**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

Компьютерная графика и геометрия

**Отчет**

по лабораторной работе № 7

**Изучение алгоритма повышения резкости**

Выполнила: студент гр. M32342

Ларионов А.Н.

Преподаватель: Скаков П.С.

Санкт-Петербург

2020

**Цель работы:** изучить алгоритм повышения резкости Contrast Adaptive Sharpening (с better diagonals, без масштабирования).

**Описание:**

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

Важно: Помимо реализации будет оцениваться изложение теории, представленной в отчете. Без раскрытия теоретического материала решение засчитано не будет.

Аргументы передаются через командную строку:

**lab7.exe <input> <output> <sharpen>,**

где sharpen - параметр резкости в диапазоне [0..1] (вещественное значение).

**Входные/выходные данные:** PNM P5 или P6 (RGB).

**Полное решение**: всё работает + корректно выделяется и освобождается память, закрываются файлы, есть обработка ошибок.

/\* да, частичного решения нет \*/

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:

С: fprintf(stderr, "Error\n");

C++: std::cerr

**Теоретическая часть**

**Резкость** описывает различимость деталей на фотографии, и она может использоваться как важный творческий инструмент для выделения текстуры.

На воспринимаемую резкость изображения влияют два фундаментальных фактора: разрешение и чёткость.

**Чёткость** описывает степень размытия границ, так что высокая чёткость подразумевает резкие, чётко видимые границы.

**Разрешение** описывает способность камеры отделить близко расположенные элементы друг от друга.

Резкость в изображении мы воспринимаем как степень контраста на контурах. Классический пример необходимости увеличить резкость — две поверхности различной яркости. При уменьшении изображения большинство алгоритмов интерполяции размывают границы между поверхностями. Чтобы вернуть резкость или, точнее, иллюзию таковой, необходимо затемнить контур на темном участке и осветлить его на светлом. Этим повышением контраста на контурах и пользуются алгоритмы усиления резкости.

Для повышения резкости изображения существует несколько алгоритмов, однако большинство из них основаны на сложных математических вычислениях.

**FidelityFX CAS** – алгоритм разработанный компанией AMD, объединяет в себе простоту и высокую эффективность. Алгоритм регулирует величину резкости на пиксель, чтобы обеспечить равномерный уровень резкости по всему изображению. Области входного изображения, которые уже являются резкими, становятся менее резкими, в то время как области, в которых отсутствует детализация, становятся более резкими. Это позволяет получить более высокую общую естественную визуальную четкость при меньшем количестве артефактов.

Описанный ниже алгоритм, рассмотрен на примере изображения с одним каналом, для работы с RGB достаточно произвести преобразования над каждым из каналов.

Алгоритм работает с аналоговыми значениями каналов, поэтому необходимо перед обработкой перевести все значения в диапазон [0; 1].

Преобразования выполняются для всех пикселей изображения, за исключением крайних (верхний ряд пикселей, нижний, правый и левый), их значения остаются прежними.

**Алгоритм:**

Для повышения яркости пикселя e, будем рассматривать его окрестность с радиусом 1:

|  |  |  |
| --- | --- | --- |
| a | b | c |
| d | e | f |
| g | h | i |

Непосредственными соседями пикселя e являются: d, b, f, h, диагональными: a, c, g, i.

1. Находим мягкий максимум и минимум окрестности – максимум и минимум, среди пикселей, расположенных непосредственно сверху, снизу, слева, справа, и самого пикселя.
2. К мягким максимуму и минимуму прибавим максимум и минимум по всей окрестности.
3. Выведем величину A, называемую базовой величиной резкости.

1. Найдем вес ядра.

Здесь *sh* – параметр резкости в диапазоне [0; 1], величина, передаваемая алгоритму на вход.

Окончательно ядро фильтра будет выглядеть таким образом:

|  |  |  |
| --- | --- | --- |
| 0 |  | 0 |
|  | 1 |  |
| 0 |  | 0 |

1. Вычислим новое значение для пикселя, поделив произведение локальной окрестности и весов ядра на сумму весов ядра.

**Экспериментальная часть**

Язык программирования: С++17, компилятор Microsoft Visual C++.

Этапы работы программы:

1. Чтение файла картинки (хедера и информации о цвете каждого пикселя) с обработкой ошибок чтения.
2. Выполнения алгоритма повышения резкости изображения с параметром, заданным в аргументах командной строки.
3. Запись полученного изображения в выходной файл

**Выводы**

Выполнение данной лабораторной работы позволило изучить алгоритм повышения резкости изображения. Была реализована программа, использующая высокоэффективный алгоритм FidelityFX CAS.

**Листинг**

hw7.cpp  
  
#include <cmath>

#include <cstdio>

#include <iostream>

using std::min;

using std::max;

typedef unsigned char uchar;

const double CONST\_255 = 255.0;

const long long INF = static\_cast<long long>(1e8);

struct RGB {

uchar r, g, b;

};

RGB \*my\_data;

RGB \*temp\_data;

FILE \*my\_file;

uchar \*my\_bytes;

uchar \*temp\_bytes;

int cnt = 0;

int width, height;

void reader(int ind, int cur\_type) {

if (cur\_type == 5) {

cnt += fread(&my\_bytes[ind], sizeof(uchar), 1, my\_file);

} else {

cnt += fread(&my\_data[ind], sizeof(RGB), 1, my\_file);

}

}

void writer(int ind, int cur\_type) {

if (cur\_type == 5) {

fwrite(&temp\_bytes[ind], sizeof(uchar), 1, my\_file);

} else {

fwrite(&temp\_data[ind], sizeof(RGB), 1, my\_file);

}

}

uchar index(uchar \*d, int i, int j) {

return d[i \* width + j];

}

RGB indexRGB(RGB \*d, int i, int j) {

return d[i \* width + j];

}

void type5\_converter(double sh) {

for (int j = 0; j < height - 1; j++) {

for (int k = 0; k < width - 1; k++) {

const double a = index(my\_bytes, j - 1, k - 1) / CONST\_255;

const double b = index(my\_bytes, j, k - 1) / CONST\_255;

const double c = index(my\_bytes, j + 1, k - 1) / CONST\_255;

const double d = index(my\_bytes, j - 1, k) / CONST\_255;

const double e = index(my\_bytes, j, k) / CONST\_255;

const double f = index(my\_bytes, j + 1, k) / CONST\_255;

const double g = index(my\_bytes, j - 1, k + 1) / CONST\_255;

const double h = index(my\_bytes, j, k + 1) / CONST\_255;

const double i = index(my\_bytes, j + 1, k + 1) / CONST\_255;

const double mn = min(min(b, d), min(min(e, f), h)) +

min(min(b, min(min(d, e), min(f, h))), min(min(a, c), min(g, i)));

const double mx = max(max(b, d), max(max(e, f), h)) +

max(max(b, max(max(d, e), max(f, h))), max(max(a, c), max(g, i)));

const double A = sqrt((min(mn, (2 - mx))) / mx);

const double W = A \* (-1) / (8 - 3 \* sh);

const double brightness = ((b \* W + d \* W + e + f \* W + h \* W) / (1 + 4 \* W)) \* CONST\_255;

temp\_bytes[j \* width + k] = brightness;

}

}

}

void type6\_converter(double sh) {

for (int j = 1; j < height - 1; j++) {

for (int k = 1; k < width - 1; k++) {

const double aR = indexRGB(my\_data, j - 1, k - 1).r / CONST\_255;

const double bR = indexRGB(my\_data, j, k - 1).r / CONST\_255;

const double cR = indexRGB(my\_data, j + 1, k - 1).r / CONST\_255;

const double dR = indexRGB(my\_data, j - 1, k).r / CONST\_255;

const double eR = indexRGB(my\_data, j, k).r / CONST\_255;

const double fR = indexRGB(my\_data, j + 1, k).r / CONST\_255;

const double gR = indexRGB(my\_data, j - 1, k + 1).r / CONST\_255;

const double hR = indexRGB(my\_data, j, k + 1).r / CONST\_255;

const double iR = indexRGB(my\_data, j + 1, k + 1).r / CONST\_255;

const double aG = indexRGB(my\_data, j - 1, k - 1).g / CONST\_255;

const double bG = indexRGB(my\_data, j, k - 1).g / CONST\_255;

const double cG = indexRGB(my\_data, j + 1, k - 1).g / CONST\_255;

const double dG = indexRGB(my\_data, j - 1, k).g / CONST\_255;

const double eG = indexRGB(my\_data, j, k).g / CONST\_255;

const double fG = indexRGB(my\_data, j + 1, k).g / CONST\_255;

const double gG = indexRGB(my\_data, j - 1, k + 1).g / CONST\_255;

const double hG = indexRGB(my\_data, j, k + 1).g / CONST\_255;

const double iG = indexRGB(my\_data, j + 1, k + 1).g / CONST\_255;

const double aB = indexRGB(my\_data, j - 1, k - 1).b / CONST\_255;

const double bB = indexRGB(my\_data, j, k - 1).b / CONST\_255;

const double cB = indexRGB(my\_data, j + 1, k - 1).b / CONST\_255;

const double dB = indexRGB(my\_data, j - 1, k).b / CONST\_255;

const double eB = indexRGB(my\_data, j, k).b / CONST\_255;

const double fB = indexRGB(my\_data, j + 1, k).b / CONST\_255;

const double gB = indexRGB(my\_data, j - 1, k + 1).b / CONST\_255;

const double hB = indexRGB(my\_data, j, k + 1).b / CONST\_255;

