High Dynamic Range Imaging

In this project, we will learn how to create a High Dynamic Range (HDR) image using multiple images taken with different exposure settings.Most of the contents are borrowed from this tutorial. And there is another good tutorial with Matlab code you may like to read. Matlab also provides a web based HDR image renderer.

What is High Dynamic Range (HDR) imaging?

Most digital cameras and displays capture or display color images as 24-bits matrices. There are 8-bits per color channel and the pixel values are therefore in the range 0–255 for each channel. In other words, a regular camera or a display has a limited dynamic range.

However, the world around us has a very large dynamic range. It can get pitch black inside a garage when the lights are turned off and it can get really bright if you are looking directly at the Sun. Even without considering those extremes, in everyday situations, 8-bits are barely enough to capture the scene. So, the camera tries to estimate the lighting and automatically sets the exposure so that the most interesting aspect of the image has good dynamic range, and the parts that are too dark and too bright are clipped off to 0 and 255 respectively.

In the Figure below, the image on the left is a normally exposed image. Notice the sky in the background is completely washed out because the camera decided to use a setting where the subject (a boy) is properly photographed, but the bright sky is washed out. The image on the right is an HDR image produced by the iPhone.

How does an iPhone capture an HDR image? It actually takes 3 images at three different exposures. The images are taken in quick succession so there is almost no movement between the three shots. The three images are then combined to produce the HDR image. We will see the details in the next section.

> The process of combining different images of the same scene acquired under different exposure settings is called High Dynamic Range (HDR) imaging.

To summarize, 传统图像具有256级像素亮度范围，而High dynamic range (HDR，高动态范围)have much larger dynamic range than traditional images' 256 brightness levels. In addition, they correspond linearly 线性对应 to physical irradiance values of the scene 场景中物理辐照度. Hence, they have many applications in graphics and vision. In this project, you are expected to finish the following tasks to assemble an HDR image.

Step 1: Capture multiple images with different exposures

Taking images. Taking a series of photographs for a scene under different exposures 曝光. As discussed in the class, changing shutter speed 快门速度 is probably the best way to change exposure for this application. For that, you need a digital camera that allows you to set exposures. (Note that not every camera allows a user to manually set exposures.) You can use your own camera on your cellphone. 我的华为手机摄像程序里有"专业"模式，可以设置快门速度，如下图所示，标黄的那里就是修改快门的地方。

One thing to note is that you should avoid moving your camera during this process so that all pictures are well registered.要保持相机静止，否则发生图像像素的错位。 Some digital cameras have their own programs which allow users to remotely control the shutters via their USB cables.单反相机有线控快门或者usb连到电脑上通过软件控制拍摄。 Using such programs prevent you from shaking the camera while pressing the shutter. You are welcome to write down your findings for that matter in your report.

If you don't want to capture your own images, you can use those in the "data" directory (AI Studio). Unzip the "exposure.zip" or "Memorial_SourceImages.zip" or "hdr.zip" and you will see a folder of a series of images. The exposure time comes with the image in a text file (the "exposure" and "Memorial_SourceImages" datasets) or is just indicated in the file name ("hdr" dataset). Debevec's images are also provided.

In [ ]:
```
# unzip the dataset provided or your own uploaded data
# !cd data/data54145/ && unzip -o exposures.zip -d exposures
!unzip -o exposures.zip -d exposures
!unzip -o Memorial_SourceImages.zip -d Memorial_SourceImages
!unzip -o hdr.zip -d hdr
```

```
'unzip' �����گ�����⧻���  X����ǿ����� e ij���
���������|���
'unzip' �����گ�����⧻���  X����ǿ����� e ij���
���������|���
'unzip' �����گ�����⧻���  X����ǿ����� e ij���
���������|���
```

Please write your own code to load the images and exposure times

In [ ]:
```
# read images and exposure times
import cv2
import numpy as np

### TODO: write your code to load the images and corresponding exposure times. Note that the exposure times are in the text file "shutter.txt".
images = [cv2.imread(x) for x in ['./hdr/img_0.033.jpg', './hdr/img_0.25.jpg', './hdr/img_2.5.jpg', './hdr/img_15.jpg']]
times = np.array([0.033, 0.25, 2.5, 15.0]).astype(np.float32)
pass
```

Step 2: Align Images

Misalignment of images used in composing the HDR image can result in severe artifacts. In the Figure below, the image on the left is an HDR image composed using unaligned images and the image on the right is one using aligned images. By zooming into a part of the image, shown using red circles, we see severe ghosting artifacts in the left image.

