Министерство образования и науки Российской Федерации
федеральное государственное автономное образовательное
учреждение высшего образования
Санкт-Петербургский исследовательский университет
Информационных технологий, механики и оптики
Факультет информационных технологий и программирования
Дисциплина: компьютерная геометрия и графика

Отчет

по лабораторной работе №3

Изучение алгоритмов псевдотонирования изображений

Выполнила: студент гр. М3102
Карпов Арсений В.
Преподаватель: Скаков П.С.

Санкт-Петербург
2020

Цель работы: изучить алгоритмы и реализовать программу, применяющий алгоритм дизеринга к изображению в формате PGM (P5) с учетом гамма-коррекции.

Описание работы

Программа должна быть написана на C/C++ и не использовать внешние библиотеки.

Аргументы передаются через командную строку:
program.exe <имя_входного_файла> <имя_выходного_файла> <градиент> <дизеринг> <битность> <гамма>
где

- <имя_входного_файла>, <имя_выходного_файла>: формат файлов: PGM P5; ширина и высота берутся из <имя_входного_файла>;
- <градиент>: 0 - используем входную картинку, 1 - рисуем горизонтальный градиент (0-255) (ширина и высота берутся из <имя_входного_файла>);
- <дизеринг> - алгоритм дизеринга:
  - 0 – Нет дизеринга;
  - 1 – Ordered (8x8);
  - 2 – Random;
  - 3 – Floyd–Steinberg;
  - 4 – Jarvis, Judice, Ninke;
  - 5 - Sierra (Sierra-3);
  - 6 - Atkinson;
  - 7 - Halftone (4x4, orthogonal);
- <битность> - битность результата дизеринга (1..8);
- <гамма>: 0 - sRGB гамма, иначе - обычная гамма с указанным значением.

Частичное решение:

- <градиент> = 1;
- <дизеринг> = 0..3;
- <битность> = 1..8;
- <гамма> = 1 (аналогично отсутствию гамма-коррекции)

+ корректно выделяется и освобождается память, закрываются файлы, есть обработка ошибок.
Полное решение: все остальное

Если программе передано значение, которое не поддерживается – следует сообщить об ошибке.

Коды возврата:
0 - ошибок нет
1 - произошла ошибка

В поток вывода ничего не выводится (printf, cout).
Сообщения об ошибках выводятся в поток вывода ошибок:
C: fprintf(stderr, "Error\n");
C++: std::cerr

Следующие параметры гарантировано не будут выходить за обусловленные значения:

- <градиент> = 0 или 1;
- <битность> = 1..8;
- width и height в файле - положительные целые значения;
- яркостных данных в файле ровно width * height;
- <гамма> - вещественная неотрицательная;

Теоретическая часть

Дизеринг (англ. *dither*), псевдотонирование — при обработке цифровых сигналов представляет собой подмешивание в первичный сигнал псевдослучайного шума со специально подобранным спектром. Применяется при обработке цифрового звука, видео и графической информации для уменьшения негативного эффекта от квантования.

Определение пороговых цветов для битностей

Но перед описанием самих алгоритм определим, как именно будут определяться наши пороговые (threshold) цвета для наших битностей: для округления текущего значения цвета до ближайшего, который можно отобразить в задаваемой битности **B**, из целочисленного значения цвета берутся **B** старших бит и дублируются сдвигами по **B** бит в текущее значение цвета.

Порядок бит в значении цвета:

7	6	5	4	3	2	1	0

Битность **B**	Маска изменения порядка бит
8	7 6 5 4 3 2 1 0
5	7 6 5 4 3 7 6 5
2	7 6 7 6 7 6 7 6
1	7 7 7 7 7 7 7 7

В данной лабораторной работе необходимо было реализовать 7 видов дизеринга:

1. **Ordered dithering.** Данный алгоритм уменьшает количество цветов, применяя карту порогов к отображаемым пикселям, в результате чего некоторые пиксели меняют цвет в зависимости от расстояния исходного цвета от доступных записей цветов в уменьшенной палитре.

Пороговые карты **M** бывают разных размеров:

$$\frac{1}{4} \times \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}$$

$$\frac{1}{16} \times \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix}$$

$$\frac{1}{64} \times \begin{bmatrix} 0 & 48 & 12 & 60 & 3 & 51 & 15 & 63 \\ 32 & 16 & 44 & 28 & 35 & 19 & 47 & 31 \\ 8 & 56 & 4 & 52 & 11 & 59 & 7 & 55 \\ 40 & 24 & 36 & 20 & 43 & 27 & 39 & 23 \\ 2 & 50 & 14 & 62 & 1 & 49 & 13 & 61 \\ 34 & 18 & 46 & 30 & 33 & 17 & 45 & 29 \\ 10 & 58 & 6 & 54 & 9 & 57 & 5 & 53 \\ 42 & 26 & 38 & 22 & 41 & 25 & 37 & 21 \end{bmatrix}$$

Каждый элемент матрицы дополнительно преобразуется по формуле
M[i][j] = (M[i][j] + 1) / 64 – 0.5
Тогда цвет каждого пикселя рассчитывается так:
$color' = FindNearestColor(color + 255 * M[x\%8][y\%8])$,

где color’ – новый цвет пикселя, color – старый цвет пикселя, FindNearestColor – функция, возвращающая ближайший цвет к данному в новой палитре.

2. **Random dithering.** Данный алгоритм аналогичен предыдущему, только вместо элемента матрицы, к старому цвету прибавляется рандомное число в диапазоне (0, 1].
3. **Floyd-Steinberg dithering.**

Первая и возможно самая известная формула рассеивания ошибок была опубликована Робертом Флойдом и Луисом Стейнбергом в 1976 году. Рассеивание ошибок происходит по следующей схеме:

- **x** - текущий пиксель, от которого распространяется ошибка
- **y,x** - строка/столбец изображения
- **ymx, xmx** - номер последние строки и столбца

y=0	x=0										xmx
y	x	7/16			x	7/16					x
	5/16	1/16		3/16	5/16	1/16				3/16	5/16
ymx											

4. **Jarvis, Judice, Ninke.**

   В год, когда Флойд и Стейнберг опубликовали свой знаменитый алгоритм дизеринга, был издан менее известный, но гораздо более мощный алгоритм. Фильтр Джарвиса, Джудиса и Нинке значительно сложнее, чем Флойда-Стейнберга:

$$\frac{1}{48}\begin{bmatrix} - & - & \# & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{bmatrix}$$

   При таком алгоритме ошибка распределяется на в три раза больше пикселей, чем у Флойда-Стейнберга, что приводит к более гладкому и более тонкому результату.

5. **Sierra-3.** Аналогичен вышеназванным алгоритмам, только с иной матрицей: $\begin{pmatrix} - & - & X & 5/32 & 3/32 \\ 2/32 & 4/32 & 5/32 & 4/32 & 2/32 \\ - & 2/32 & 3/32 & 2/32 & - \end{pmatrix}$.
6. **Atkinson.** Этот алгоритм отличается от предыдущих тем, что рассеивает не всю ошибку, а только ее часть, что позволяет уменьшить зернистость. $\begin{pmatrix} - & X & 1/8 & 1/8 \\ 1/8 & 1/8 & 1/8 & - \\ - & 1/8 & - & - \end{pmatrix}$.
7. **Halftone 4x4.** Полутонирование - создание изображения со многими уровнями серого или цвета (т.е. слитный тон) на аппарате с меньшим количеством тонов, обычно чёрно-белый принтер**.** В случае обработки цифрового изображения halftone представляет собой матрицы порогов (для различных углов поворота), позволяющие воспроизводить это “точки” как при печать изображения. Аналогичен Ordered dithering, только имеет другую матрицу порогов: $\begin{pmatrix} 7 & 13 & 11 & 4 \\ 12 & 16 & 14 & 8 \\ 10 & 15 & 6 & 2 \\ 5 & 9 & 3 & 1 \end{pmatrix}$.

   Каждый элемент матрицы преобразуется по формуле:
   M[i][j] = M[i][j] / 16 – 0.5

Экспериментальная часть

Язык программирования: C++/17.

Данные с картинки считываются в массив типа unsigned char.

После чтения, все цвета обрабатываются с помощью прямой гаммы, или же обратного sRGB. Если же ввели команду «градиент», то эта обработка не производится. Перед записью в файл все цвета обрабатываются обратной гаммой или прямым sRGB.

