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. М3101
Шмарина Л.С.
Преподаватель: Скаков П.С.

Санкт-Петербург
2020

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

Описание работы

Описание:
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

Примеры преобразования 2 (random) при разных значениях гаммы и битностей: dither_random

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.

При оптимизации изображений путём уменьшения количества цветов, применение дизеринга приводит к визуальному улучшению изображения

(однако для отдельных сжатых форматов (например, PNG), увеличивает его размер). Дизеринг также находит применение в веб-дизайне, где этот полезный метод используется для сокращения числа цветов изображения, что уменьшает размер файла (и трафик) без ущерба для качества. Он также используется при уменьшении цифровых фотографий в формате RAW в 48 или 64 бита на пиксель до RGB в 24 бита на пиксель для редактирования.

Однако уменьшение количества цветов практически всегда приводит к появлению специфических эффектов. Обычные фотографии могут иметь тысячи и даже миллионы различающихся цветов и оттенков, и преобразование их в индексированный формат с фиксированной палитрой приводит к потере огромного количества информации о цвете.

Алгоритмы дизеринга, использующиеся в лабораторной работе:

1. **Нет дизеринга (no dithering)**

Данный метод подразумевает лишь округление всех цветов до пороговых.

Алгоритмы с упорядоченным распределением ошибки (Ordered):

2. **Ordered (8x8)**

Алгоритм уменьшает количество цветов, применяя карту порогов M (другое обозначение: Bayer matrix) к отображаемым пикселям, в результате чего некоторые пиксели меняют цвет в зависимости от расстояния исходного цвета от доступных записей цветов в уменьшенной палитре.

Для каждого пикселя производится смещение его значения цвета на соответствующее значение из карты порогов M в соответствии с его местоположением, в результате чего значение пикселя квантуется на другой цвет, если оно превышает пороговое значение.

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

3. **Halftone (4x4, orthogonal)**

Halftone, 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, позволяющие воспроизводить “точки” как при печати изображения.

4. **Random**

Данный алгоритм можно также отнести к типу ordered с тем условием, что для определения добавки к значению текущего пикселя берется не элемент матрицы, а случайное число.

<u>Алгоритмы с рассеиванием ошибки (Error diffusion):</u>

5. **Floyd–Steinberg**

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

	X	**+7 * 0,6875**
+3 * 0,6875	**+5 * 0,6875**	**+1 * 0,6875**

Если наш пиксель округляется в большее значение (129 к 170 например), то ошибка будет отрицательной соответственно.

Путем распространения ошибки нескольким пикселям, каждому с различным значением, мы сводим к минимуму все отвлекающие полосы с точками, заметные в исходном примере алгоритма рассеивания ошибок. Вот изображение куба с применением алгоритма Флойда-Стейнберга:

Алгоритм даёт достаточно хорошее качество, а также требует только один передний массив (одномерный массив шириной в изображение, где хранятся значения ошибок, распространяемые к следующей строке). Кроме того, поскольку его делитель 16, вместо деления можно использовать битовые сдвиги. Так алгоритм достигает высокой скорости работы даже на старом оборудовании.

Что касается значений 1/3/5/7, используемых для распространения ошибки – они были выбраны специально, потому что они создают равномерный клетчатый узор для серого изображения.

6. 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:

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

Алгоритм Джарвиса, Джудиса и Нинке аналогичен алгоритму Флойда-Стейнберга с точностью до матрицы распределения ошибки. При данном алгоритме ошибка распределяется на в три раза больше пикселей, чем у Флойда-Стейнберга, что приводит к более гладкому и более тонкому результату.

7. **Sierra (Sierra-3)**

Аналогично алгоритму Флойда-Стейнберга, но с другой матрицей:

```
      X  5  3
2  4  5  4  2
   2  3  2
    (1/32)
```

8. **Atkinson**

Аналогично алгоритму Флойда-Стейнберга, но с другой матрицей:

```
0 X 1 1
1 1 1 0
0 1 0 0
  x 1/8
```

Экспериментальная часть

Работа над изображением производится следующим образом: считываем изображение, применяем гамма-коррекцию к каждому пикселю, после чего изображение обрабатывается алгоритмом дизеринга, а потом после гамма коррекции снова записывается в файл.

