令和 4 年度卒業制作

実写画像における階調の線形化

指導教員 高橋信雄
名古屋市立大学芸術工学部
産業イノベーションデザイン学科
学籍番号 195225
氏名 野本 駿輔

目次

実写画像における階調の線形化

Linearization of tones in live-action images

野本 駿輔 *Shunsuke Nomoto*

要旨

実写合成などに用いられる HDR 画像は，カメラレスポンスカーブと呼ばれる，カメラが輝度情報から画素値を決定する際に用いられる関数を用いて，複数の露光量の異なる画像を統合することで作成できる．しかし，このカメラレスポンスカーブが正確でないと HDR 画像がそのシーンにおいて自然でない色味になってしまうことがあり，しばしば問題となる．そこで，カメラレスポンスカーブを推定する際にサンプルとして利用される画素について，明度による分類と，サンプル箇所の周辺での画素値の変化の大きさで条件付けを行い，その精度を高めようとした．しかし，色味の偏りをなくす操作を行っていなかったために，条件によっては精度が落ちてしまうなどの問題があった．今後は各条件付けを見直す必要がある．

キーワード: 画像処理/CG/HDR/カメラレスポンスカーブ/階調特性

Abstract

HDR images used for live-action compositing can be created by merging multiple images with different exposures using a camera response curve, which is a function used by a camera to determine pixel values based on luminance information. However, if the camera response curve is not accurate, the HDR image may have colors that are not natural for the scene, which is often a problem. Therefore, i tried to improve the accuracy of the camera response curve by conditioning the pixels used as samples for estimating the curve on the classification of brightness and the magnitude of pixel value changes around the sampled points. However, since we did not perform any operation to eliminate color bias, there were some problems such as a drop in accuracy depending on the condition. It is necessary to review each conditioning in the future.

Keyword: Image processing/CG/HDR/Camera Response Curve/Gradation Characteristics

1. 序論

本節では，本制作で取り上げる実写画像における階調の線形化に関する研究背景と制作手法について述べる．

1.1. 研究・制作背景

CG 空間における階調が線形であるのに対し，一般的なカメラで撮影された画像の階調は，それぞれのカメラの持つカメラレスポンスカーブ(CRC: Camera Response Curve)と呼ばれる，輝度情報から画素値を決定する際に用いられる関数によって決定される．この階調の違いは，実写画像と CG の合成を行う際にシーンの違和感を引き起こすという問題がある.そこで用いられるのが HDR(HDR: High Dynamic Range)画像である．HDR 画像とは自然シーンのような幅広いレンジの輝度情報を保存することができる画像形式である.[1] またその階調は線形であるため，CG 空間上でそのシーンの自然な色味や明るさを表現することができ，IBL(IBL: Image Based Lighting)[2]と呼ばれる画像を光源として扱うレンダリング手法などに用いられる．

このように，HDR 画像は実写画像と CG の合成を行う際に非常に有用である．そこで，本制作では線形な階調を持つ HDR 画像の作成を行うプログラムの制作を行う．

1.2. 制作手法

HDR 画像の作成にあたり，まず CRC を取得する. CRC は多重露光画像統合と呼ばれる，複数の露光量を変えて撮影した実写画像とその露光時間から計算する手法[3]によって求めることができる .HDR 画像は取得した CRC にその逆関数をかけることで線形化し，露光時間から輝度値を計算し HDR を作成する[4].

この手法を用いた CRC の取得と HDR 画像の生成は，画像処理を行うことができる OpenCV ライブラリの cv.createCalibrateDebevec 関数と cv.createMergeDebevec 関数を使用することで行うことができる[5].

1.3. 本制作の目標

前節で述べたように，CRC の取得と HDR 画像の生成は既存のライブラリを利用することで容易に行うことができる.しかし，CRC の取得についてはしばしばその精度が問題となる. CRC は露光量の異なる複数の画像を撮影する必要があるが，図 1 に示すように使用する画像や撮影環境によって大きく結果が異なる．そこで，本制作ではより精度の高い CRC の取得，及びそれを用いた HDR 画像の生成を目標とする.

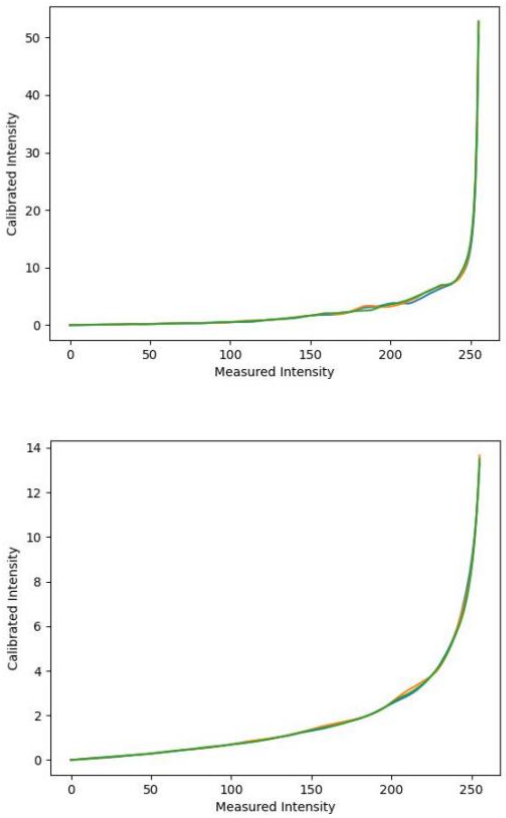

図 1 同じカメラで撮影した異なる画像によって出力された CRC

1.4. 撮影方法

撮影に使用するカメラは SONY 社製 ILCE-6300 を使用する．また撮影時のブレを排除するため，三脚を用い，さらに同社が提供している Imaging Edge Mobile を使用し，スマートフォンを用いて遠隔で撮影を行う.

撮影は太陽光量が安定している雲の少ない天気の良い日に行う．その際，直射日光や強い反射光が直接写らないよう留意する.

撮影時は ISO 感度，焦点距離，F 値は固定し，ホワイトバランスはカメラの自動調節機能を利用する．また，シャッタースピードは手動で操作し，露光量を変える.

