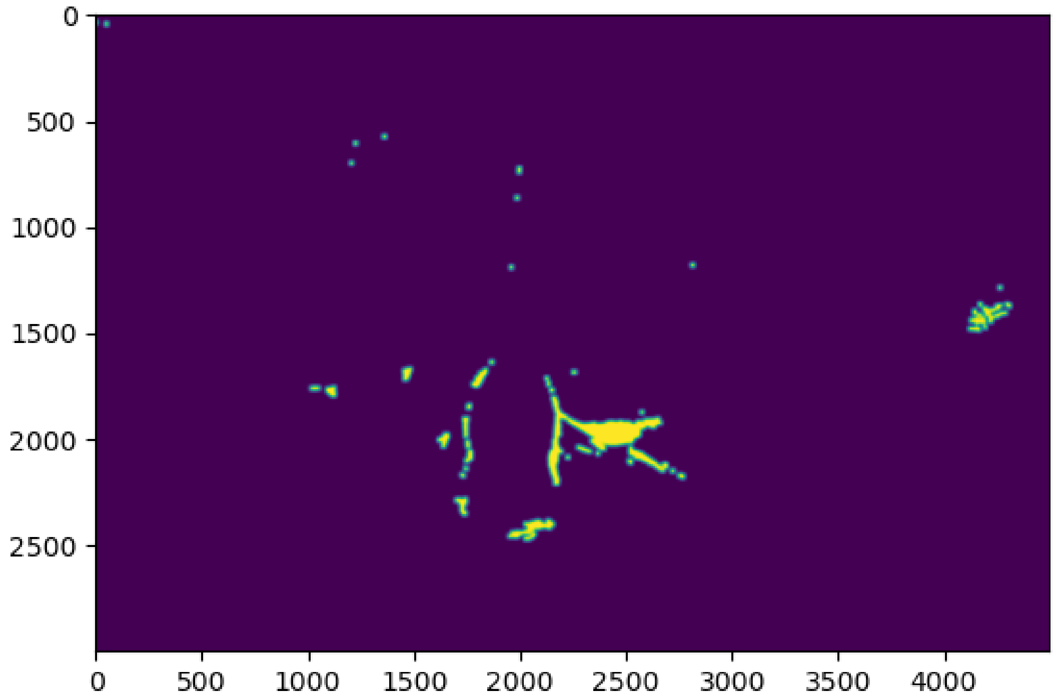
[9]:
```
<matplotlib.image.AxesImage at 0x7f2f7f18a740>
```



[10]:
```
# Computing mask for shadows and specularities

# linearizing images

@np.vectorize
def linearize_image(C_nonlinear) :

    if C_nonlinear <= 0.0404482 :
        return C_nonlinear / 12.92
    else :
        out_num = ( C_nonlinear + 0.055 ) ** 2.4
        out_den = 1.055 ** 2.4

        return out_num / out_den

lamp_amb_norm_lin = linearize_image(lamp_amb_norm)
lamp_flash_norm_lin = linearize_image(lamp_flash_norm)

#iso correction
# flash image iso-100
# amb image iso-800
# shutter speed 1/2 s for both
lamp_amb_norm_lin = lamp_amb_norm_lin * (100/800)
```