Naturally, while taking the pictures for creating an HDR image, professional photographer mount the camera on a tripod. They also use a feature called mirror lockup to reduce additional vibrations. Even then, the images may not be perfectly aligned because there is no way to guarantee a vibration-free environment. The problem of alignment gets a lot worse when images are taken using a handheld camera or a phone.

Fortunately, OpenCV provides an easy way to align these images using AlignMTB. This algorithm converts all the images to median threshold bitmaps (MTB). An MTB for an image is calculated by assigning the value 1 to pixels brighter than median luminance and 0 otherwise. An MTB is invariant to the exposure time. Therefore, the MTBs can be aligned without requiring us to specify the exposure time.

MTB based alignment is performed using the following lines of code.

In [ ]:
```
# Align input images
alignMTB = cv2.createAlignMTB()
alignMTB.process(images, images)
```

Step 3: Recover the Camera Response Function

Write a program to assemble 合成 an HDR image. Write a program to take these captured images as inputs and output an HDR image as well as the response curve of the camera 相机的响应曲线. You will use the `Debevec's method` . Please refer to Debevec's SIGGRAPH 1997 paper below. The most difficult part probably is to solve the over-determined linear system.

Paul E. Debevec, Jitendra Malik, Recovering High Dynamic Range Radiance Maps from Photographs, SIGGRAPH 1997.

这篇文章的方法首先要恢复相机的响应函数Recover the Camera Response Function The response of a typical camera is not linear非线性 to scene brightness. What does that mean? Suppose, two objects are photographed by a camera and one of them is twice as bright as the other in the real world. When you measure the pixel intensities of the two objects in the photograph, the pixel values of the brighter object will not be twice that of the darker object! Without estimating the Camera Response Function (CRF), we will not be able to merge the images into one HDR image.

What does it mean to merge multiple exposure images into an HDR image?通过多张不同曝光的图像如何合成一个HDR图像?

Consider just ONE pixel at some location (x,y) of the images. If the CRF was linear假设相机响应函数是线性的, the pixel value would be directly proportional to the exposure time unless the pixel is too dark (i.e. nearly 0) or too bright (i.e. nearly 255) in a particular image. We can filter out these bad pixels (too dark or too bright), and estimate the brightness at a pixel by dividing the pixel value by the exposure time and then averaging this brightness value across all images where the pixel is not bad (too dark or too bright). We can do this for all pixels and obtain a single image where all pixels are obtained by averaging "good" pixels.把"好的"像素挑出来，亮度除以曝光时间，然后多张图像取平均即可。

But the CRF is not linear and we need to make the image intensities linear before we can merge/average them by first estimating the CRF.

The good news is that the CRF can be estimated from the images if we know the exposure times for each image. Like many problems in computer vision, the problem of finding the CRF is set up as an optimization problem where the goal is to minimize an objective function consisting of a data term and a smoothness term. These problems usually reduce to a linear least squares problem which are solved using Singular Value Decomposition (SVD) that is part of all linear algebra packages. The details of the CRF recovery algorithm are in the paper titled Recovering High Dynamic Range Radiance Maps from Photographs.

Finding the CRF is done using just two lines of code in OpenCV using CalibrateDebevec or CalibrateRobertson. In this project we will use CalibrateDebevec:

In [ ]:
```
# Obtain Camera Response Function (CRF) using OpenCV functions. Note that the image array should be in descending order of exposure times.
```

```python
calibrateDebevec = cv2.createCalibrateDebevec()
responseDebevec = calibrateDebevec.process(images, times)
```

The figure below shows the CRF recovered using the images for the red, green and blue channels. Note that the calibrated intensity is after an exponential operation.

Please write your own code for recovering CRF using Debevec's algorithm. Note that the image has RGB three channels, and you need to precess each channel separately.

The most important part is to resemble the matrix for the linear system.

