CSC2529 HW4

Martin Pham

October 27, 2022

Code available at https://github.com/mdpham/CSC2529/tree/main/homework4

1 HDR image fusion

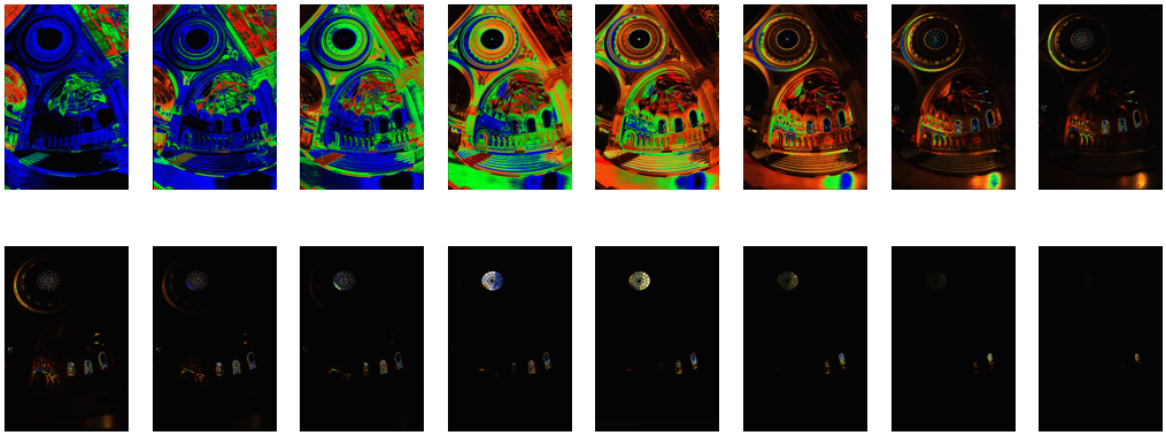


Figure 1: Debevec weight maps for LDR images

2 Burst Denoising and SNR

Consider a burst denoised image as the average of K images $I_i = \tilde{I}_i + n_i$ where n_i is some independent additive noise for $i = 1...K$.

In the case of heat noise modelled by a Gaussian, define $\mu_g = \sum_{i=1}^{K} \mu_i$ and $\sigma_g = \sqrt{\sum_{i=1}^{K} \sigma_i^2}$ for $n_i \sim G(\mu_i, \sigma_i^2)$. By properties of Gaussian distributions, the sum has distribution:

$$\sum_{i=1}^{K} n_i \sim G(\mu_g, \sigma_g^2)$$

To obtain the average distribution of noise, divide random variables by K to get distribution of additive noise in burst image

$$N_{heat} := K^{-1} \sum_{i=1}^{K} n_i \sim G(K^{-1}\mu_g, K^{-2}\sigma_g^2)$$

For $\mu_i = \mu$ and $\sigma_i = \sigma$, the SNR is thus

$$SNR = \frac{K^{-1}\mu_g}{\sqrt{K^{-2}\sigma_g^2}} = \frac{\sum_{i=1}^{K} \mu_i}{\sqrt{\sum_{i=1}^{K} \sigma_i^2}} = \frac{K\mu}{\sqrt{K}\sigma} = \frac{\sqrt{K}\mu}{\sigma}$$

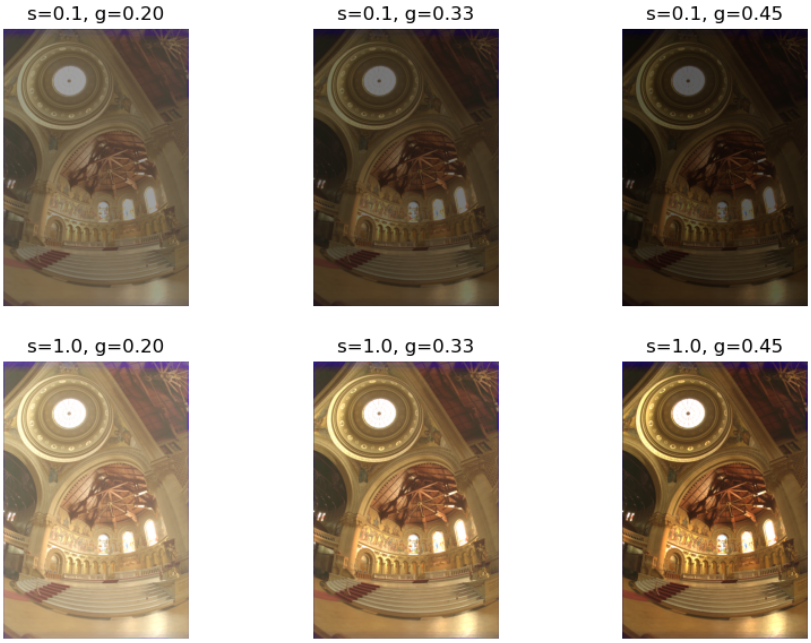


Figure 2: Scale ($s = \{0.1, 1.0\}$) and gamma ($g = \{1/5, 1/3, 1/2.2\}$) based tonemapping