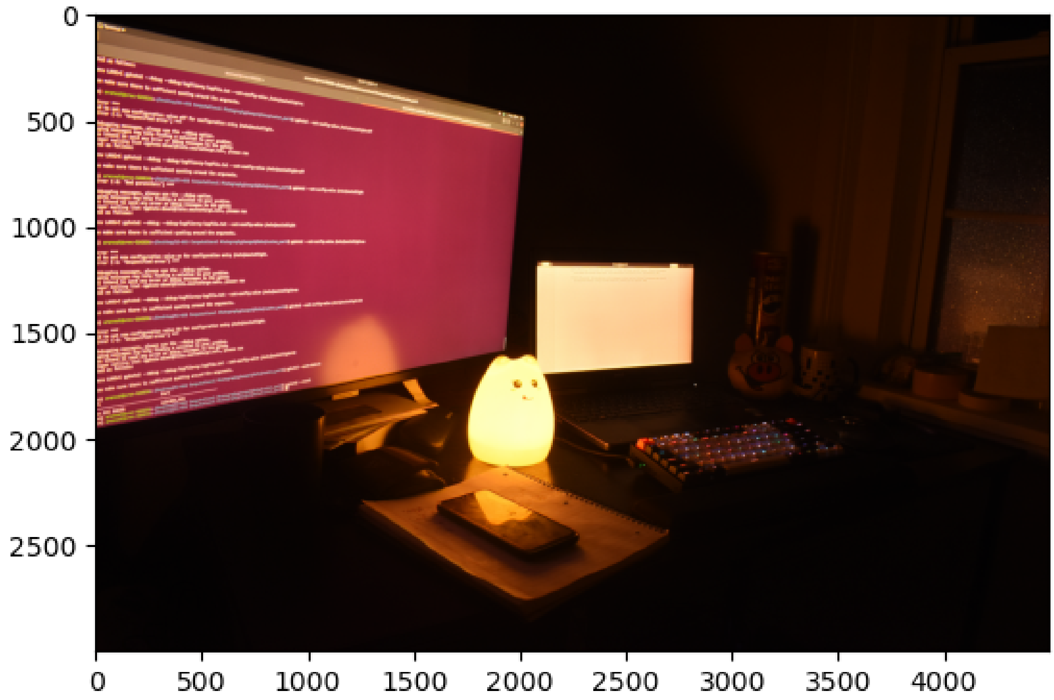
```
shadow_threshold = 0.0005
speckle_thershold = 0.9

luminance_amb = cv2.cvtColor(lamp_amb_norm_lin[:,:,::-1].astype('float32'), cv2.COLOR_BGR2YCR_CB)[:,:,0]
luminance_flash = cv2.cvtColor(lamp_flash_norm_lin[:,:,::-1].astype('float32'), cv2.COLOR_BGR2YCR_CB)[:,:,0]

shadow_map = np.zeros(luminance_amb.shape)
shadow_map[np.abs(luminance_flash - luminance_amb) <= shadow_threshold] = 1

speckle_map = np.zeros(luminance_amb.shape)
speckle_map[luminance_flash > speckle_thershold] = 1

opening_kernel = np.ones((3,3),np.uint8) #clears noise
closing_kernel = np.ones((8,8),np.uint8) #fills holes
dilation_kernel = np.ones((20,20),np.uint8) # dilation

shadow_map_opened = cv2.morphologyEx(shadow_map, cv2.MORPH_OPEN, opening_kernel)
shadow_map_closed = cv2.morphologyEx(shadow_map_opened, cv2.MORPH_CLOSE, closing_kernel)
shadow_map_dilated = cv2.dilate(shadow_map_closed,dilation_kernel)

speckle_map_opened = cv2.morphologyEx(speckle_map, cv2.MORPH_OPEN, opening_kernel)
speckle_map_closed = cv2.morphologyEx(speckle_map_opened, cv2.MORPH_CLOSE, closing_kernel)
speckle_map_dilated = cv2.dilate(speckle_map_closed,dilation_kernel)

final_map = shadow_map_dilated.copy()
final_map[speckle_map_dilated==1] = 1

final_map_blurred = cv2.GaussianBlur(final_map,[21,21],75)
plt.imshow(final_map_blurred)
```

[10]: <matplotlib.image.AxesImage at 0x7f2f7f1bd060>




```
[11]: # Mask assisted merge
final_map_blurred_3channel = np.repeat(np.expand_dims(final_map_blurred,2),3,␣
 ↪axis=2)
A_final = (1-final_map_blurred_3channel) * A_detail +␣
 ↪final_map_blurred_3channel * A_base
plt.imsave('outputs/outputs_report_part3/
 ↪custom_part1_J_sr_{}_ss_{}_ks_{}_Afinal.png'.format(sigma_r, sigma_s,␣
 ↪kernel_size[0]), np.clip(A_detail, 0,1))
plt.imshow(np.clip(A_detail, 0,1))
```

```
[11]: <matplotlib.image.AxesImage at 0x7f2f7f20ad10>
```



[13]:

```
```

[13]: (2048, 3072, 3)

[ ]:

```
```