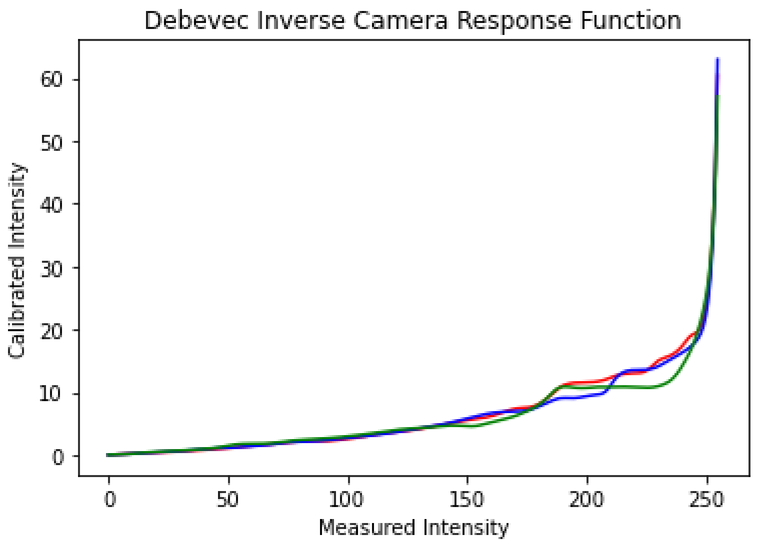
In [ ]:

```python
### TODO: implementation of Debevec's CRF recovering algorithm. You need to return the crf (descrete array) for each channel.
import math
from matplotlib import pyplot as plt
import numpy.matlib
import warnings
warnings.filterwarnings("ignore")

def gsolve(Z):
        '''
        Z(i,j) is the pixel values of pixel locations number i in image j
        B(j)   is the log delta t for image j
        l      is lambda, the constant that determines the amount of smoothness
        w(z)   is the weighting function value for pixel value z

        Returns:
        g(z)   is the log exposure corresponding to pixel value z
        lE(i)  is the log film irradiance at pixel location i
        '''

        n = 256

        s1, s2 = Z.shape
        # Secondly, generate the matrix A and b
        A = np.zeros((s1 * s2 + n + 1, n + s1))
        b = np.zeros((A.shape[0], 1))

        # include the data-fitting equations
        k = 0
        for i in range(s1):
            for j in range(s2):
                wij = w[Z[i, j]]
                A[k,Z[i, j]] = wij
                A[k, n + i] = -wij
                b[k] = wij * B[i, j]
                k += 1

        # fix the curve by setting its middle value to 0
        A[k, 129] = 0
        k += 1

        # include the smoothness equations
        for i in range(1, n - 2):
            A[k, i] = l * w[i + 1]
            A[k,i + 1] = -2 * l * w[i + 1]
            A[k,i + 2] = l * w[i + 1]
            k += 1

        # Solve the system using SVD
        x = np.linalg.lstsq(A, b)
        x = x[0]
        g = x[0 : n]
        lE = x[n: len(x)]

        return g,lE

## some param required to use
N = len(images)         #N指照片个数
row = len(images[0])    #len(images[0])获得的是images列表中第一个项的shape[0],也就是图像的width
col = len(images[0][0]) #len(images[0][0])获得的是images列表中第一个项的shape[1],也就是图像的height
l = 10


## add a hat weighting function
Zmin = 0
Zmax = 255
w = np.zeros((Zmax - Zmin + 1))
for z in range(Zmin, Zmax + 1):
    if z <= (Zmax + Zmin) // 2:
        w[z] = z - Zmin + 1
    else:
        w[z] = Zmax - z + 1

# Firstly, randomly select N points
##
numSamples = math.ceil(255 * 2 / (N - 1)) * 2
numPixels = row * col                #一张图片上总的像素个数
step = int(numPixels / numSamples)
sampleIndices = list(range(0, numPixels, step))[:-1]  #每张图像选择len(sampleIndices)个点,即numSamples个
flattenImage = np.zeros((N, 3, numPixels),dtype=np.uint8)  #flattenImage[i,j]表示第i张图像的第j通道的所有像素
for i in range(N):  #遍历每张图像
    for j in range(3):  #遍历每个通道
        flattenImage[i,j] = np.reshape(images[i][:,:,j], (numPixels,))  #取出每张图像在每个通道的所有像素点,将其flatten成一维
Z_r = np.zeros((numSamples, N), dtype=np.uint8)  #Z[i,j]表示在第j个图像上的第i个像素位置
Z_g = np.zeros((numSamples, N), dtype=np.uint8)
Z_b = np.zeros((numSamples, N), dtype=np.uint8)
for i in range(N):
    Z_b[:,i] = flattenImage[i,0][sampleIndices]
    Z_g[:,i] = flattenImage[i,1][sampleIndices]
    Z_r[:,i] = flattenImage[i,2][sampleIndices]
index_ = np.arange(0, numSamples)  #要保留的位置
idx = []   #要剔除的位置
for i in range(numSamples):
    for k in range(N - 1):
        if Z_g[i, k] > Z_g[i, k + 1]:   #这里让像素值呈升序排列(输入图像是从低曝光时间到高曝光时间的顺序排列的,所以后一张图像的一个位置的像素值要高于前一张图像的对应位置的像素值),所以将不按升序排列的
            idx.append(i)
            break
index_ = np.delete(index_, idx, 0)
Z_b = Z_b[index_]
Z_g = Z_g[index_]
Z_r = Z_r[index_]
B = np.matlib.repmat(np.log(times), Z_b.shape[0] * Z_b.shape[1], 1)  #dim=0方向np.log(times)要重复Z_b.shape[0] * Z_b.shape[1]次,dim=1方向重复1次.


# Thirdly, solve the linear system using SVD
# this part is in the function 'gsolve'

# Finally, return the crf, which should be a numpy array of shape (3, 256) as we have 3 channels
##
g_b, lE_b = gsolve(Z_b)  #gsolve函数就对应Debevec的论文最后面的gsolve.m代码的python实现形式
g_g, lE_g = gsolve(Z_g)
g_r, lE_r = gsolve(Z_r)
```