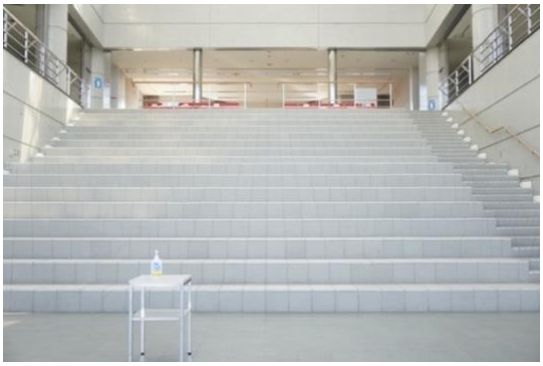

図 2 ISO 感度 100，焦点距離 24，F 値 f/8，露出時間 1/6 で撮影した画像

本制作ではより細部の精度を高めるため，画面全体が黒潰れする 1/4000 秒から画面全体が白潰れする 4 秒まで，1/3 段ずつシャッタースピードを変えて 43 枚の画像を撮影した．

2. プログラムの制作

2.1. 使用する言語・ライブラリ

・Python 3.10.5
・OpenCV
・Numpy
・Math
・Cmath
・Scipy.stats
・Matplotlib
・Copy

2.2. 精度を上げるための処理

本制作の目標として，より精度の高い CRC の取得とそれを用いた HDR 画像の生成を掲げた．そのための手法として，画素のサンプリングを行う際に，撮影時の細かなブレ等の影響を受けやすいオブジェクトのエッジ付近の画素を避け，色が安定している部分からサンプリングを行うこととした．また，白潰れした部分から多くサンプリングしてしまうなど，サンプリングに偏りがないよう，画素に対して明度によるばらつきが出るようサンプル箇所に条件を設定した．

前節でも述べたが CRC は既存のライブラリ関数[6] (1)，(2)を使用することで取得することができる．

cv::createCalibrateDebevec(samples,lamda,random) (1)

cv::createCalibrateCRF.process(src,dst,times) (2)

(1)の第 3 引数 random は bool 値によって定義され，true であれば画像中からランダムに，false であれば第 1 引数 samples に設定された数値を元に，画像中から一定の幅でサンプリングを行う．この引数によってサンプル箇所が決定されるため，自身で任意の点をサンプル箇所に設定することはできないという問題がある.そのため本制作では，関数側で設定されたサンプル箇所に対して，先に述べた条件を満たす画素値を代入するという手法を用いてサンプルの入れ替えを行った．

2.2.1. 条件の良いピクセルの選定

前節で述べたオブジェクトのエッジ部分を避ける処理については，サンプル箇所とその周辺の画素値の違いに注目した．まず，サンプル箇所の画素値とその周辺の画素値についてサンプル箇所との距離に応じた差の加重平均をとり，差の小さいものから順にリスト化した．さらにリスト化した各サンプル箇所の座標における各画像の明度の合計値によってさらに分類を行い，それぞれ分類した中から加重平均の値が小さいものから順に選びそれを新たなサンプル箇所に設定した．

2.2.2. 重み付け

画像処理における重み付け処理では，ガウス分布に沿ったガウシアンフィルタが代表されるが，図 3 に示すようにガウシアンフィルタは

あらかじめ適用される範囲の値が設定されている．そのため本制作では，そのグラフの特徴を持ち，かつ任意の範囲で重み付けを行うことができる標準正規分布の確率密度関数[7]を用いて差の加重平均を取る際の重み付けを行った．ただし，確率密度分布から得たサンプル箇所からの距離による重みの総和は 1 にならないため，加重平均を取る際にその重みの総和で徐算する必要がある．

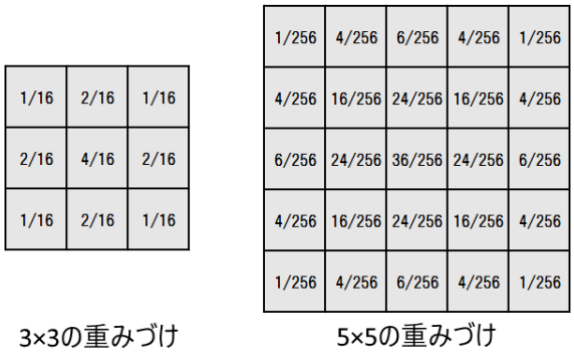

図 3 ガウシアンフィルタの例

(出典：三谷商事株式会社ビジュアルシステム部．”ミブログ MiVlog ガウシアンフィルター画像処理におけるノイズの除去”．(httph://www.mitani-visual.jp/mivlog/imageproimagepr/gf3r89.php)．(2023-1-30))

2.2.3. 平滑化

前節の(1)で示すように，出力される CRC の滑らかさに関する数値が第 2 引数に設定されており，これによって出力されるグラフの形が変わることがある．そのため，本制作では精度の違いについて比較を行うため,この数値を 0 に設定し，外れ値に対してはその部分で線形補間を行った．

3. ソースコード

3.1. ライブラリのインポート

```
from cmath import inf
from statistics import variance
import cv2
import numpy as np
from matplotlib import pyplot as pyp
import math
from scipy.stats import norm
import copy
```

3.2. 使用する画像のインポート

```
rows = 4000
cols = 6000
sample = 50

img1 = cv2.imread('./Python/HDR/SourcePicture/DSC00325.jpg')
img43 = cv2.imread('./Python/HDR/SourcePicture/DSC00371.jpg')

img_list = [img1, img43]
```

画像枚数が多いため一部のみ抜粋．

・rows，cols：インポートする画像の辺の長さ．

・sample：cv2.createCalibrateDebevec 関数の第 1 引数．

・img_list：インポートした画像を格納するリスト．

3.3. サンプル箇所の取得

3.3.1. 元のサンプル箇所の特定

```
def get_samplepoints():
sample_point = []
x_points = int(math.sqrt(int(sample) * cols / rows))
```

```
if 0 < x_points and x_points <= cols:
y_points = int(sample/x_points)

if (0 < y_points and y_points <= rows):
step_x = int(cols/x_points)
step_y = int(rows/y_points)

from_x = int(step_x/2)
from_y = int(step_y/2)

to_x = from_x + step_x * x_points
to_y = from_y + step_y * y_points

for x in range(from_x,to_x,step_x):
for y in range(from_y,to_y,step_y):
if 0<= x and x < cols and 0 <= y and y < ro
ws:
sample_point.append((y,x))

return sample_point,step_x,step_y
```

関数 get_samplepoints:

・戻り値 sample_point: cv.createCalibrateDebeve c 関数によって定義されるサンプル箇所[8].

・戻り値 step_x, step_y: サンプリングが行われる際の幅.

3.3.2. 新たなサンプル箇所の設定

```
def new_sample_pos(width,radius):

center_pos_list = []
sample_pos_list = []
num_r = int(rows / width)
num_c = int(cols / width)

f = get_samplepoints()
img_samplepoint = f[0]

for c in range(1,num_c,1):
for r in range(1,num_r,1):
center_pos_list.append((width*r,width*c))

for i in img_samplepoint:
center_pos_list.append(i)

center_pos_list = set(center_pos_list)

for i in center_pos_list:

if (i[0]>= radius and i[1]>=radius) or (i[0]<=ro
ws-radius and i[1]<=cols-radius) :
sample_pos_list.append(i)

return sample_pos_list
```

関数 new_sample_pos:

・引数 width: サンプリングを行う際の参照幅.

・引数 radius: 周辺ピクセルの範囲.

・戻り値 sample_pos_list: 新たに設定したサンプル箇所.

引数 width を新たに設定し，格子状のグリッド上の点を center_pos_list に設定する.さらに元のサンプルポイント img_samplepoint を senter_ pos_list に追加し，新たなサンプル箇所 sample_ pos_list とした.