Выводы

В ходе работы был разработана программа, которая обрабатывает изображение любым из восьми алгоритмов дизеринга. Также в ходе выполнения работы я более глубоко ознакомилась с алгоритмом гамма-коррекции, а также применила его в работе, что позволило мне избежать значительных искажений изображения вследствие нелинейности человеческого восприятия цвета.

Листинг

main.cpp

```
#include <iostream>
#include <cstdlib>
#include <set>
#include "NetPBM.h"

using namespace std;

int main(int argc, char **argv) {
   try {
       if (argc == 666) {
           throw logic_error("Invalid user");
       }
       if (argc != 7) {
           throw logic_error("Invalid argument count");
       }
       char *infilename = argv[1];
       char *outfilename = argv[2];
```

```
bool gradient = atoi(argv[3]);
int dithering = atoi(argv[4]);
int bitness = atoi(argv[5]);
double gamma = atof(argv[6]);
auto *file = new ifstream(infilename, std::ios::binary);
(*file).unsetf(ios_base::skipws);
if (!(*file).is_open()) {
    delete file;
    throw logic_error("Can't open file to read");
}

auto *netPcm = new NetPBM(file, gamma, gradient);
switch (dithering) {
    case 0:
        netPcm->no_dithering(bitness);
        break;

    case 1:
        netPcm->ordered_dithering(bitness);
        break;

    case 2:
        netPcm->random_dithering(gamma, bitness);
        break;

    case 3:
        netPcm->Floyd_Steinberg_dithering(bitness);
        break;
    case 4:
        netPcm->jjn_dithering(bitness);
        break;
```

```
            case 5:
                netPcm->sierra_dithering(bitness);
                break;
            case 6:
                netPcm->atikson_dithering(bitness);
                break;
            case 7:
                netPcm->halftone_dithering(bitness);
                break;

        }

        auto *outfile = new ofstream(outfilename,
std::ios::binary);
        if (!outfile->is_open()) {
            file->close();
            delete file;
            delete outfile;
            delete netPcm;
            throw logic_error("Can't open file to write");
        }

        netPcm->write_to_file(outfile, gamma);
        outfile->close();
        file->close();
        delete file;
        delete outfile;
        delete netPcm;
        return 0;
    }
    catch (const logic_error &error) {
```

```
        cerr << error.what() << endl;
        return 1;
    }
}
```