In [ ]:

```python
### TODO: plot the CRF (3 plots for 3 channels) you have recovered.
xx = list(range(0, 256))
plt.title('Debevec Inverse Camera Response Function')
plt.plot(xx, np.exp(g_r), 'r')
plt.plot(xx, np.exp(g_g), 'b')
plt.plot(xx, np.exp(g_b), 'g')
plt.xlabel('Measured Intensity')
```

```
plt.ylabel('Calibrated Intensity')
plt.show()
```



Step 3: Merge Images

Once the CRF has been estimated, we can merge the exposure images into one HDR image using MergeDebevec with OpenCV.

In [ ]:
```
# Merge images into an HDR linear image using OpenCV's function
mergeDebevec = cv2.createMergeDebevec()
hdrDebevec = mergeDebevec.process(images, times, responseDebevec)
# You may want to save the HDR image (radiance map).
# cv2.imwrite("hdrDebevec.hdr", hdrDebevec)

import matplotlib.pyplot as plt
plt.imshow(hdrDebevec)
plt.show()
```

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

And you should implement your own code according to the equation below. The w is the hat weighting function we used before.

$$\ln E_i = \frac{\Sigma_{j=1}^{P} w(Z_{ij})(g(Z_{ij}) - \ln \Delta t_j)}{\Sigma_{j=1}^{P} w(Z_{ij})}$$

In [ ]:
```
### TODO: recover the radiance map

m = np.zeros(flattenImage.shape[1:])
wsum = np.zeros(flattenImage.shape[1:])
hdr = np.zeros(flattenImage.shape[1:])

lnDt = np.log(times)  #ln delta t
for i in range(N):
    wij_b = w[flattenImage[i,0]]
    wij_g = w[flattenImage[i,1]]
    wij_r = w[flattenImage[i,2]]
    wsum[0,:] += wij_b
    wsum[1,:] += wij_g
    wsum[2,:] += wij_r
    m0 = np.subtract(g_b[flattenImage[i,0]], lnDt[i])[:,0]
    m1 = np.subtract(g_g[flattenImage[i,1]], lnDt[i])[:,0]
    m2 = np.subtract(g_r[flattenImage[i,2]], lnDt[i])[:,0]
    hdr[0] += np.multiply(m0, wij_b)
    hdr[1] += np.multiply(m1, wij_g)
    hdr[2] += np.multiply(m2, wij_r)
hdr = np.divide(hdr, wsum)
hdr = np.exp(hdr)
hdr = np.reshape(np.transpose(hdr), (row, col, 3))

radiancemap = (hdr / np.amax(hdr) * 255).astype(np.float32)

# plot your radiance map
plt.imshow(cv2.cvtColor(radiancemap, cv2.COLOR_BGR2RGB))
plt.show()
```

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

Step 4: Tone mapping

Now we have merged our exposure images into one HDR image. Can you guess the minimum and maximum pixel values for this image? The minimum value is obviously 0 for a pitch black condition. What is the theoretical maximum value? Infinite! In practice, the maximum value is different for different situations. If the scene contains a very bright light source, we will see a very large maximum value.

Even though we have recovered the relative brightness information using multiple images, we now have the challenge of saving this information as a 24-bit image for display purposes.

> The process of converting a High Dynamic Range (HDR) image to an 8-bit per channel image while preserving as much detail as possible is called Tone mapping.

There are several tone mapping algorithms. OpenCV implements four of them. The thing to keep in mind is that there is no right way to do tone mapping. Usually, we want to see more detail in the tonemapped image than in any one of the exposure images. Sometimes the goal of tone mapping is to produce realistic images and often times the goal is to produce surreal images. The algorithms implemented in OpenCV tend to produce realistic and therefore less dramatic results.

Let's look at the various options. Some of the common parameters of the different tone mapping algorithms are listed below.