Также, сразу создается палитра на основе считанного значения битности. И при работе с алгоритмами, ближайший цвет находится именно с помощью этой палитры.

Далее идет запуск введенного алгоритма дизеринга.

Выводы

В ходе проделанной работы была получена программа, преобразующая изображение в различные битности а также применяющая к этим изображениям алгоритмы дизеринга. Картинки, обработанные с помощью алгоритмов дизеринга, выглядят намного красивее и качественнее, чем необработанные. Полученные результаты совпадают с теоретическими.

Листинг

main.cpp

```
#include <iostream>
#include <algorithm>
#include <cstdlib>
#include <vector>
#include <cmath>
#include <string>
typedef unsigned char uchar;
void gradient(uchar* p, int width, int height, int bpp, double gamma)
{
    double k = double(255)/ width;
    for (int i = 0; i < height; i++)
        for (int j = 0; j < width; j++)
        {
            *(p + i * width + j) = round(bpp * pow(double(j * k)/ bpp, double(1 / gamma)));
        }
}

class Exception
{
public:
    Exception(std::string error)
    {
        error_ = error;
    }
    std::string out_error()
```

```
    {
        return error_;
    }
private:
    std::string error_;
};

class dithering
{
private:
    static std::vector<std::vector<double>> matrix()
    {
        std::vector<std::vector<double>> v(8, std:: vector<double> (8));
        v[0] =  {1, 49,  13,  61,   4,  52,  16,  64};
        v[1] = {33,  17,  45,  29,  36,  20,  48,  32};
        v[2] = {9,  57,   5 , 53,  12,  60 ,  8,  56};
        v[3] = {41,  25,  37,  21,  44,  28,  40 , 24};
        v[4] = {3,  51,  15,  63,   2,  50 , 14 , 62};
        v[5] = {35,  19,  47,  31,  34,  18,  46 , 30};
        v[6] = {11,  59,   7,  55,  10,  58,   6 , 54};
        v[7] = {43,  27,  39,  23,  42,  26,  38 , 22};
        for (int i = 0; i < 8; i++)
            for (int j = 0; j < 8; j++)
                v[i][j] = double(v[i][j]) / 65 - 0.5;
        return v;
    }
    static std::vector<std::vector<double>> matrix2()
    {
        std::vector<std::vector<double>> v(4, std:: vector<double> (4));
        v[0] =  {7, 13, 11, 4};
        v[1] = {12, 16, 14, 8};
        v[2] = {10, 15, 6, 2};
        v[3] = {5, 9, 3, 1};
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                v[i][j] = double(v[i][j]) / 16 - 0.5;
        return v;
    }
    static std::vector<std::vector<double>> err1(int type, int width)
    {
```

```
    std::vector<std::vector<double>> err(type, std::vector<double> (width));
    for (int i = 0; i < type; i++)
        for (int j = 0; j < width; j++)
            err[i][j] = 0;
    return err;
}
static int closest_color(double k, int bit)
{
    int k1 = round(k);
    if (k1 >= 255)
        return 255;
    if (k1 < 0)
        return 0;
    int k2 = k1 >> (8 - bit);
    k1 = 0;
    for (int i = 0; i < 7/ bit + 1; i++)
        k1 = (k1 << bit) + k2;
    k1 = k1 >> ((7 / bit + 1) * bit - 8);
    return k1;
}
static void _err_floyd(std::vector<std::vector<double>> &er, int i, int j, double err, int width)
{
    if (j < width - 1)
    {
        er[i % 2][j + 1] += double(7 * err) / 16 ;
        er[(i + 1) % 2][j + 1] += double(err) / 16;
    }
    er[(i + 1) % 2][j] += double(5 * err) / 16;
    if (j > 0)
        er[(i + 1) % 2][j - 1] += double(3 * err) / 16;
}
static void _err_jjn(std::vector<std::vector<double>> &er, int i, int j, double err, int width)
{
    if (j < width - 1)
    {
        er[i % 3][j + 1] += double(7) / 48 * err;
        er[(i + 1) % 3][j + 1] += double(5) / 48 * err;
        er[(i + 2) % 3][j + 1] += double(3) / 48 * err;
    }
    if (j < width - 2)
```

```
    {
        er[i % 3][j + 2] += double(5) / 48 * err;
        er[(i + 1) % 3][j + 2] += double(3) / 48 * err;
        er[(i + 2) % 3][j + 2] += double(1) / 48 * err;
    }
    er[(i + 1) % 3][j] += double(7) / 48 * err;
    er[(i + 2) % 3][j] += double(5) / 48 * err;
    if (j > 0)
    {
        er[(i + 1) % 3][j - 1] += double(5) / 48 * err;
        er[(i + 2) % 3][j - 1] += double(3) / 48 * err;
    }
    if (j > 1)
    {
        er[(i + 1) % 3][j - 2] += double(3) / 48 * err;
        er[(i + 2) % 3][j - 2] += double(1) / 48 * err;
    }
}
static void _err_sierra(std::vector<std::vector<double>> &er, int i, int j, double err, int width)
{
    if (j < width - 1)
    {
        er[i % 3][j + 1] += double(5) / 32 * err;
        er[(i + 1) % 3][j + 1] += double(4) / 32 * err;
        er[(i + 2) % 3][j + 1] += double(2) / 32 * err;
    }
    if (j < width - 2)
    {
        er[i % 3][j + 2] += double(3) / 32 * err;
        er[(i + 1) % 3][j + 2] += double(2) / 32 * err;
    }
    er[(i + 1) % 3][j] += double(5) / 32 * err;
    er[(i + 2) % 3][j] += double(3) / 32 * err;
    if (j > 0)
    {
        er[(i + 1) % 3][j - 1] += double(4) / 32 * err;
        er[(i + 2) % 3][j - 1] += double(2) / 32 * err;
    }
    if (j > 1)
    {
```

```
            er[(i + 1) % 3][j - 2] += double(2) / 32 * err;
        }
    }
    static void _err_atk(std::vector<std::vector<double>> &er, int i, int j, double err, int width)
    {
        if (j < width - 1)
        {
            er[i % 3][j + 1] += double(1) / 8 * err;
            er[(i + 1) % 3][j + 1] += double(1) / 8 * err;
        }
        if (j < width - 2)
            er[i % 3][j + 2] += double(1) / 8 * err;
        er[(i + 1) % 3][j] += double(1) / 8 * err;
        er[(i + 2) % 3][j] += double(1) / 8 * err;
        if (j > 0)
            er[(i + 1) % 3][j - 1] += double(1) / 8 * err;
    }
    static int closest_color(double k, std::vector<std::vector<double>> &er, int bit, int i, int j, int width, int
type)
    {
        k += er[i % (std::min(3, type))][j];
        er[i % (std::min(3, type))][j] = 0;
        int k1 = round(k);
        if (k1 > 255)
            k1 =  255;
        else
        if (k1 < 0)
            k1 = 0;
        else
        {
            int k2 = k1 >> (8 - bit);
            k1 = 0;
            for (int i = 0; i < 7 / bit + 1; i++)
                k1 = (k1 << bit) + k2;
            k1 = k1 >> ((7 / bit + 1) * bit - 8);
        }
        double err = k - k1;
        if (type == 2)
        {
            _err_floyd(er, i, j, err, width);
```

```
        }
        if (type == 3)
        {
            _err_jjn(er, i, j, err, width);
        }
        if (type == 4)
        {
            _err_sierra(er, i, j, err, width);
        }
        if (type == 5)
        {
            _err_atk(er, i, j, err, width);
        }
        return k1;
    }
    std::vector<std::vector<double>> v;
public:
    dithering(int k, uchar* p, int width, int height, int bpp, double gamma, int bit)
    {
        switch (k)
        {
            case 0:
                _nxn(p, width, height, bpp, gamma, bit, 0);
                break;
            case 1:
                v = matrix();
                dithering::_nxn(p, width, height, bpp, gamma, bit, 8);
                break;
            case 2:
                dithering::rnd(p, width, height, bpp, gamma, bit);
                break;
            case 3:
                error(p, width, height, bpp, gamma, bit, 2);
                break;
            case 4:
                error(p, width, height, bpp, gamma, bit, 3);
                break;
            case 5:
                error(p, width, height, bpp, gamma, bit, 4);
                break;
```

```
        case 6:
            error(p, width, height, bpp, gamma, bit, 5);
            break;
        case 7:
            v = matrix2();
            dithering::_nxn(p, width, height, bpp, gamma, bit, 4);
            break;
        default:
            break;
    }
}