Figure 3: Drago based tonemapping from OpenCV

In the case of shot noise modelled by a Poisson, consider $n_i \sim P(\lambda_i)$. By properties of Poisson distributions, the sum has distribution:

$$\sum_{i=1}^{K} n_i \sim P(\sum_{i=1}^{K} \lambda_i)$$

Define $\lambda_p = \sum_{i=1}^{K} \lambda_i$. Dividing the sum by K produces a non-Poisson distribution but we apply expectation/variance properties

$$N_{shot} := K^{-1} \sum_{i=1}^{K} n_i$$

where

$$E[\sum_{i=1}^{K} n_i] = \lambda_p = Var[\sum_{i=1}^{K} n_i]$$

so

$$E[K^{-1}\sum_{i=1}^{K} n_i] = K^{-1}E[\sum_{i=1}^{K} n_i] = K^{-1}\lambda_p$$

and

$$Var[K^{-1}\sum_{i=1}^{K} n_i] = K^{-2}Var[\sum_{i=1}^{K} n_i] = K^{-2}\lambda_p$$

For $\lambda_i = \lambda$, the SNR is thus

$$SNR = \frac{K^{-1}\lambda_p}{\sqrt{K^{-2}\lambda_p}} = \frac{K^{-1}\sum_{i=1}^{K}\lambda_i}{\sqrt{K^{-2}\sum_{i=1}^{K}\lambda_i}} = \frac{K^{-1}K\lambda}{K^{-1}\sqrt{K\lambda}} = \sqrt{K\lambda}$$

3 Flutter Shutter and SNR

Consider consumer camera ($n = 10000$) with flutter shutter, so $\mu = \frac{n}{2} = 5000$ and $\sigma = 50$.

$$SNR = \frac{\mu}{\sigma} = \frac{5000}{50} = 100$$

Consider consumer camera ($n = 10000$) with burst, so $\mu = \frac{n}{100} = 100$ and $\sigma = 50$. We can use the SNR formula from previous task to compute for case of Gaussian noise.

$$SNR = \frac{\sqrt{K}\mu}{\sigma} = \frac{\sqrt{100}100}{50} = 20$$

Consider scientific camera ($n = 1000$) with flutter shutter, so $\lambda = \sigma^2 = \mu = \frac{n}{2} = 500$.

$$SNR = \frac{\mu}{\sigma} = \frac{500}{\sqrt{500}} = \sqrt{500} \approx 22.36$$

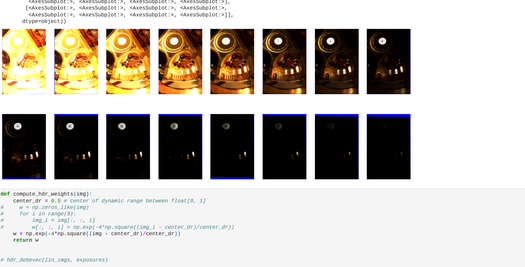
Consider scientific camera ($n = 1000$) with burst, so $\lambda = \sigma^2 = \mu = \frac{n}{100} = 10$. We can use the SNR formula from previous task to compute for case of Poisson noise.

$$SNR = \sqrt{K\lambda} = \sqrt{100}\sqrt{10} \approx 31.62$$

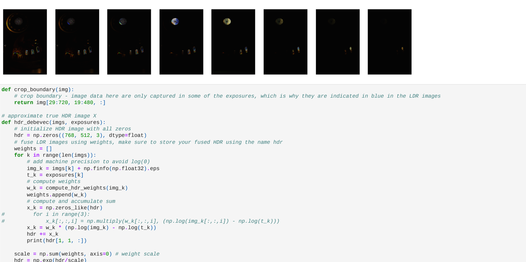
	SNR	
Camera	Flutter shutter	Burst
Consumer	100	20
sCMOS ($n = 1000$)	22	31

