```
#include "stdafx.h"
#include "funtions.h"

#define TEST_RGB 0; //0，使用灰度图进行演示 1.使用RGB图进行演示
#define GAMMA_FACTOR 0.5; //gamma矫正的gamma值


int gammaMain()
{

cout << "gamma correction demo" << endl;

int readType = TEST_RGB;
cv::Mat srcMat;

//检查参数是否合法
if (readType == 0) {
srcMat = imread("../testImages\\gtest.jpg", 0);
}
else if (readType == 1) {
srcMat = imread("../testImages\\gtest.jpg");
}
else {
cout << "parameter erroe!" << endl;
return -1;
}

if (srcMat.empty()) {
cout << "fail to read pic!" << endl;
return -1;
}

cv::Mat dstMat;
float gamma = GAMMA_FACTOR;

if (srcMat.type() == CV_8UC1) {

gammaCorrection(srcMat, dstMat, gamma);

}
else if (srcMat.type() == CV_8UC3) {
Mat channel[3];
Mat out[3];
float hist[3][256];

//通道分离
split(srcMat, channel);

for (int i = 0; i < 3; i++) {

gammaCorrection(channel[i], out[i], gamma);
}

merge(out, 3, dstMat);

}


imshow("src",srcMat);
imshow("dst",dstMat);
waitKey(0);

destroyAllWindows();

return 0;
}

int equalizeMain()
{

cout << "Histogram equalization demo" << endl;

int readType = TEST_RGB;
cv::Mat srcMat;

//检查参数是否合法
if (readType == 0) {
srcMat = imread("../testImages\\gtest.jpg", 0);
}
else if (readType == 1) {
srcMat = imread("../testImages\\gtest.jpg");
```

```
}
else {
return -1;
}

if (srcMat.empty()) {
cout << "fail to read pic!" << endl;
return -1;
}
cv::Mat dstMat;
cv::Mat dstHistMat;
cv::Mat srcHistMat;
cv::Mat histMat[3];

float srcHist[256];
float dstHist[256];

int bin_width=2;
int bin_heigth = 100;

//0.输入图像，类型是 8位单通道
//1.输出图像，与输入同样尺寸同样类型
if (srcMat.type() == CV_8UC1) {

equalizeHist(srcMat, dstMat);

//计算并绘制直方图
calcIntenHist(dstMat, dstHist);
drawIntenHist(dstHistMat, dstHist, 3, 100);
imshow("dstMat", dstMat);
imshow("dstMat hist", dstHistMat);

calcIntenHist(srcMat, srcHist);
drawIntenHist(srcHistMat, srcHist, 3, 100);

imshow("srcMat hist", srcHistMat);
imshow("srcMat", srcMat);

}
else if (srcMat.type() == CV_8UC3) {
Mat channel[3];
Mat out[3];
float hist[3][256];

//通道分离
split(srcMat, channel);

for (int i = 0; i < 3; i++) {
equalizeHist(channel[i], out[i]);
calcIntenHist(out[i], hist[i]);
drawIntenHist(histMat[i], hist[i], bin_width, bin_heigth);

//按照channel编号命名窗口
stringstream ss;
ss << i;
string histWindow = "Hist of chanel "+ss.str();
string matWindow= "Image of chanel "+ss.str();

imshow(histWindow, histMat[i]);
imshow(matWindow, out[i]);

}

merge(out,3,dstMat);

//原图转换位灰度图，并计算灰度直方图
cv::Mat grayMat;
cv::Mat graydstMat;
cvtColor(srcMat,grayMat,CV_BGR2GRAY);
cvtColor(dstMat,graydstMat,CV_BGR2GRAY);

//计算并绘制直方图
calcIntenHist(graydstMat, dstHist);
drawIntenHist(dstHistMat, dstHist, 3, 100);
imshow("dstMat", dstMat);
imshow("dstMat hist", dstHistMat);

calcIntenHist(grayMat, srcHist);
drawIntenHist(srcHistMat, srcHist, 3, 100);

imshow("srcMat hist", srcHistMat);
imshow("srcMat", srcMat);

}


waitKey(0);

destroyAllWindows();

return 0;

}

int gammaCorrection(cv::Mat srcMat, cv::Mat & dstMat, float gamma)
{

//本函数只处理单通道图像
if (srcMat.channels()!=1)return -1;

//建立查询表
unsigned char lut[256];
for (int i = 0; i < 256; i++)
{
//saturate_cast，防止像素值溢出，如果值<0,则返回0，如果大于255，则返回255
lut[i] = saturate_cast<uchar>(pow((float)(i / 255.0f), gamma) * 255.0f);
}

srcMat.copyTo(dstMat);

MatIterator_<uchar> it, end;
for (it = dstMat.begin<uchar>(), end = dstMat.end<uchar>(); it != end; it++) {
*it = lut[(*it)];
}

return 0;
}

int drawIntenHist(cv::Mat & histMat, float * srcHist,int bin_width,int bin_heght)
{
histMat.create(bin_heght, 256 * bin_width, CV_8UC3);
histMat = Scalar(255,255,255);

float maxVal = *std::max_element(srcHist, srcHist + 256);

for (int i = 0; i < 256; i++) {
Rect binRect;
binRect.x = i*bin_width;
float height_i = (float)bin_heght*srcHist[i] / maxVal;
binRect.height = (int)height_i;
binRect.y = bin_heght- binRect.height;
binRect.width = bin_width;
rectangle(histMat, binRect, CV_RGB(255, 0, 0), -1);
}

return 0;
}

int calcIntenHist(const cv::Mat src, float * dstHist)
{

//输入必为单通道图
if (src.type() != CV_8UC1) {
return -1;
}

memset(dstHist, 0, sizeof(float) * 256);
int height = src.rows;
int width = src.cols;
//指针遍历
for (int k = 0; k < height; k++)
{
// 获取第k行的首地址
const uchar* inData = src.ptr<uchar>(k);
//处理每个像素
for (int i = 0; i < width; i++)
{
int gray = inData[i];
dstHist[gray]++;
}
}
```

```
//直方图归一化
float norm = height*width;
for (int n = 0; n < 256; n++) {
dstHist[n] = dstHist[n] / norm;
}


return 0;
}
```