

Работа 1. Исследование гамма-коррекции

автор: Котелевский А.А. дата: 2022-02-16T18:53:43

url: <https://github.com/artemiskot/Methods-and-means-of-image-processing/tree/main/prj.labs/lab01>

Задание

1. Сгенерировать серое тестовое изображение I_1 в виде прямоугольника размером 768x60 пикселя с плавным изменением 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 функции `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 <iostream>
#include <opencv2/opencv.hpp>
const double gamma_ = 2.3;
cv::Mat gammaPow(cv::Mat image) {
    image.convertTo(image, CV_64F, 1 / 255.0);
    pow(image, gamma_, image);
    image.convertTo(image, CV_8UC1, 255.0);
    return image;
}
cv::Mat gammaPixel(cv::Mat image) {
    cv::Mat image_float;
    image.convertTo(image_float, CV_64F);
    for (int i = 0; i < image_float.rows; ++i)
        for (int j = 0; j < image_float.cols; ++j) {
            image_float.at<double>(i, j) = (pow(image_float.at<double>(i, j) /
                255.0, gamma_) * 255.0);
        }
}
```

```

        image_float.convertTo(image_float, CV_8UC1);
        return image_float;
    }

    int main() {
        int64 t0 = cv::getTickCount();
        cv::Mat image(cv::Mat::zeros(60, 768, CV_8UC1));
        int thickness = 3;
        int lineType = cv::LINE_8;
        for (int i(0); i < image.cols; ++i) {
            cv::line(image,
                cv::Point(i, 255),
                cv::Point(i, 0),
                cv::Scalar(i / 3, i / 3, i / 3),
                thickness,
                lineType);
        }
        //Task 1
        double t1 = (double)cv::getTickCount();
        cv::Mat image_pow = gammaPow(image);
        //Task 2
        double t2 = (double)cv::getTickCount();
        cv::Mat image_pix = gammaPixel(image);
        //Task 3
        double t3 = (double)cv::getTickCount();
        double r1 = 1000 * ((t1 - t0) / cv::getTickFrequency());
        //Calculating time endcv::Point for 1st image part
        double r2 = 1000 * ((t2 - t1) / cv::getTickFrequency());
        //Calculating time endcv::Point for 2nd image part
        double r3 = 1000 * ((t3 - t2) / cv::getTickFrequency());
        //Calculating time endcv::Point for 3d image part

        int h = image.rows;
        int l = image.cols;
        cv::Mat result = cv::Mat::zeros(h * 3, l, CV_8UC1);
        image.copyTo(result(cv::Rect(0, h * 0, l, h)));
        image_pow.copyTo(result(cv::Rect(0, h * 1, l, h)));
        image_pix.copyTo(result(cv::Rect(0, h * 2, l, h)));
        imshow("Result picture picture", result);
        std::cout << "Gradient time: " << std::setprecision(2) << r1 << " ms \n"
            << "gamma_pow time: " << std::setprecision(2) << r2 << " ms \n"
            << "gamma_pixel time: " << std::setprecision(2) << r3 << " ms " <<
        std::endl;
        imwrite("lab01.png", result);
        cv::Mat img = cv::imread("lab01.png");
        cv::waitKey(0);
    }

```