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import imageio.io as io
import imageio
import cv2

from pdb import set_trace
from pathlib import Path
```

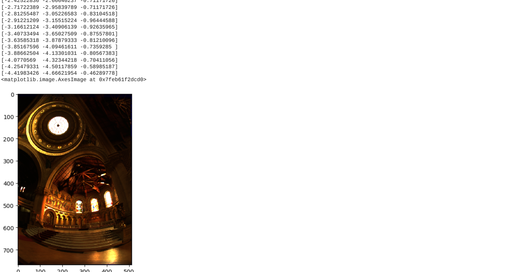
In [2]:
```
hdr_dir = 'hdr_data'

def read_exposures():
    filepaths = []
    exposures = []
    with open(Path(hdr_dir, 'memorial.hdr_image_list.txt')) as f:
        lines = f.readlines()
    for i in lines[3:]:
        data = i.split(' ')
        img_png = data[0].replace('.ppm', '.png')
        expo = 1/float(data[1]) *read value is 1/shutter speed, so inverse
        filepaths.append(Path(hdr_dir, img_png))
        exposures.append(expo)
    return filepaths, exposures

filepaths, exposures = read_exposures()
print(exposures)

def read_img(filepath):
    img = io.imread(filepath).astype(float)/255
    return img

def plot_images(imgs):
    fig, axes = plt.subplots(2, 8, figsize=(13, 5))
    ax = axes.ravel()
    for i in range(len(imgs)):
        ax[i].imshow(imgs[i])
        ax[i].axis('off')
    return fig, axes

imgs = [read_img(fp) for fp in filepaths]
# img[0].shape
plot_images(imgs)
```

Out[2]:
```
[32.0, 16.0, 8.0, 4.0, 2.0, 1.0, 0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625, 0.0078125, 0.00390625, 0.001953125, 0.0009765625]
(<Figure size 1300x500 with 16 Axes>,
 array([[<Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>,
         <Axes:>],
        [<Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>,
         <Axes:>]], dtype=object))
```


In [3]:
```
def linearize_img(img, gamma=2.2):
    return np.power(img, 2.2)

lin_imgs = list(map(lambda x: linearize_img(x, 2.2), imgs))
plot_images(lin_imgs)
```

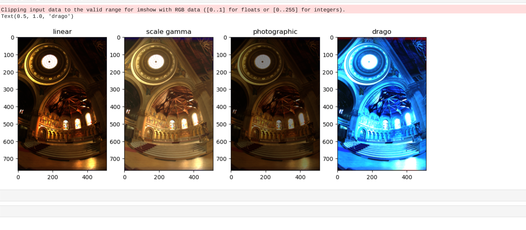
Out[3]:
```
(<Figure size 1300x500 with 16 Axes>,
 array([[<Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>,
         <Axes:>],
        [<Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>,
         <Axes:>]], dtype=object))
```


In [4]:
```
def compute_hdr_weights(img):
    center_dr = 0.5 # center of dynamic range between float[0, 1]
#     w = np.zeros_like(img)
#     for i in range(3):
#         img_i = img[:, :, i]
#         w[:, :, i] = np.exp(-4*np.square((img_i - center_dr)/center_dr))
    w = np.exp(-4*np.square((img - center_dr)/center_dr))
    return w


# hdr_debevec(lin_imgs, exposures)

weight_maps = list(map(compute_hdr_weights, lin_imgs))
plot_images(weight_maps)
```

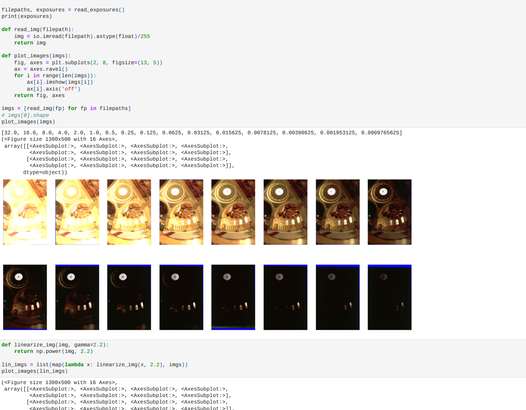
Out[4]:
```
(<Figure size 1300x500 with 16 Axes>,
 array([[<Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>,
         <Axes:>],
        [<Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>, <Axes:>,
         <Axes:>]], dtype=object))
```


In [5]:
```
def crop_boundary(img):
    # crop boundary - image data here are only captured in some of the exposures, which is why they are indicated in blue in the LDR images
    return img[29:729, 19:480, :]

# approximate true HDR image X
def hdr_debevec(imgs, exposures):
    # initialize HDR image with all zeros
    hdr = np.zeros((768, 512, 3), dtype=float)
    # fuse LDR images using weights, make sure to store your fused HDR using the name hdr
    weights = []
    for k in range(len(imgs)):
        # add machine precision to avoid log(0)
        img_k = imgs[k] + np.finfo(np.float32).eps
        # compute weights
        # w_k = compute_hdr_weights(img_k)
        weights.append(w_k)
#         # compute and accumulate sum
#         x_k = np.zeros_like(hdr)
#         for i in range(3):
#             x_k[:,:,i] = np.multiply(w_k[:,:,i], (np.log(img_k[:,:,i]) - np.log(t_k)))
        x_k = w_k * (np.log(img_k) - np.log(t_k))
        hdr += x_k
        print(hdr[1, 1, :])

    scale = np.sum(weights, axis=0) # weight scale
    hdr = np.exp(hdr/scale)

