Работа 1. Исследование гамма-коррекции

автор: Полевой Д.В. дата:

Задание

1. Сгенерировать серое тестовое изображение I_1 в виде прямоугольника размером 768x60 пикселя с плавным изменение пикселей от черного к белому, одна градация серого занимает 3 пикселя по горизонтали.
2. Применить к изображению I_1 гамма-коррекцию с коэффициентом из интервала 2.2-2.4 и получить изображение G_1 при помощи функци pow.
3. Применить к изображению I_1 гамма-коррекцию с коэффициентом из интервала 2.2-2.4 и получить изображение G_2 при помощи прямого обращения к пикселям.
4. Показать визуализацию результатов в виде одного изображения (сверху вниз I_1, G_1, G_2).
5. Сделать замер времени обработки изображений в п.2 и п.3, результаты отфиксировать в отчете.

Результаты

Рис. 1. Результаты работы программы (сверху вниз I_1, G_1, G_2)

Текст программы

```
#include <stdio.h>
#include <opencv2/opencv.hpp>
#include <ctime>
#include <cmath>
cv::Mat task1() {
  cv::Mat img(60, 768, CV_8UC1);
  // draw dummy image
  img = 0;
  cv::Rect2d rc = {0, 0, 3, 60 };
  double brightnes = 0;
  while(rc.x + rc.width <= img.cols){
    brightnes = double(rc.x)/(img.cols-rc.width) * 255;
    printf("%d - %f\n",int(rc.x/rc.width),brightnes);
    cv::rectangle(img, rc, { brightnes }, -1);
    rc.x += rc.width;
  }
    // save result
  cv::imwrite("lab01_1.png", img);
  cv::imshow("Display Image 1", img);
  return img;
}
cv::Mat task2(cv::Mat& INimg) {
  cv::Mat img = INimg.clone();
  img.convertTo(img, CV_32F);
  img = img/255;
  cv::Mat dst;

  std::srand(int(std::time(nullptr)));
```

```
  double power = 2.2 + double(std::rand()%2001)/10000;
  cv::pow(img,power,dst);

    // save result
  dst = dst*255;
  dst.convertTo(dst, CV_8UC1);
  cv::imwrite("lab01_2.png", dst);
  cv::imshow("Display Image 2", dst);
  return dst;
}
cv::Mat task3(cv::Mat& INimg) {
  cv::Mat img = INimg.clone();
  img.convertTo(img, CV_32F);
  img = img/255;

  std::srand(int(std::time(nullptr)));
  double power = 2.2 + double(std::rand()%2001)/10000;
  for(int y=0;y<img.rows;y++)
  {
      for(int x=0;x<img.cols;x++)
      {
          img.at<float>(y,x) = std::pow(img.at<float>(y,x),power);
      }
  }

    // save result
  img = img*255;
  img.convertTo(img, CV_8UC1);
  cv::imwrite("lab01_3.png", img);
  cv::imshow("Display Image 3", img);
  return img;
}
cv::Mat task4(cv::Mat& img1,cv::Mat& img2,cv::Mat& img3) {
  cv::Mat img(img1.rows+img2.rows+img3.rows, img1.cols, CV_8UC1);
  img1.copyTo(img(cv::Rect(0, 0, img1.cols, img1.rows)));
  img2.copyTo(img(cv::Rect(0, img1.rows, img2.cols, img2.rows)));
  img3.copyTo(img(cv::Rect(0, img1.rows+img2.rows, img3.cols, img3.rows)));
  cv::imwrite("lab01_4.png", img);
  cv::imshow("Display Image 4", img);
  return img;
}
int main() {
  cv::Mat img1 = task1();
  clock_t start = std::clock();
  cv::Mat img2 = task2(img1);
  clock_t end1 = std::clock();
  cv::Mat img3 = task3(img1);
  clock_t end2 = std::clock();
  cv::Mat img4 = task4(img1,img2,img3);
  printf("\n%f\n%f",double(end1-start)/CLOCKS_PER_SEC,double(end2-
end1)/CLOCKS_PER_SEC);
  cv::waitKey(0);
}
```