3.4. 加重平均の算出

```
def
pixelvalue_distinction_inimg(img,width,radius):

weighted_ave_list_inimg = []
sample_pos_list = new_sample_pos(width,radius)
```

```
for i in sample_pos_list:
around_pixel_list = []
for w in range((i[0]-radius),(i[0]+radius+1),1):
for j in range((i[1]-radius),(i[1]+radius+1),1):
if (w-i[0])**2+(j-i[1])**2 <= radius**2:
around_pixel = (w,j)
around_pixel_list.append(around_pixel)

ave_b_list = []
ave_g_list = []
ave_r_list = []

weighted_val_list = []

for inside_pixel in around_pixel_list:
pixel_dis = (math.sqrt((inside_pixel[0]-i[0])**2
+ (inside_pixel[1]-i[1])**2))

weighted_val = norm.pdf(pixel_dis)
weighted_val_list.append(weighted_val)

val_b = img[inside_pixel[0],inside_pixel[1],0]
val_g = img[inside_pixel[0],inside_pixel[1],1]
val_r = img[inside_pixel[0],inside_pixel[1],2]

ave_b = val_b * weighted_val
ave_g = val_g * weighted_val
ave_r = val_r * weighted_val

ave_b_list.append(ave_b)
ave_g_list.append(ave_g)
ave_r_list.append(ave_r)

weighted_ave_b = 0
weighted_ave_g = 0
weighted_ave_r = 0
weighted_val_sum = 0

for b in ave_b_list:
weighted_ave_b += b
for g in ave_g_list:
weighted_ave_g += g
for r in ave_r_list:
weighted_ave_r += r

for weight in weighted_val_list:
weighted_val_sum += weight

weighted_ave = [weighted_ave_b/weighted_val_s
um,weighted_ave_g/weighted_val_sum,weighted_a
ve_r/weighted_val_sum]
weighted_ave_list_inimg.append([i,weighted_ave])
return weighted_ave_list_inimg
```

関数 pixelvalue_distinction_inimg:

・引数 img: 入力画像.

・引数 width: サンプリングを行う際の参照幅.

・引数 radius: 周辺ピクセルの範囲.

・戻り値 weighted_ave_list_inimg: img における sample_pos_list の各ピクセルの座標とそれに対応する加重平均を格納した 2 次元配列.

3.5. 各サンプル箇所における加重平均の算出

```
def samplepoint_condition(width,radius):

original_pos = get_samplepoints()[0]
original_pos_val = []

sample_pos = new_sample_pos(width,radius)
sample_num = len(sample_pos)
pos_sum_val_eachimg = []
```

```
sum_val_eachpos = [[] for i in range(sample_num)]

for img_num in range(len(img_list)):

img = img_list[img_num]
pos_sum_val_eachpos = []
weighted_ave_list_inimg = pixelvalue_distinction_inimg(img,width,radius)

for i in weighted_ave_list_inimg:

samplepoint_pixelvalue_b = img[i[0][0],i[0][1],0]
weighted_ave_val_b = i[1][0]

samplepoint_pixelvalue_g = img[i[0][0],i[0][1],1]
weighted_ave_val_g = i[1][1]

samplepoint_pixelvalue_r = img[i[0][0],i[0][1],2]
weighted_ave_val_r = i[1][2]

val_diff_b = abs(samplepoint_pixelvalue_b-weighted_ave_val_b)
val_diff_g = abs(samplepoint_pixelvalue_g-weighted_ave_val_g)
val_diff_r = abs(samplepoint_pixelvalue_r-weighted_ave_val_r)

val_sum = val_diff_b+val_diff_g+val_diff_r
pos_sum_val_eachpos.append([i[0],val_sum])
pos_sum_val_eachimg.append(pos_sum_val_eachpos)

for num in pos_sum_val_eachimg:
for i in range(sample_num):
sum_val_eachpos[i].append(num[i])

sum_total_eachpos = [[i,0] for i in sample_pos]

for i in range(sample_num):
for img_num in range(len(img_list)):
sum_total_eachpos[i][1] += sum_val_eachpos[i][img_num][1]
if sum_total_eachpos[i][0] in original_pos:
original_pos_val.append(sum_total_eachpos[i].copy())

lower_val_pos = []
sum_total_eachpos = sorted(sum_total_eachpos,key = lambda x:x[1])

original_pos_val = sorted(original_pos_val,key = lambda x:x[1])
lower_lim = original_pos_val[len(original_pos_val)-1][1]
for i in sum_total_eachpos:
if i[1] <= lower_lim:
lower_val_pos.append(i)
else:
break
return lower_val_pos,original_pos_val
```

関数 samplepoint_condition:

・引数 width: サンプリングを行う際の参照幅.

・引数 radius: 周辺ピクセルの範囲.

・戻り値 lower_val_pos: 元のサンプル箇所のうち，加重平均の最も大きい値を lower_val とし，その値以下を持つものを sum_total_eachpos から lower_pos に追加する.

・戻り値 original_val: 元のサンプル箇所の座標とそれぞれの加重平均を持つ 2 次元のリストを加重平均の値で昇順に並べ替えたもの.

3.6. 明度による分類

```
def check_luminance(width,radius):


hsv_img_list = []
samplepointcondition = samplepoint_condition(width,radius)
sample_pos = samplepointcondition[0]
original_pos_val = samplepointcondition[1]
img_list_copy  = copy.deepcopy(img_list)

for img in img_list_copy:
img = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
hsv_img_list.append(img)

pos_lumi_val_list = []

for pos in sample_pos:
pos_lumi_val = [pos,0]
for img in hsv_img_list:
pos_lumi_val[1] += img[pos[0]][2]
pos_lumi_val_list.append(pos_lumi_val)

lumi_list = []
for i in pos_lumi_val_list:
lumi_list.append(i[1])

max_lumi_val = 255 * len(img_list)
num = 1
bins_f = []

for i in range(1,num+1,1):
x = int(i/num*max_lumi_val)
bins_f.append(x)
separate_lumi_val_list = np.digitize(lumi_list,bins=bins_f)
separate_pos_lumi_list = [[] for i in range(num)]

for i in range (len(pos_lumi_val_list)):
for n in range(num):
if n == separate_lumi_val_list[i]:

separate_pos_lumi_list[n].append(pos_lumi_val_list[i][0][0])


return separate_pos_lumi_list,original_pos_val
```

関数 check_luminance:

・引数 width: サンプリングを行う際の参照幅.

・引数 radius: 周辺ピクセルの範囲.

・戻り値 separate_pos_lumi_list: 関数 samplepoint_condition によって取得した lower_val_pos の各座標について，各画像における明度の総和を求め，num 個の要素に分類したもの.

・戻り値 original_pos_val: 関数 samplepoint_condition で取得したものと同じもの.