const double iB = indexRGB(my\_data, j + 1, k + 1).b / CONST\_255;

const double mnR = min(min(bR, dR), min(min(eR, fR), hR)) +

min(min(bR, min(min(dR, eR), min(fR, hR))), min(min(aR, cR), min(gR, iR)));

const double mxR = max(max(bR, dR), max(max(eR, fR), hR)) +

max(max(bR, max(max(dR, eR), max(fR, hR))), max(max(aR, cR), max(gR, iR)));

const double mnG = min(min(bG, dG), min(min(eG, fG), hG)) +

min(min(bG, min(min(dG, eG), min(fG, hG))), min(min(aG, cG), min(gG, iG)));

const double mxG = max(max(bG, dG), max(max(eG, fG), hG)) +

max(max(bG, max(max(dG, eG), max(fG, hG))), max(max(aG, cG), max(gG, iG)));

const double mnB = min(min(bB, dB), min(min(eB, fB), hB)) +

min(min(bB, min(min(dB, eB), min(fB, hB))), min(min(aB, cB), min(gB, iB)));

const double mxB = max(max(bB, dB), max(max(eB, fB), hB)) +

max(max(bB, max(max(dB, eB), max(fB, hB))), max(max(aB, cB), max(gB, iB)));

const double AR = sqrt((min(mnR, (2 - mxR))) / mxR);

const double AG = sqrt((min(mnG, (2 - mxG))) / mxG);

const double AB = sqrt((min(mnB, (2 - mxB))) / mxB);

const double WR = AR \* (-1) / (8 - 3 \* sh);

const double WG = AG \* (-1) / (8 - 3 \* sh);

const double WB = AB \* (-1) / (8 - 3 \* sh);

const double brightnessR = ((bR \* WR + dR \* WR + eR + fR \* WR + hR \* WR) / (1 + 4 \* WR)) \* CONST\_255;

const double brightnessG = ((bG \* WG + dG \* WG + eG + fG \* WG + hG \* WG) / (1 + 4 \* WG)) \* CONST\_255;

const double brightnessB = ((bB \* WB + dB \* WB + eB + fB \* WB + hB \* WB) / (1 + 4 \* WB)) \* CONST\_255;

temp\_data[j \* width + k].r = brightnessR;

temp\_data[j \* width + k].g = brightnessG;

temp\_data[j \* width + k].b = brightnessB;

}

}

}

void convert(double sh, int cur\_type) {

if (cur\_type == 5) {

type5\_converter(sh);

} else if (cur\_type == 6) {

type6\_converter(sh);

}

}

void clear\_data(int cur\_type) {

if (cur\_type == 6) {

delete[] my\_data;

} else {

delete[] my\_bytes;

}

}

int show\_error\_and\_exit(const std::string &message) {

std::cerr << message << std::endl;

return 1;

}

int main(int argc, char \*argv[]) {

int type, flag;

if (argc != 4) {

show\_error\_and\_exit("Wrong number of arguments");

}

my\_file = fopen(argv[1], "rb");

if (my\_file == nullptr) {

show\_error\_and\_exit("Input file can't be opened");

}

fscanf(my\_file, "P%d%d%d%d\n", &type, &width, &height, &flag);

if (type < 5 || type > 6) {

fclose(my\_file);

show\_error\_and\_exit("Input file has a wrong format");

}

if (width <= 0 || height <= 0 || (width \* 1ll \* height > INF)) {

fclose(my\_file);

show\_error\_and\_exit("Not acceptable amount of data");

}

if (type == 5) {

my\_bytes = new uchar[height \* width + 1];

temp\_bytes = new uchar[height \* width + 1];

} else {

my\_data = new RGB[height \* width + 1];

temp\_data = new RGB[height \* width + 1];

}

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

reader(i \* width + j, type);

}

}

reader(height \* width, type);

if (cnt != height \* width) {

clear\_data(type);

fclose(my\_file);

show\_error\_and\_exit("Not acceptable amount of data");

}

for (int j = 0; j < height; j++) {

for (int k = 0; k < width; k++) {

if (type == 6) {

temp\_data[j \* width + k].r = my\_data[j \* width + k].r;

temp\_data[j \* width + k].g = my\_data[j \* width + k].g;

temp\_data[j \* width + k].b = my\_data[j \* width + k].b;

}

if (type == 5) {

temp\_bytes[j \* width + k] = my\_bytes[j \* width + k];

}

}

}

double sh = atof(argv[3]);

convert(sh, type);

fclose(my\_file);

my\_file = fopen(argv[2], "wb");

if (my\_file == nullptr) {

clear\_data(type);

show\_error\_and\_exit("Output file can't be opened");

}

fprintf(my\_file, "P%d\n%d %d\n%d\n", type, width, height, flag);

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

writer(i \* width + j, type);

}

}

clear\_data(type);

fclose(my\_file);

return 0;

}