NetPBM.h

```
#include <iostream>
#include <fstream>
#include <cstdint>
#include <cstdlib>
#include <cmath>
#include <set>jndt

#ifndef LAB3_NETPBM_H
#define LAB3_NETPBM_H

using namespace std;
struct point_t {
    double x;
    double y;
};

class NetPBM {
private:
    ifstream *file;
    uint16_t type = -1;
    uint16_t width = -1;
    uint16_t height = -1;
    uint16_t depth = -1;
    unsigned char **array;

    void read_header() {
```

```
        unsigned char buf;
        *this->file >> buf;
        if (buf != 'P') {
            throw logic_error("Bad file");
        }
        *this->file >> buf;
        if (buf == '1' || buf == '2' || buf == '3' || buf == '4'
|| buf == '6' || buf == '7') {
            throw logic_error("Not supported type of NetPCM");
        } else if (buf == '5')
            this->type = 5;
        else {
            throw logic_error("Bad file");
        }
        *this->file >> buf;
        *this->file >> this->width;
        *this->file >> buf;
        *this->file >> this->height;
        *this->file >> buf;
        *this->file >> this->depth;
        if (this->depth != 255) {
            throw logic_error("Not supported non 255 colors
count");
        }
    }

    void read_data(double gamma_value, bool gradient) {
        for (int i = 0; i < this->height; i++) {
            for (int j = 0; j < this->width; j++) {
                unsigned char tmp;
                if (!gradient)
```

```
                    *this->file >> tmp;

                else {
                    tmp = (unsigned char) ((double) j * 256 /
this->width);
                }

                double value = ((double) tmp) / this->depth;
                if (gamma_value == 0) {
                    value = ungammasRGB(value);
                } else {
                    value = ungamma(value, gamma_value);
                }
                this->array[i][j] = (unsigned char) (value *
this->depth);
            }
        }
    }

//
https://en.wikipedia.org/wiki/SRGB#The_sRGB_transfer_function_(%
22gamma%22)
    static double gammasRGB(double u) {
        if (u <= 0.0031308) {
            return 323 * u / 25.;
        } else {
            return (211. * pow(u, 5. / 12.) - 11.) / 200.;
        }
    }

    static double ungammasRGB(double u) {
        if (u <= 0.04045) {
```

```
            return 25 * u / 323;
        } else {
            return pow((200. * u + 11.) / 211., 12. / 5.);
        }
    }

    static double set_gamma(double u, double gamma) {
        return pow(u, 1.0 / gamma);
    }

    static double ungamma(double u, double gamma) {
        return pow(u, gamma);
    }

public:
    int16_t getType() const {
        return type;
    }


    void write_to_file(ofstream *outfile, double gamma_value) {
        *outfile << "P5" << (unsigned char) (10) << width << " "
<< height << (unsigned char) (10) << depth
                 << (unsigned char) (10);
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                double value;
                double color_value = (double) array[i][j] /
this->depth;
                if (gamma_value == 0) {
                    value = gammasRGB(color_value);
```

```
                } else {
                    value = set_gamma(color_value, gamma_value);
                }
                if (value >= 1 - 1 / 1e9)
                    value = 1;
                *outfile << (unsigned char) round(this->depth *
value);
            }
        }
    }

    unsigned char findNearestPaletteColor(unsigned char color,
int bitness) {
        if (bitness != 8) {
            unsigned char tmp;
            tmp = color & (((1u << bitness) - 1) << (8 -
bitness));
            color = 0;
            for (unsigned i = 0; i < 8 / bitness + 1; ++i) {
                color = color | ((unsigned char) (tmp >> bitness
* i));
            }
            return color;
        }

    }

    void no_dithering(int bitness) {
        if (bitness != 8) {
            for (int i = 0; i < this->height; i++) {
                for (int j = 0; j < this->width; j++) {
```

```
                    unsigned char color =
findNearestPaletteColor(this->array[i][j], bitness);
                    this->array[i][j] = color;
                }
            }
        }

    }

    void ordered_dithering(int bitness) {
        if (bitness != 8) {
            double resizer = this->depth / (pow(2., (double)
bitness) - 1);
//          this->depth = (pow(2., (double) bitness)-1);
            double bayerMatrix[8][8] = {
                    {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}
            };

            for (auto &i : bayerMatrix) {
                for (double &j : i) {
                    j = j / 64 - 0.5;
                }
            }
            for (int i = 0; i < this->height; i++) {
```

```
                for (int j = 0; j < this->width; j++) {
                    int tmp = (int) (this->array[i][j] + resizer
* bayerMatrix[i % 8][j % 8]);
                    if (tmp > 255) {