3.7. 画素値の代入

```
def replace_pixel(width,radius):


checkluminance = check_luminance(width,radius)
original_pos_val = checkluminance[1]
original_pos = []

for i in original_pos_val:
original_pos.append(i[0])

separate_pos_lumi_list = checkluminance[0]
replacement_point_num = 0
```

```
replacement_point = []

while replacement_point_num < len(original_pos):
for i in range(len(separate_pos_lumi_list)):
n = len(separate_pos_lumi_list[i])
if n > 0:
x = separate_pos_lumi_list[i][0]
replacement_point.append(x)
separate_pos_lumi_list[i].remove(x)
replacement_point_num += 1

else:
continue

if replacement_point_num == len(original_pos):
break

for i in replacement_point:
if i in original_pos:

sort_y = sorted(replacement_point, key = lambda x:(x[1],x[0]))

im_list = copy.deepcopy(img_list)
pix_val = [[] for i in range(len(img_list))]
for num in range(len(img_list)):
for i in sort_y:
pix_val[num].append(img_list[num][i])

for num in range(len(im_list)):
for i in range(len(original_pos)):
val = pix_val[num][i]
for x in range(-15,15,1):
for y in range(-15,15,1):
pos = (original_pos[i][0]+x,original_pos[i][1]+y)

filename = "./Python/HDR/Replaceimg_lambda0/Replaced_img/test/im_list" + str(num) + ".jpg"
cv2.imwrite(filename,im_list[num])
cv2.imshow("dst1",im_list[25])
```

関数 replace_pixel:

・引数 width: サンプリングを行う際の参照幅.

・引数 radius: 周辺ピクセルの範囲.

関数 check_luminance で取得した separate_pos_lumi_list について，元のサンプル数を満たすまで各要素内リストの各要素を順に replacement_point に追加し,それを新たなサンプル箇所とする.

変数 radius については，元のサンプルとその周辺の画素値を変更する際に参照する範囲が 15 以上になると結果に変化が見られなかったため，本制作では radius の値を 15 と設定した.

元のサンプル箇所の各座標について，その座標を含む x 座標 y 座標それぞれ-15 から 15 の範囲に存在する周辺のピクセルの画素値について，図 4 に示すように，新たに設定したサンプル箇所の画素値を代入し，その結果を新たな画像として保存する.

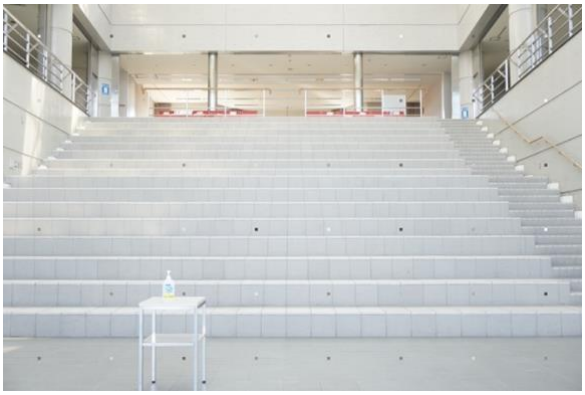

図 4 新たに作成した画像

3.8. 新たに作成した画像の読み込み

```
img_list_replaced = []


for num in range(len(img_list)):
filename = "./Python/HDR/Replaceimg_lambda0/Replaced_img/notreplace/im_list" + str(num) + ".jpg"
img = cv2.imread(filename)
img_list_replaced.append(img)
```

関数 replace_pixel にて新たに作成した画像を読み込み,img_list_replaed リストに追加する.

3.9. CRC の出力

```
def CalibrateDebevec():
cal_debevec = cv2.createCalibrateDebevec(samples=sample,lambda_=10,random=False)
crf_debevec = cal_debevec.process(img_list,exporsure_times.copy())
crf_debevec = crf_debevec.astype(np.float32)


crf_b = crf_debevec[:,:,0]
crf_g = crf_debevec[:,:,1]
crf_r = crf_debevec[:,:,2]


def check_bgr():
for c in range(3):
for i in range(256):
if (crf_debevec[:,:,c][i] == inf and i > 1 and i < 255) or (crf_debevec[:,:,c][i] == 0 and i > 1 and i < 255) or (crf_debevec[:,:,c][i] > 100 and i > 1 and i < 255):
for num in range(1,255,1):
if crf_debevec[:,:,c][i+num] != 0 and crf_debevec[:,:,c][i+num] != inf and crf_debevec[:,:,c][i+num] < 100:
for n in range(num):
crf_debevec[:,:,c][i+n] = crf_debevec[:,:,c][i+n-1] + ((crf_debevec[:,:,c][i+num] - crf_debevec[:,:,c][i-1]) / (num+1))
if crf_debevec[:,:,c][i+num] == 0 or crf_debevec[:,:,c][i+num] == inf or crf_debevec[:,:,c][i+num] > 100:
continue
break
else:
continue


check_bgr()


crf_b = crf_b.flatten()
crf_g = crf_g.flatten()
crf_r = crf_r.flatten()


crf_gragh1 = pyp.plot(crf_b)
crf_gragh2 = pyp.plot(crf_g)
crf_gragh3 = pyp.plot(crf_r)
pyp.xlabel("Measured Intensity",{"fontsize":10})
pyp.ylabel("Calibrated Intensity",{"fontsize":10})


file_gragh = "./Python/HDR/Replaceimg_lambda0/OutputGragh/" + condition + ".jpg"
pyp.savefig(file_gragh)


file_curve = "./Python/HDR/Replaceimg_lambda0/ResponseCurve/" + condition + ".npy"
np.save(file_curve,crf_debevec)
```

関数 check_bgr:

cv2.createcalibrateDebevec 関数によって出力された CRC(crf_debevec)に対して，BGR 各チャネルごとにそれぞれの値が適切な範囲に含まれているか調べ，0 や inf といった外れ値があれ

ばその前後の値で線形補間を行い，値を修正する.その後，各チャネル別に pyp.plot 関数を用いてグラフとして出力し,任意のフォルダに保存する.

3.10. HDR 画像の生成

```
def MergeDebevec():

file_reloadcurve = "./Python/HDR/Replaceimg_lambda0/ResponseCurve/" + condition + ".npy"
crf_debevec = np.load(file_reloadcurve)

merge_debevec = cv2.createMergeDebevec()
hdr_debevec = merge_debevec.process(img_list,exporsure_times.copy(),crf_debevec)

file_HDR = "./Python/HDR/Replaceimg_lambda0/Created_HDR/" + condition + ".hdr"
cv2.imwrite(file_HDR,hdr_debevec)
```

関数 MergeDebevec:

出力された CRC(crf_debevec)から cv2.createMergeDebevec 関数を用いて，img_list の各画像を HDR 画像に統合する.その後，出力された HDR 画像 hdr_debevec を任意のフォルダに保存する.

4. 結果と考察

新たにサンプリングを行う際の参照幅を width，サンプル箇所の明度による分類の数を num とし，num を 1・64・128・192・256，width を 200・500・1000 と変化させ結果の出力を行った．また，画素値の入れ替えを行わずに出力したものを従来手法 CRC(CRC: Camera Response Curve)，width の値を 1000，num の値を 64 として画素値の入れ替えを行ったものを num64_width1000 とし，各条件による比較を行った.