- gamma : This parameter compresses the dynamic range by applying a gamma correction. When gamma is equal to 1, no correction is applied. A gamma of less than 1 darkens the image, while a gamma greater than 1 brightens the image.
- saturation : This parameter is used to increase or decrease the amount of saturation. When saturation is high, the colors are richer and more intense. Saturation value closer to zero, makes the colors fade away to grayscale.
- contrast : Controls the contrast (i.e. log (maxPixelValue/minPixelValue)) of the output image.

Let us explore one of the tone mapping algorithms available in OpenCV.

Reinhard Tonemap

```
createTonemapReinhard
(
float   gamma = 1.0f,
float   intensity = 0.0f,
float   light_adapt = 1.0f,
float   color_adapt = 0.0f
)
```

The parameter intensity should be in the [-8, 8] range. Greater intensity value produces brighter results. light_adapt controls the light adaptation and is in the [0, 1] range. A value of 1 indicates adaptation based only on pixel value and a value of 0 indicates global adaptation. An in-between value can be used for a weighted combination of the two. The parameter color_adapt controls chromatic adaptation and is in the [0, 1] range. The channels are treated independently if the value is set to 1 and the adaptation level is the same for every channel if the value is set to 0. An in-between value can be used for a weighted combination of the two.

For more details, check out this paper.

In [ ]:

```
import matplotlib.pyplot as plt
# Tonemap using Reinhard's method to obtain 24-bit color image
tonemapReinhard = cv2.createTonemapReinhard(1.5, 0,0,0)

### The following two lines is to tone map radiance map "hdrDebevec" from OpenCV's algorithm. Please use this function to tone map your radiance map and plot it.
ldrReinhard = tonemapReinhard.process(hdrDebevec)
plt.imshow(ldrReinhard)
plt.show()
### TODO: Call OpenCV's function to tonemap the radiance map you have recovered.
ldrReinhard = tonemapReinhard.process(radiancemap)
plt.imshow(ldrReinhard)
plt.show()
# You may want to save the tonemapped image to a file.
# cv2.imwrite("ldr-Reinhard.jpg", ldrReinhard * 255)
```

Recent methods

There is a NTIRE challenge in HDRI, and the methods proposed in year 2021 (almost deep learning based) are summarized in this paper.

References

Book

- High dynamic range imaging: acquisition, display, and image-based lighting
  - Reinhard, Erik, et al. ,2010
  - The bible of the HDR imaging

HDR Image Reconstruction

- Debevec, Paul E., and Jitendra Malik. "Recovering high dynamic range radiance maps from photographs." Proceedings of the 24th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co., 1997.
  - The basic but effective method for HDR image reconstruction, this paper motivates many works about HDR imaging
  - Matrix form solution(including least square term and smoothness term)
- Robertson, Mark, Sean Borman, and Robert L. Stevenson. "Dynamic range improvement through multiple exposures." Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on. Vol. 3. IEEE, 1999.
  - Maximum likelihood solution, consider the additive noise and dequantization error as independent Gaussian random variable
  - Iterative method derived from the partial differential equation results
- Mitsunaga, Tomoo, and Shree K. Nayar. "Radiometric self calibration." Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.. Vol. 1. IEEE, 1999.
  - Use polynomials to model the CRF
  - Iterative method with rough estimation of exposure time ratio
- Lin, Stephen, et al. "Radiometric calibration from a single image." Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on. Vol. 2. IEEE, 2004.
  - Single image HDR scheme
  - Based on the edge color distribution

Image Alignment and Registration

- Ward, Greg. "Fast, robust image registration for compositing high dynamic range photographs from hand-held exposures." Journal of graphics tools 8.2 (2003): 17-30.
  - Median threshold bitmap (MTB) for global alignment
  - Very efficient (due to based on bitwise operation)
- Kang, Sing Bing, et al. "High dynamic range video." ACM Transactions on Graphics (TOG) 22.3 (2003): 319-325.
  - Both global and local registration
  - A variant of LK optical flow
  - A strategy to obtain HDR video

Alternative to HDR imaging Exposure Fusion Two series of paper

- Mertens, Tom, Jan Kautz, and Frank Van Reeth. "Exposure fusion." Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on. IEEE, 2007.
- Mertens, Tom, Jan Kautz, and Frank Van Reeth. "Exposure fusion: A simple and practical alternative to high dynamic range photography." Computer Graphics Forum. Vol. 28. No. 1. Blackwell Publishing Ltd, 2009.
  - Scalar-weighted map is first generated based on the quality measurement (contrast, saturation, well-exposedness) of the exposure bracketed image, then the fusion is performed in the multiresolution manner (Each layer of the Laplacian pyramid of resulting image is computed by the pixel-based multiplication if Gaussian pyramids of the weighted map with Laplacian pyramids of the original image)