void _nxn(uchar* p, int width, int height, int bpp, double gamma, int bit, int n)
{
    for (int i = 0; i < height; i++)
        for (int j = 0; j < width; j++)
        {
            *(p + i * width + j) = bpp * pow(double(*(p + i * width + j)) / bpp, gamma);
            if (n == 0)
                *(p + i * width + j) = closest_color(*(p + i * width + j), bit);
            else
                *(p + i * width + j) = closest_color(*(p + i * width + j) +  bpp * (dithering::v[i % n][j % n]), bit);
            *(p + i * width + j) = pow((double(*(p + i * width + j)) / bpp), double(1) / gamma) * bpp;
        }
}
static void rnd(uchar* p, int width, int height, int bpp, double gamma, int bit)
{
    random();
    for (int i = 0; i < height; i++)
        for (int j = 0; j < width; j++)
        {
            *(p + i * width + j) = bpp * pow(double(*(p + i * width + j)) / bpp, gamma);
            *(p + i * width + j) = closest_color(*(p + i * width + j) +pow(-1, j) * bpp * double(random() % 50) /
100, bit);
            *(p + i * width + j) = pow((double(*(p + i * width + j)) / bpp), double(1) / gamma) * bpp;
        }
}
static void error(uchar* p, int width, int height, int bpp, double gamma, int bit, int type)
{
    std::vector<std::vector<double>> er = err1(std::min(type, 3) ,width);
    for (int i = 0; i < height; i++)
```

```
        {
            for (int j = 0; j < width; j++)
            {
                *(p + i * width + j) = bpp * pow(double(*(p + i * width + j)) / bpp, gamma);
                *(p + i * width + j) = closest_color(*(p + i * width + j), er, bit, i, j, width, type);
                *(p + i * width + j) = pow((double(*(p + i * width + j)) / bpp), double(1) / gamma) * bpp;
            }
        }
    }
};
int main(int argc, char* argv[])
{
    try{
        if ((argc > 7) || (argc < 7))
            throw Exception("Wrong arguments");
        FILE* fin = fopen(argv[1], "rb");
        if (fin == nullptr)
            throw Exception("Can't open inputfile");
        int width = 256;
        int height = 256;
        int bpp = 255;
        int type = 0;
        int i = fscanf(fin, "P %d", &type);
        if ((i != 1) || (type != 5))
        {
            fclose(fin);
            throw Exception("Incorrect file type");
        }
        i = fscanf(fin, "\n%d %d\n%d\n", &width, &height, &bpp);
        if (i != 3)
        {
            fclose(fin);
            throw Exception("Incorrect file type");
        }
        if ((width <=0) || (height <= 0) || (bpp != 255))
        {
            fclose(fin);
            throw Exception("Incorrect file type");
        }
        double gamma;
```

```
        int bit;
        uchar *p = (uchar*)malloc(width*height*sizeof(uchar));
        i = fread(p, sizeof(uchar), width * height, fin);
        if (i != width*height)
        {
            fclose(fin);
            throw Exception("Incorrect file type");
        }
        FILE* fout = fopen(argv[2], "wb");
        if (fout == nullptr)
        {
            fclose(fin);
            throw Exception("Can't open outputfile");
        }
        int dith = std::stoi(argv[4]);
        if ((dith < 0) || (dith > 7))
        {
            fclose(fin);
            fclose(fout);
            throw Exception("No such dithering");
        }
        bit = std::stoi(argv[5]);
        gamma = std::stod(argv[6]);
        if (gamma == 0)
            gamma = 2.2;
        if(std::stoi(argv[3]) == 1)
            gradient(p, width, height, bpp, gamma);
        dithering A(dith, p, width, height, bpp, gamma, bit);
        fprintf(fout, "P5\n%d %d\n%d\n", width, height, bpp);
        fwrite(p, sizeof(uchar), width*height, fout);
        fclose(fin);
        fclose(fout);
    }
    catch (Exception &ex)
    {
        std::cerr << ex.out_error() << std::endl;
        return 1;
    }
    return 0;
}
```