4.1. CRC の出力結果

RGB チャンネルごとに出力された CRC を図 5 に示す．ただし，青はカラーチャンネル B，緑はカラーチャンネル G，オレンジはカラーチャンネル R のグラフを示す.

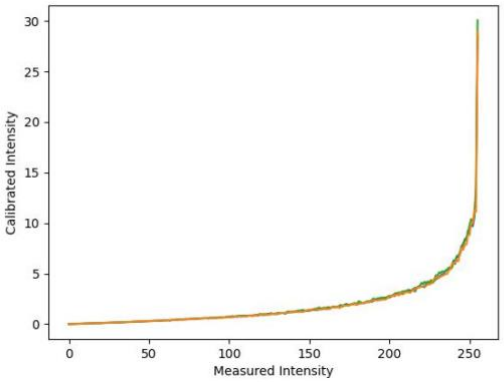

図 5.1. 従来手法 CRC

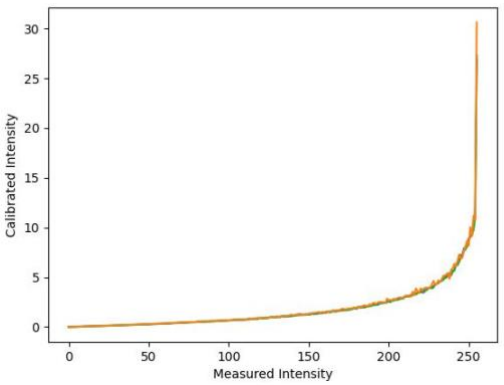

図 5.2. num1_width1000

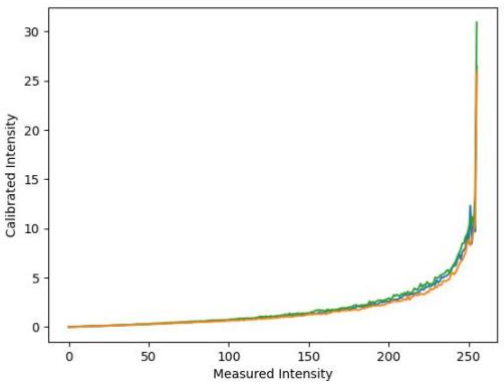

図 5.3. num1_width500

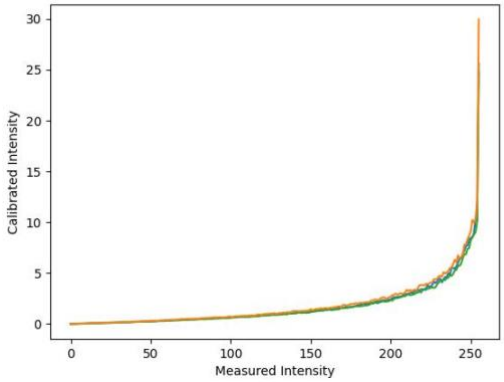

図 5.4. num1_width200

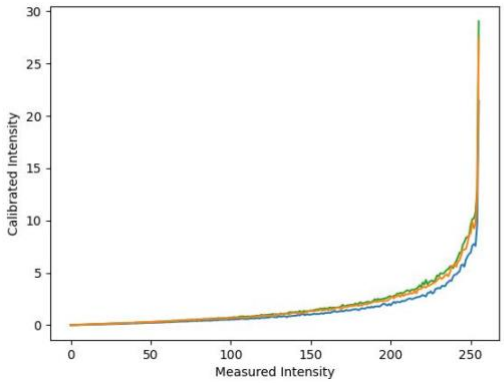

図 5.5. num64_width1000

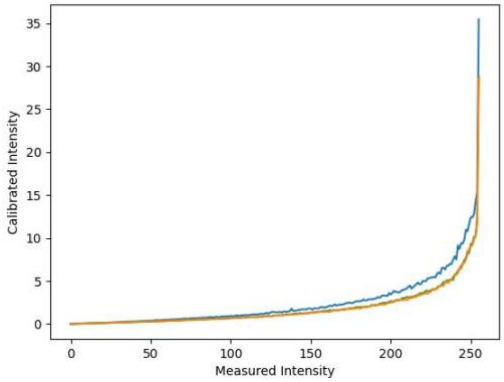

図 5.6. num64_width500

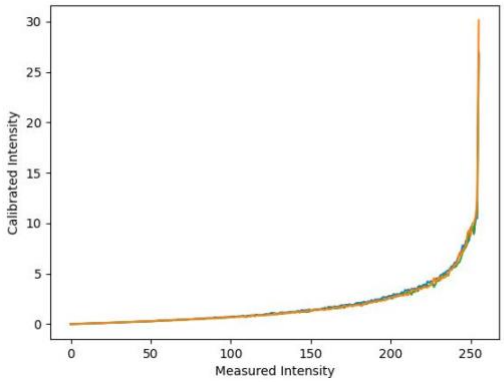

図 5.7. num64_width200

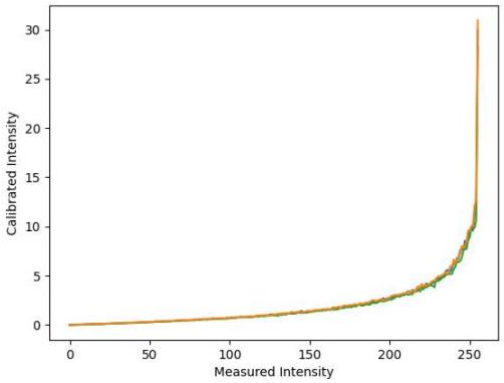

図 5.8. num128_width1000

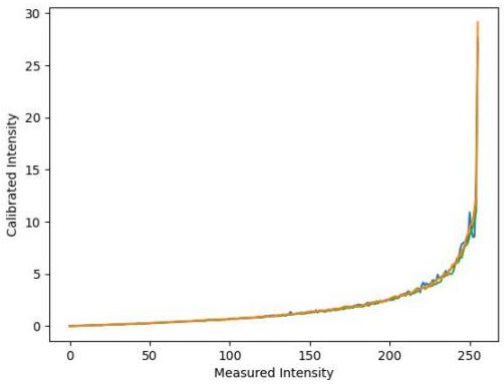

図 5.9. num128_width500

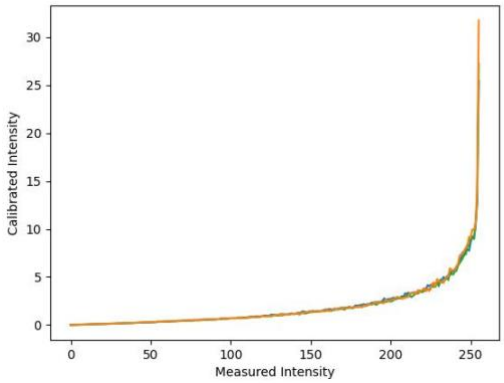

図.5.10. num128_width200

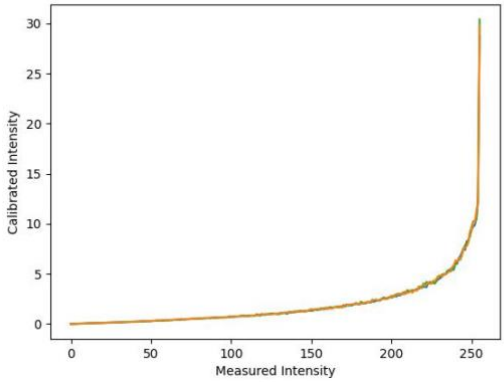

図 5.11. num192_width1000

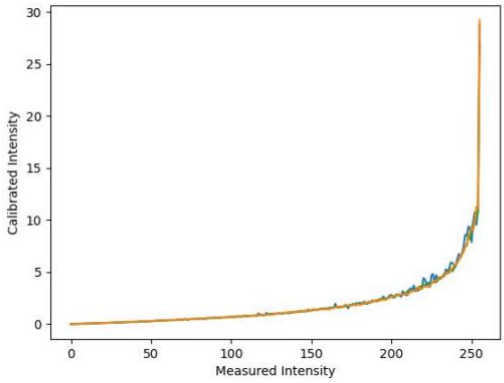

図 5.12. num192_width500

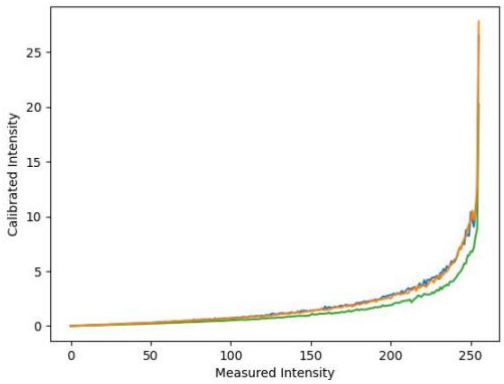

図 5.13. num192_width200

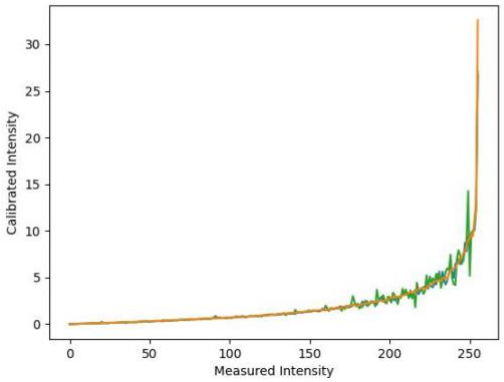

図 5.14. num256_width1000

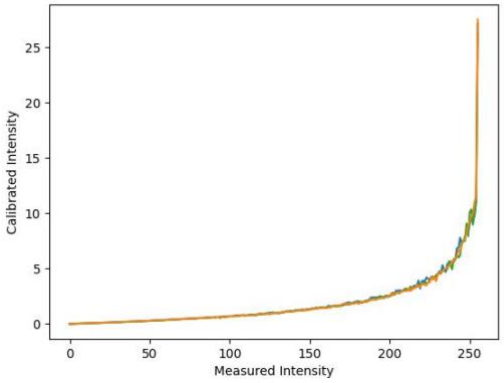

図 5.15. num256_width500

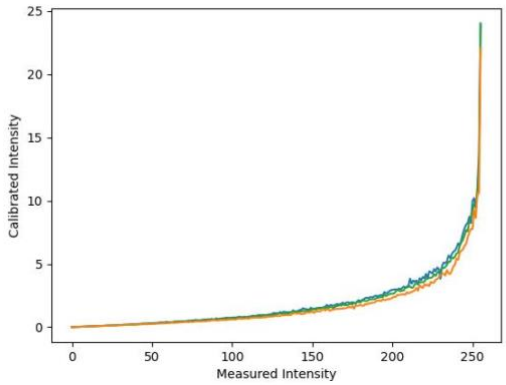

図 5.16. num256_width200

4.2. 出力された HDR 画像

図 5 で示した，各条件で出力された CRC を使用して統合した HDR 画像を，画像処理を行うことができるソフトウェアである GIMP を用いて露出を調整し図 6 に示す．ただし，枚数が多いため従来手法 CRC と width=500 のものを使用した HDR 画像のみ抜粋して示す．

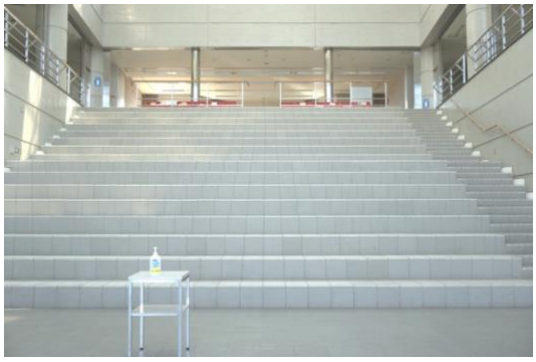

図 6.1 従来手法 CRC を使用した HDR 画像

図 6.2 num1_width500 を使用した HDR 画像

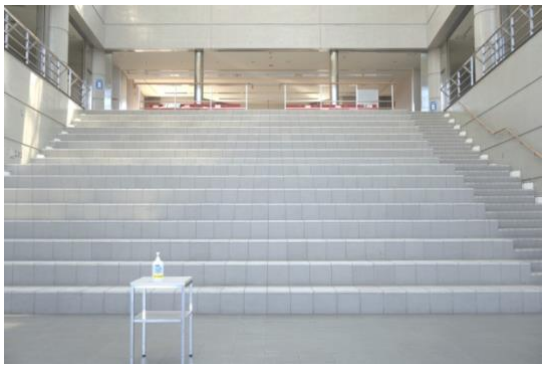

図 6.3 num64_width500 を使用した HDR 画像

図 6.4 num128_width500 を使用した HDR 画像

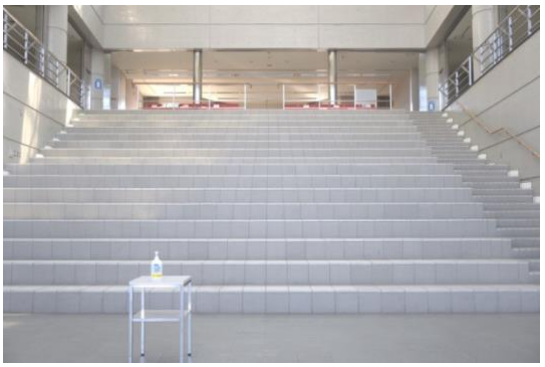

図 6.5 num192_width500 を使用した HDR 画像

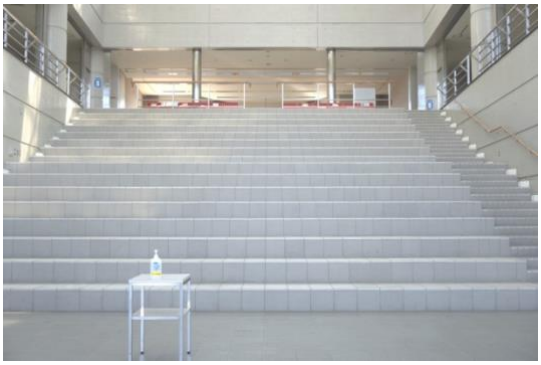

図 6.6 num256_width500 を使用した HDR 画像

4.3. 出力グラフの比較方法

出力された CRC について，各条件における CRC のグラフを従来手法 CRC のグラフと比較する．そのための方法として,CRC の生成に使用した露光量の異なる画像から，真の CRC の影響を受け撮影されたピクセルの画素値の変化をグラフにしたものを真の CRC とし，それと出力された CRC の形を比較することで，条件によってどの程度形が変わったかを分析する．ただし，どちらも y 軸が相対的な輝度であるため，直接的な数値による比較はできず，あくまでグラフの形の比較であることに注意する必要がある.以下にその比較の手法を示す.

まず，使用した画像中でグレーのピクセルを代表ピクセルとして設定する．ただし，今回は全画像を通じてグレーであるピクセルがなかったため，グレーに近いピクセルを代表ピクセルとした.

次に代表ピクセルについて，x 軸を画素値，y 軸を画像枚数として，画素値の変化を図 7 のようにプロットした．ただし，真の CRC は RGB が一致することから，各値の平均を算出し，プロットした.

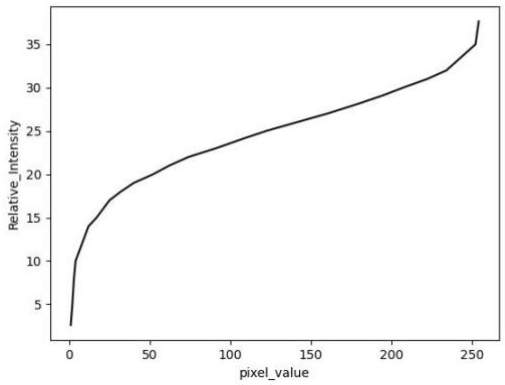

図 7 代表ピクセルの画素値のプロット

図 7 で示したグラフについて，x 軸に対しデータの無い部分に，(x,0)という新たなデータを追加した上でその部分で線形補間を行う．その際，同じ x の値に複数のデータが存在する場合はその平均値を採用した．また,x=0 と X=255 は値が収束し，正確な値を記録できないと考えられるため除外し，画素値の範囲を 1 から 254 までとする.

最後に，データを補間した真の CRC と出力された CRC の形の比較を行う．ただし，出力された CRC について，y 軸の尺度を揃えるために，図 8 に示すように各 x における y の値に対して底 2 の対数を取る．また，情報の精度が高いと考えられる画素値 128 を基準とし，x=128 の y の値が 0 になるようそれぞれのグラフを移動させた後，x=64，x=192 の点が真のレスポンスカーブと重なるよう定数倍した際の定数の平均をグラフ全体にかけ，グラフの y 軸のスケールを調整する．x=64,x=192 の点を基準にした理由としては，端に行くほど誤差が大きいと考えられるため，x=128 を基準とした際，左右どちらかに寄った定数を掛けてしまうと，全体的な誤差が大きくなるのではないかと考えたためである．また，真の CRC は RGB が一致す

ることと，G チャンネルの画素値は他のチャンネルと比べ感度が高いことから，全体に掛ける定数は G チャンネルのものを使用した．

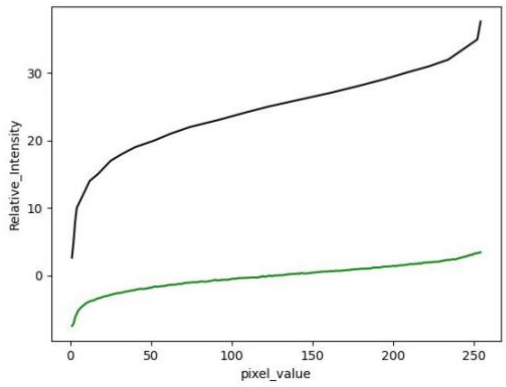

図 8 CRC のプロット

（黒：真の CRC，緑：出力された CRC の G チャンネル）

また例として，図 9 に上記の手順を行いプロットした従来手法 CRC と num1_width200，そして真の CRC を示す．

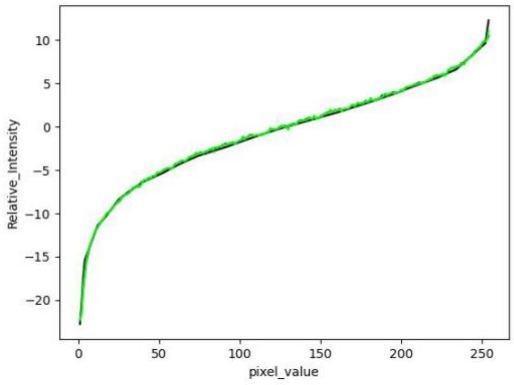

図 9 3 つのグラフをプロットしたもの

(黒:真の CRC,濃緑:num1_width1000 の G チャンネル,薄緑:従来手法 CRC の G チャンネル)

グラフの形の比較には MSE(MSE: Mean Squared Error)と呼ばれる指標[9]を用いる．MSE は誤差の二乗値の総和をデータ数で割ることで求められる．

4.4. MSE の算出結果

各条件における真の CRC との MSE を表 1 に示す．

表 1 各条件における MSE

	B	G	R
従来手法CRC	0.17769926	0.07316706	0.16359062
num1_width1000	0.19995598	0.0752242	0.11894567
num1_width500	0.28918383	0.0923381	0.1623674
num1_width200	0.16866096	0.08317003	0.13259114
num64_width1000	0.23158285	0.1122143	0.13146321
num64_width500	0.12575615	0.06483422	0.08064288
num64_width200	0.19525366	0.1192647	0.08722514
num128_width1000	0.21026205	0.07423166	0.12387752
num128_width500	0.2573693	0.09984774	0.15762867
num128_width200	0.14561627	0.10181128	0.07567852
num192_width1000	0.16900623	0.12406873	0.15123247
num192_width500	0.13798146	0.07311791	0.10209797
num192_width200	0.19451191	0.09318337	0.10373156
num256_width1000	0.18011414	0.50982666	0.0889003
num256_width500	0.08042694	0.09127844	0.11116006
num256_width200	0.20873614	0.16250509	0.1078881

また，各条件における真の CRC との MSE について，従来手法 CRC における MSE との差を比較した結果を表 2 に示す．ただし，負の値は従来手法と比べ誤差が減少したことを示し，正の値は誤差が増加したことを示している．

