**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ТВ**

отчет

**по лабораторной работе № 1**

**по дисциплине «*Мультимедийная техника и технология производства аудиовизуальных программ*»**

Тема: **HDR**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 4105 |  | Суханов Р. А. |
|  |  | Гусев Д. В. |
| Преподаватель |  | Мотыко А. А. |

Санкт-Петербург

2019

**1 Цель работы.**

Получить HDR изображение на основе изображений с разной экспозицией.

**2. Краткие теоретические сведения:**

HDR — расширенный динамический диапазон, т е больший диапазон между самым темным и самым ярким полутонами.

Алгоритм представляет из себя гамма преобразование над изображениями с разной экспозицией, затем нормализацией пикселей и усреднение.

В результате должно получится изображение на котором более различимы детали, находящиеся в пересвеченной или затемненной областях на изображениях перед обработкой.

**3. Результаты работы:**

Входные изображения с разной экспозицией:







Результат ручной обработки:



Результат с вычитанием 10%



Результат с использованием библиотек opencv:



**4. Листинг программы:**

#include <vector>

#include <queue>

#include <math.h>

#include <iostream>

#include <opencv2/core.hpp>

#include <opencv2/imgproc.hpp>

#include <opencv2/highgui.hpp>

#include <opencv2/photo.hpp>

*using* *namespace* std;

*using* *namespace* cv;

void **minTenPersent**( Mat &src )

{

cvtColor( src, src, *COLOR\_BGR2HSV* );

Mat channels[3];

split( src, channels );

double min\_V = 0, max\_V = 0;

minMaxLoc( channels[2], &min\_V, &max\_V );

cout << "Max\_V: " << max\_V << endl;

double ten\_percent = double( max\_V \* 0.1 );

cout << "10%: " << ten\_percent << endl;

*for* ( int i = 0; i < int( src.total() ); i++ )

{

int temp = int( src.at< Vec3b >(i).val[2] - ten\_percent );

*if* ( temp > 0 )

src.at< Vec3b >(i).val[2] = uchar( src.at< Vec3b >(i).val[2] - ten\_percent );

}

cvtColor( src, src, *COLOR\_HSV2BGR* );

}

int **main**()

{

Mat img\_0 = imread( "HDR\_0.jpg" );

Mat img\_m2 = imread( "HDR\_-2.jpg" );

Mat img\_p2 = imread( "HDR\_+2.jpg" );

*//* *Gamma* *correction*

Mat img\_0\_Gamma = img\_0.clone();

Mat img\_m2\_Gamma = img\_m2.clone();

Mat img\_p2\_Gamma = img\_p2.clone();

float Gamma = 2.2f;

*for* ( int x = 0; x < img\_0.cols; x ++ )

{

*for* ( int y = 0; y < img\_0.rows; y ++ )

{

img\_0\_Gamma.at< Vec3b >(y, x).val[0] = uchar( pow( float(img\_0.at< Vec3b >(y, x).val[0]) / 255, Gamma ) \* 255 );

img\_0\_Gamma.at< Vec3b >(y, x).val[1] = uchar( pow( float(img\_0.at< Vec3b >(y, x).val[1]) / 255, Gamma ) \* 255 );

img\_0\_Gamma.at< Vec3b >(y, x).val[2] = uchar( pow( float(img\_0.at< Vec3b >(y, x).val[2]) / 255, Gamma ) \* 255 );

img\_m2\_Gamma.at< Vec3b >(y, x).val[0] = uchar( pow( float(img\_m2.at< Vec3b >(y, x).val[0]) / 255, Gamma ) \* 255 );

img\_m2\_Gamma.at< Vec3b >(y, x).val[1] = uchar( pow( float(img\_m2.at< Vec3b >(y, x).val[1]) / 255, Gamma ) \* 255 );

img\_m2\_Gamma.at< Vec3b >(y, x).val[2] = uchar( pow( float(img\_m2.at< Vec3b >(y, x).val[2]) / 255, Gamma ) \* 255 );

img\_p2\_Gamma.at< Vec3b >(y, x).val[0] = uchar( pow( float(img\_p2.at< Vec3b >(y, x).val[0]) / 255, Gamma ) \* 255 );

img\_p2\_Gamma.at< Vec3b >(y, x).val[1] = uchar( pow( float(img\_p2.at< Vec3b >(y, x).val[1]) / 255, Gamma ) \* 255 );

img\_p2\_Gamma.at< Vec3b >(y, x).val[2] = uchar( pow( float(img\_p2.at< Vec3b >(y, x).val[2]) / 255, Gamma ) \* 255 );

}

}

imshow( "img\_0\_Gamma", img\_0\_Gamma );

imshow( "img\_-2\_Gamma", img\_m2\_Gamma );

imshow( "img\_+2\_Gamma", img\_p2\_Gamma );

*//* *-10%*

*//* *minTenPersent(* *img\_0\_Gamma* *);*

*//* *minTenPersent(* *img\_m2\_Gamma* *);*

*//* *minTenPersent(* *img\_p2\_Gamma* *);*

*//* *Nornalizetion*

Mat img\_0\_Norm = Mat::zeros( img\_0\_Gamma.size(), CV\_32FC3 );

Mat img\_m2\_Norm = Mat::zeros( img\_m2\_Gamma.size(), CV\_32FC3 );

Mat img\_p2\_Norm = Mat::zeros( img\_p2\_Gamma.size(), CV\_32FC3 );

*for* ( int x = 0; x < img\_0.cols; x ++ )

{

*for* ( int y = 0; y < img\_0.rows; y ++ )

{

img\_0\_Norm.at< Vec3f >(y, x).val[0] = img\_0\_Gamma.at< Vec3b >(y, x).val[0] / 255.0f;

img\_0\_Norm.at< Vec3f >(y, x).val[1] = img\_0\_Gamma.at< Vec3b >(y, x).val[1] / 255.0f;

img\_0\_Norm.at< Vec3f >(y, x).val[2] = img\_0\_Gamma.at< Vec3b >(y, x).val[2] / 255.0f;

img\_m2\_Norm.at< Vec3f >(y, x).val[0] = img\_m2\_Gamma.at< Vec3b >(y, x).val[0] / 255.0f;

img\_m2\_Norm.at< Vec3f >(y, x).val[1] = img\_m2\_Gamma.at< Vec3b >(y, x).val[1] / 255.0f;

img\_m2\_Norm.at< Vec3f >(y, x).val[2] = img\_m2\_Gamma.at< Vec3b >(y, x).val[2] / 255.0f;

img\_p2\_Norm.at< Vec3f >(y, x).val[0] = img\_p2\_Gamma.at< Vec3b >(y, x).val[0] / 255.0f;

img\_p2\_Norm.at< Vec3f >(y, x).val[1] = img\_p2\_Gamma.at< Vec3b >(y, x).val[1] / 255.0f;

img\_p2\_Norm.at< Vec3f >(y, x).val[2] = img\_p2\_Gamma.at< Vec3b >(y, x).val[2] / 255.0f;

}

}

*//* *Averaging*

Mat img\_Resault = Mat::zeros( img\_0.size(), CV\_8UC3 ); *//* *CV\_8UC3*

*for* ( int x = 0; x < img\_0.cols; x ++ )

{

*for* ( int y = 0; y < img\_0.rows; y ++ )

{

img\_Resault.at< Vec3b >(y, x).val[0] = uchar( ( img\_0\_Norm.at< Vec3f >(y, x).val[0] +

img\_m2\_Norm.at< Vec3f >(y, x).val[0] +

img\_p2\_Norm.at< Vec3f >(y, x).val[0] ) / 3.0f \* 255.0f);

img\_Resault.at< Vec3b >(y, x).val[1] = uchar( ( img\_0\_Norm.at< Vec3f >(y, x).val[1] +

img\_m2\_Norm.at< Vec3f >(y, x).val[1] +

img\_p2\_Norm.at< Vec3f >(y, x).val[1] ) / 3.0f \* 255.0f);

img\_Resault.at< Vec3b >(y, x).val[2] = uchar( ( img\_0\_Norm.at< Vec3f >(y, x).val[2] +

img\_m2\_Norm.at< Vec3f >(y, x).val[2] +

img\_p2\_Norm.at< Vec3f >(y, x).val[2] ) / 3.0f \* 255.0f);

}

}

imshow( "img\_Resault", img\_Resault );

imwrite( "img\_Resault.png", img\_Resault );

vector< Mat > images;

images.push\_back( img\_0 );

images.push\_back( img\_m2 );

images.push\_back( img\_p2 );

vector< float > times;

float center = 4.2f;

float epsilo = 3.f;

times.push\_back( center );

times.push\_back( center - epsilo );

times.push\_back( center + epsilo );

Mat responseDebevec;

Ptr< CalibrateDebevec > calibrateDebevec = createCalibrateDebevec();

calibrateDebevec->*process*( images, responseDebevec, times );

Mat hdrDebevec;

Ptr< MergeDebevec > mergeDebevec = createMergeDebevec();

mergeDebevec->*process*( images, hdrDebevec, times, responseDebevec );

imshow( "HDR", hdrDebevec );

imwrite( "img\_Resault\_HDR.png", hdrDebevec \* 255 );

waitKey(0);

*return* 0;

}