    # crop and clip
#     hdr = crop_boundary(hdr)
    hdr = np.clip(hdr, 0, 1)

    return hdr

hdr_lin_img = hdr_debevec(lin_imgs, exposures)
test = hdr_lin_img
# test = np.uint8( np.clip(3*test, 0, 1) *255)
# test[:, :, 1] = 0
# test[:, :, 0] = 0
io.imshow(test)
```


```
# np.max(hdr_lin_img)
```

```
[-0.12346241 -1.54024489 -0.35715922]
[-0.46626977 -1.89202145 -0.40794085]
[-1.55428564 -2.23487461 -0.57130085]
[-2.25168229 -2.55226135 -0.59669985]
[-2.42523836 -2.66648237 -0.71717726]
[-2.71722389 -2.95839789 -0.71717726]
[-2.81295497 -3.05226583 -0.83104518]
[-2.91221209 -3.15515224 -0.96444588]
[-3.16612124 -3.40906139 -0.92639965]
[-3.40733494 -3.65027509 -0.87557801]
[-3.63585518 -3.87879533 -0.81210996]
[-3.85167596 -4.09461611 -0.7359285 ]
[-3.86662504 -4.11391831 -0.80567383]
[-4.0770569  -4.32344218 -0.76411056]
[-4.25479331 -4.50117859 -0.58985187]
[-4.41983426 -4.66621954 -0.4628778 ]
<matplotlib.image.AxesImage at 0x7f4b6f20cdd0>
```


In [12]:
```
def clip_img(img):
    return np.clip(img, 0, 1)

def tonemap_linear(img):
    toned_img = np.zeros_like(img)
    for i in range(3):
        img_channel = img[:, :, i]
        max_value = np.max(img_channel)
        toned_img[:,:,i] = img_channel/max_value
    return toned_img

def tonemap_scale_gamma(scale, gamma, img):
    toned_img = np.power(scale*img, gamma)
    return toned_img

def tonemap_photograph(hdr):
    img = np.divide(hdr, 1+hdr)
    return img

fig, axes = plt.subplots(2, 3, figsize=(10, 7))
ax = axes.ravel()
sg_hdr_imgs = {}
for s in [0.1, 1.0]:
    for g in [1/5, 1/3, 1/2.2]:
        sg_hdr_imgs[f"s={s}, g={1/g:.2f}"] = tonemap_scale_gamma(s, g, hdr_lin_img)

sg = list(sg_hdr_imgs.keys())
print(sg)
for i in range(len(sg)):
    key = sg[i]
    sg_img = sg_hdr_imgs[key]
    ax[i].axis('off')
    ax[i].set_title(key)
    ax[i].imshow(sg_img)
```

```
['s=0.1, g=0.20', 's=0.1, g=0.33', 's=0.1, g=0.45', 's=1.0, g=0.20', 's=1.0, g=0.33', 's=1.0, g=0.45']
```


In [18]:
```
def tonemap_drago(hdr):
    hdr *= 0.6371096/np.mean(hdr)  # this makes the mean of the created HDR image match the reference image (totally optional)

    # convert to 32-bit floating point format, required for OpenCV
    hdr = np.float32(hdr)

    # tonemap image and save LDR image using OpenCV's implementation of Drago's tonemapping operator
    gamma = 1.0
    saturation = 0.7
    bias = 0.85
    tonemapDrago = cv2.createTonemapDrago(gamma,saturation,bias)
    ldrDrago = tonemapDrago.process(hdr)
    io.imsave('my_hdr_image_tonemapped.jpg', np.uint8( np.clip(3*ldrDrago, 0, 1) *255))

    # write HDR image (can compare to hw4_1_memorial_church.hdr reference image in an external viewer)
    hdr = cv2.cvtColor(hdr, cv2.COLOR_BGR2RGB)

    cv2.imwrite('my_hdr_image.hdr', hdr)

    return hdr
```

In [19]:
```
hdr_lin_clip = clip_img(hdr_lin_img)
fig, axes = plt.subplots(1, 4, figsize=(13, 5))
axes[0].imshow(tonemap_linear(hdr_lin_clip))
axes[0].set_title("linear")
axes[1].imshow(tonemap_scale_gamma(1, 1/2.2, hdr_lin_clip))
axes[1].set_title("scale gamma")
axes[2].imshow(tonemap_photograph(hdr_lin_clip))
axes[2].set_title("photographic")
axes[3].imshow(tonemap_drago(clip_img(hdr_lin_clip)))
axes[3].set_title("drago") # doesn't render in notebook well but .hdr file saves fine for viewing
```

Out[19]:
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Text(0.5, 1.0, 'drago')
```


In [ ]:

In [ ]: