Работа 1. Исследование гамма-коррекции

автор: Мартынов Д.А. дата: 07.05.2020

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.
3. Сгенерировать серое тестовое изображение I_2 в виде прямоугольника размером 768x60 пикселя со ступенчатым изменением яркости от черного к белому (от уровня 5 с шагом 10), одна градация серого занимает 30 пикселя по горизонтали.
4. Применить к изображению I_2 гамма-коррекцию с коэффициентом из интервала 2.2-2.4 и получить изображение G_2.
5. Показать визуализацию результатов в виде одного изображения, об

Результаты

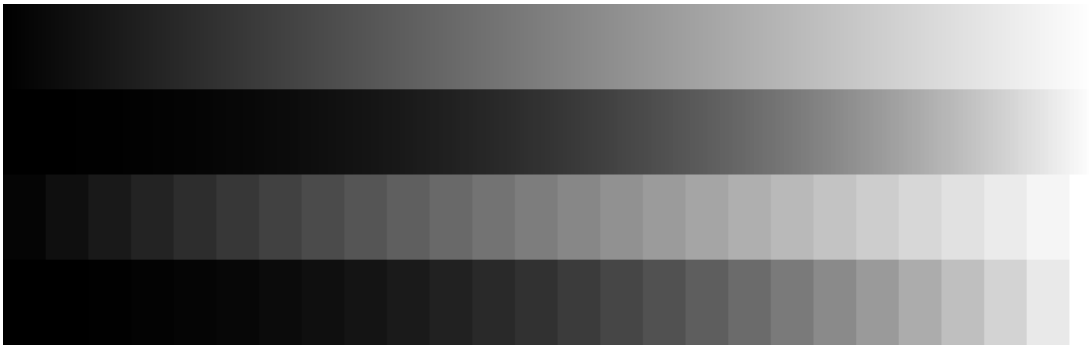

Рис. 1. Результаты работы программы (сверху вниз I_1, G_1, I_2, G_2)

Текст программы

```
#include <opencv2/opencv.hpp>
#include <vector>

void drawGrayScale(cv::Mat &img, int step, int brStep, int lvl, int width)
{
    for (int i{ 0 }; i < width; i += step)
    {
        // Рисуем оттенок серого в виде прямоугольника HEIGHTxSTEP
        cv::rectangle(img, cv::Point(i, 0), cv::Point((i + step), img.rows), cv::Scalar(lvl, lvl, lvl),
         CV_FILLED);
        lvl += brStep;
    }
}

cv::Mat gammaCorrection(const cv::Mat& img)
{
    cv::Mat G;

    // Преобразование в 64 битный float
    img.convertTo(G, CV_64F);

    // Нормализация и применение гамма-коррекции
    G /= 255;
    cv::pow(G, 2.3, G);
```

```
    G *= 255;

    // Конвертирование обратно в 8 битный uchar
    G.convertTo(G, CV_8UC1);

    return G;
}

int main()
{
    // Градация серого {0..255}
    int lvl{ 0 };
    int lvl2{ 5 };
    // Шаг с которым меняется градация серого
    int brStep{ 1 };
    int brStep2{ 10 };
    // Сколько в пикселях занимает одна градация серого
    int step{ 3 };
    int step2{ 30 };

    // Размеры окна в пикселях (768x60)
    int width{ 256 * step };
    int height{ 60 };

    cv::Mat img(height, width, CV_8UC1);
    cv::Mat img2(height, width, CV_8UC1);

    drawGrayScale(img, step, brStep, lvl, width);
    drawGrayScale(img2, step2, brStep2, lvl2, width);

    cv::Mat G1 = gammaCorrection(img);
    cv::Mat G2 = gammaCorrection(img2);

    // Соединение всех картинок в одну ----------------------------------------------

    cv::Mat allImages(height * 4, width, CV_8UC1);

    // Выделение областей окна под мозаику
    std::vector<cv::Mat>split;
    for (int i{ 0 }; i < 4; ++i)
    {
        split.push_back(cv::Mat(allImages, cv::Rect(0, i * height, width, height)));
    }

    // Вектор из наших картинок
    std::vector<cv::Mat> temp;
    temp.push_back(img);
    temp.push_back(G1);
    temp.push_back(img2);
    temp.push_back(G2);

    // Копирование картинок в выделенные области
    for (int i{ 0 }; i < 4; ++i)
    {
        temp[i].copyTo(split[i]);
    }

    // ------------------------------------------------------------------------------
    std::string writeTo{ "C:/Users/dimam/Documents/Admin/Programming/C++/OpenCV/OpenCVData/test_data/" };

    cv::imwrite(writeTo + "All images.png", allImages, { CV_IMWRITE_PNG_COMPRESSION, 0 });
}
```