                        tmp = 255;
                    }
                    if (tmp < 0) {
                        tmp = 0;
                    }
                    unsigned char color =
findNearestPaletteColor((unsigned char) tmp, bitness);
                    this->array[i][j] = color;
                }
            }
        }

    }

    void random_dithering(double gamma_value, int bitness) {
        if (bitness != 8) {
            double resizer = this->depth / (pow(2., (double)
bitness) - 1);
            for (int i = 0; i < this->height; i++) {
                for (int j = 0; j < this->width; j++) {
                    int tmp = (int) (this->array[i][j] + resizer
* (-50 + rand() % 100) / 100);
                    if (tmp > 255) {
                        tmp = 255;
                    }
                    if (tmp < 0) {
                        tmp = 0;
                    }
```

```
                    unsigned char color =
findNearestPaletteColor((unsigned char) tmp, bitness);
                    this->array[i][j] = color;
                }
            }
        }

    }

    void Floyd_Steinberg_dithering(int bitness) {
        if (bitness != 8) {
            auto **err = new double *[(int) this->height];
            for (int i = 0; i < this->height; i++) {
                err[i] = new double[(int) this->width];
                for (int j = 0; j < this->width; j++) {
                    err[i][j] = 0;
                }
            }
            for (int i = 0; i < this->height; i++)
                for (int j = 0; j < this->width; j++) {
                    double tmp = (double) this->array[i][j] +
err[i][j];
                    if (tmp > 255)
                        tmp = 255;
                    if (tmp < 0)
                        tmp = 0;
                    this->array[i][j] = (unsigned char) tmp;
                    unsigned char nc =
findNearestPaletteColor(this->array[i][j], bitness);
                    double error = (this->array[i][j] - nc) /
16.0;
                    this->array[i][j] = nc;
```

```
                    if (j + 1 < this->width)
                        err[i][j + 1] += error * 7;
                    if (i + 1 < this->height) {
                        if (j - 1 >= 0)
                            err[i + 1][j - 1] += error * 3;
                        err[i + 1][j] += error * 5;
                        if (j + 1 < this->width)
                            err[i + 1][j + 1] += error;
                    }
                }
            for (int i = 0; i < height; i++)
                delete[] err[i];
        }

    }

    void jjn_dithering(int bitness) {
        if (bitness != 8) {
            auto **err = new double *[(int) this->height];
            for (int i = 0; i < this->height; i++) {
                err[i] = new double[(int) this->width];
                for (int j = 0; j < this->width; j++) {
                    err[i][j] = 0;
                }
            }
            for (int i = 0; i < this->height; i++)
                for (int j = 0; j < this->width; j++) {
                    double tmp = (double) this->array[i][j] +
err[i][j];
                    if (tmp > 255)
                        tmp = 255;
```

```
                    if (tmp < 0)
                        tmp = 0;
                    this->array[i][j] = (unsigned char) tmp;
                    unsigned char nc =
findNearestPaletteColor(this->array[i][j], bitness);
                    double error = (this->array[i][j] - nc) /
48.0;
                    this->array[i][j] = nc;
                    if (j + 1 < this->width) err[i][j + 1] +=
error * 7;
                    if (j + 2 < this->width) err[i][j + 2] +=
error * 5;
                    if (i + 1 < this->height) {
                        if (j - 2 >= 0) err[i + 1][j - 2] +=
error * 3;
                        if (j - 1 >= 0) err[i + 1][j - 1] +=
error * 5;
                        err[i + 1][j] += error * 7;
                        if (j + 1 < this->width) err[i + 1][j +
1] += (error * 5);
                        if (j + 2 < this->width) err[i + 1][j +
2] += (error * 3);
                    }
                    if (i + 2 < this->height) {
                        if (j - 2 >= 0) err[i + 2][j - 2] +=
(error * 1);
                        if (j - 1 >= 0) err[i + 2][j - 1] +=
(error * 3);
                        err[i + 2][j] += (error * 5);
                        if (j + 1 < this->width) err[i + 2][j +
1] += (error * 3);
                        if (j + 2 < this->width) err[i + 2][j +
2] += (error * 1);
                    }
                }
```

```
            for (int i = 0; i < height; i++)
                delete[] err[i];
        }

    }

    void sierra_dithering(int bitness) {
        if (bitness != 8) {
            auto **err = new double *[(int) this->height];
            for (int i = 0; i < this->height; i++) {
                err[i] = new double[(int) this->width];
                for (int j = 0; j < this->width; j++) {
                    err[i][j] = 0;
                }
            }
            for (int i = 0; i < this->height; i++)
                for (int j = 0; j < this->width; j++) {
                    double tmp = (double) this->array[i][j] +
err[i][j];
                    if (tmp > 255)
                        tmp = 255;
                    if (tmp < 0)
                        tmp = 0;
                    this->array[i][j] = (unsigned char) tmp;
                    unsigned char nc =
findNearestPaletteColor(this->array[i][j], bitness);
                    double error = (this->array[i][j] - nc) /
32.0;
                    this->array[i][j] = nc;
                    if (j + 1 < this->width) err[i][j + 1] +=
(error * 5);
```

```
                    if (j + 2 < this->width) err[i][j + 2] +=
(error * 3);
                    if (i + 1 < this->height) {
                        if (j - 2 >= 0) err[i + 1][j - 2] +=
(error * 2);
                        if (j - 1 >= 0) err[i + 1][j - 1] +=
(error * 4);
                        err[i + 1][j] += (error * 5);
                        if (j + 1 < this->width) err[i + 1][j +
1] += (error * 4);
                        if (j + 2 < this->width) err[i + 1][j +
2] += (error * 2);
                    }
                    if (i + 2 < this->height) {
                        if (j - 1 >= 0) err[i + 2][j - 1] +=
(error * 2);
                        err[i + 2][j] += (error * 3);
                        if (j + 1 < this->width) err[i + 2][j +
1] += (error * 2);
                    }
                }
            for (int i = 0; i < height; i++)
                delete[] err[i];
        }

    }

    void atikson_dithering(int bitness) {
        if (bitness != 8) {
            auto **err = new double *[(int) this->height];
            for (int i = 0; i < this->height; i++) {
                err[i] = new double[(int) this->width];
                for (int j = 0; j < this->width; j++) {
```

```
                    err[i][j] = 0;
                }
            }
            for (int i = 0; i < this->height; i++)
                for (int j = 0; j < this->width; j++) {
                    double tmp = (double) this->array[i][j] +
err[i][j];
                    if (tmp > 255)
                        tmp = 255;
                    if (tmp < 0)
                        tmp = 0;
                    this->array[i][j] = (unsigned char) tmp;
                    unsigned char nc =
findNearestPaletteColor(this->array[i][j], bitness);
                    double error = (this->array[i][j] - nc) /
8.0;
                    this->array[i][j] = nc;
                    if (j + 1 < this->width) err[i][j + 1] +=
error;
                    if (j + 2 < this->width) err[i][j + 2] +=
error;
                    if (i + 1 < this->height) {
                        if (j - 1 >= 0) err[i + 1][j - 1] +=
error;
                        err[i + 1][j] += error;
                        if (j + 1 < this->width) err[i + 1][j +
1] += error;
                    }
                    if (i + 2 < this->height) {
                        err[i + 2][j] += error;
                    }
                }
            for (int i = 0; i < height; i++)
```

```
                delete[] err[i];
            }

        }

        void halftone_dithering(int bitness) {
            if (bitness != 8) {
                double halftoneMatrix[4][4] = {
                        {7.,  13., 11., 4.},
                        {12., 16., 14., 8.},
                        {10., 15., 6.,  2.},
                        {5.,  9.,  3.,  1.},
                };
                for (auto &i : halftoneMatrix) {
                    for (double &j : i) {
                        j = j / 17. - 0.5;
                    }
                }
                double resizer = this->depth / (pow(2., (double)
bitness) - 1);
                for (int i = 0; i < height; i++)
                    for (int j = 0; j < width; j++) {
                        int tmp = int((double) (this->array[i][j]) +
resizer * halftoneMatrix[i % 4][j % 4]);
                        if (tmp > 255) {
                            tmp = 255;
                        }
                        if (tmp < 0) {
                            tmp = 0;
                        }
                        this->array[i][j] =
findNearestPaletteColor(tmp, bitness);
```

```
            }
    }

}

explicit
NetPBM(ifstream
       *file, double gamma, bool gradient) {
    this->
            file = file;
    this->

            read_header();

    file->ignore(1);

    this->
            array = new unsigned char *[height];
    for (
            int i = 0;
            i < this->
                    height;
            i++) {
        this->array[i] = new unsigned char[width];
    }
    read_data(gamma, gradient);

}

~
```

```
    NetPBM() {
        for (int i = 0; i < height; i++) {
            delete[] array[i];
        }
        delete[] array;
    }

};

#endif //LAB3_NETPBM_H
```