表 2 従来手法との比較

	diff_B	diff_G	diff_R
num1_width1000	0.022257	0.002057	-0.044645
num1_width500	0.111485	0.019171	-0.001223
num1_width200	-0.009038	0.010003	-0.030999
num64_width1000	0.053884	0.039047	-0.032127
num64_width500	-0.051943	-0.008333	-0.082948
num64_width200	0.017554	0.046098	-0.076365
num128_width1000	0.032563	0.001065	-0.039713
num128_width500	0.07967	0.026681	-0.005962
num128_width200	-0.032083	0.028644	-0.087912
num192_width1000	-0.008693	0.050902	-0.012358
num192_width500	-0.039718	-4.90E-05	-0.061493
num192_width200	0.016813	0.020016	-0.059859
num256_width1000	0.002415	0.43666	-0.07469
num256_width500	-0.097272	0.018111	-0.052431
num256_width200	0.031037	0.089338	-0.055703

4.5. 出力結果の考察

図 5 と表 1,2 で示した結果について，各 width における num について比較すると，明度による分類の数と誤差の大きさに比例関係は見られなかった．そのため，今回制作したプログラムでは，サンプル箇所を決定する際に各ピクセルについて明度の総和による分類を行うことに明確な効果を見出すことはできなかった．これについては色味の偏りによる影響があると考える．明度によるばらつきを与えることで，白潰れ部分のみをサンプル箇所にしてしまうなどの偏りはなくすことができると考えるが，明度の異なる特定の色味を帯びたピクセルを多くサンプル箇所にしてしまうなどの偏りを排除することができないのではないかと考える．

width の値における変化については，参照幅を小さくし，画像中からより多くのピクセルを参照すること自体は有効であると考えられるが，やはりサンプル箇所の色味によっては偏りが大きくなり,結果に大きな影響を与えてしまうという結果になった．

また，図 6 で示した統合した HDR については，統合の際に使用する CRC の特性によって色味が大きく変わることが再確認された．

5. 改善点と展望

5.1. 改善点

本制作では，CRC と HDR 画像を出力するプログラムを制作したが，従来手法 CRC よりも精度の高い CRC を出力するプログラムの実装には至らなかった．しかし，明度によるサンプル箇所の分類の他に,色味による分類を行う必要性があることが結果から推測された．

また，サンプル箇所を決定する際の参照数を条件として設定したが，サンプル箇所の増減による結果の比較は行っていないため，その部分の比較も行う必要があると考える．

さらに，本制作では新たな画素値を元のサンプル箇所に代入する際，その適用範囲を 15 に固定して結果の出力を行ったが，図 10 に示すように，その値によって同条件でも結果が変わってしまったため，サンプル箇所のみを参照しているわけではないと考えられる．そのため，サンプリングを行う際，どの程度の範囲を参照し，その値によってどの程度結果が変わるのか，優先的に検証を行う必要がある．

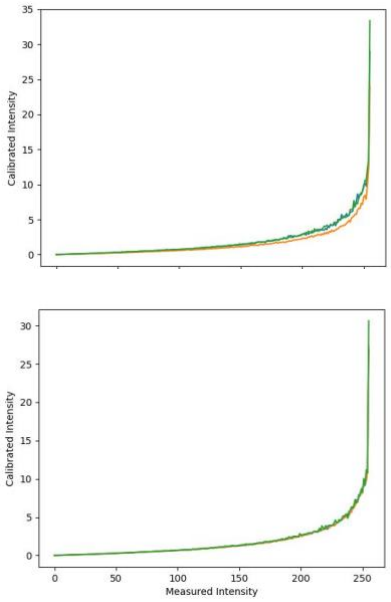

図 10 適用範囲による出力結果の違い

重み付けに関しても，新たな画素値を元のサンプル箇所に代入する際の適用範囲から重み付けを行う範囲を設定したが，その範囲の見直しが必要である．また，本制作では標準正規分布の確率密度関数を用いて重み付けの値を算出したが，その性質上，0 付近の重みの比重が非常に大きいため，よりバランスの良い重み付けの値の設定が求められる．

結果の比較方法について，本制作ではグラフの形に着目し，形の相似度合いを示す MSE の算出によって比較を行ったが，あくまで形の比較であるため，見かけ上は一致しているように見えても実際には RGB が一致していない場合があり，精度についての検証には適していない．そのため，データに即したより厳密な比較手法を確立することが求められる．

5.2. 展望

本制作をもとに，より良いプログラムの開発を行い，誰でも簡単に自然な HDR 画像を作成できるようなプログラムの実装，及び製品化につながることを願う．

6. 謝辞

本制作を進めるにあたり，数々の助言を賜り，制作の方向性に具体性を与えてくださった指導教官の高橋信雄教授に感謝いたします．

プログラムの制作にあたり，日頃から丁寧に相談に応じ，数々の温かなご助言を頂いた,京都大学工学部情報学科 4 回生の坂井優斗氏に感謝致します．

7. 参考文献

[1] 三鴨 道弘，2014，“視覚特性を考慮したハイダイナミックレンジ画像の表示手法に関する研究 (A Study on Displaying High Dynamic Range Images Considering Human Visual Perception)”，広島大学 学術情報リポジトリ，23-24，(https://ir.lib.hiroshima-u.ac.jp/ja/00035950)，(2023-02-02)

[2] Autodesk Maya，“イメージベースドライティング(空のような照明)”，(http://me.autodesk.jp/wam/maya/docs/Maya2010/index.html?url=Glossary_I_imagebased_lighting_IBL.htm,topicNumber=d0e182207)，(2023-02-02)

[3] Paul E. Debevec, Jitendra Malik, 2008，“Recovering High Dynamic Range Radiance Maps from Photographs”，ACM SIGGRAPH 2008 classes，1-10，(https://dl.acm.org/doi/pdf/10.1145/258734.258884)，(2023-02-02)

[4] 松岡 諒，2016，“画像復元のための多重露光画像の統合”，RUKSOR 北九州市立大学 学術情報総合リポジトリ，12-14，(https://core.ac.uk/download/pdf/298622158.pdf)，(2023-02-02)

[5] OpenCV，“High Dynamic Range(HDR)”，(https://docs.opencv.org/3.4/d2/df0/tutorial_py_hdr.html)，(2023-02-02)

[6] OpenCV, “cv::CalibrateCRF Class Reference”, (https://docs.opencv.org/3.4/dd/de3/classcv_1_1CalibrateCRF.html), (2023-02-02)

[7] 春山 鉄源, “Python で学ぶ入門計量経済学”, (https://py4etrics.github.io/5_SciPy_stats.html), (2023-02-02)

[8] github, “OpenCV”, (https://github.com/opencv), (2023-02-02)

[9] @IT atmarkIT, “［損失関数／評価関数］平均二乗誤差（MSE：Mean Squared Error）／RMSE（MSE の平方根）とは？”, (https://atmarkit.itmedia.co.jp/ait/articles/2105/24/news019.html), (2023-02-02)