**Министерство образования и науки Российской Федерации**

федеральное государственное автономное образовательное

учреждение высшего образования

**Санкт-Петербургский исследовательский университет**

**Информационных технологий, механики и оптики**

Факультет: ФИТиП

Дисциплина: Компьютерная графика и геометрия

**Отчет**

по лабораторной работе № 7

***Изучение алгоритма повышения резкости***

Выполнила: студент гр. M3101

Дымчикова А. Ч.

Преподаватель: Скаков П.С.

Санкт-Петербург

2020

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

**Описание работы**

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

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**

Contrast Adaptive Sharpening (CAS) – алгоритм, разработанный компанией AMD, позволяющий повысить резкость изображения.

Алгоритм работает с нормализованными значениями цвета (значения в диапазоне [0; 1]). Применяется к одному каналу. Таким образом, для цветного изображения в формате RGB необходимо привести все значения для каждой составляющей в рабочий диапазон и последовательно применить алгоритм к каждой из них.

Рассмотрим алгоритм на примере повышения яркости одного пикселя (отметим его цифрой 5). Будем работать с квадратом 3х3 из пикселей, в центре которого находится пиксель 5 (см. картинку). При этом, пиксели 2, 4, 6, 8 будут называться иначе, как ring, а пиксели 1, 3, 7, 9 – circle.

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

* + 1. Для начала нужно найти минимум (ring\_min) и максимум (ring\_max) для пикселей квадрата — минимум и максимум среди пикселей, соседних по горизонтали и вертикали к пикселю 5 и самого пикселя 5:
    2. После этого необходимо вычислить значения area\_min и area\_max, которые находятся как сумма ring\_min/ring\_max и минимума/максимума по всей окрестности соответственно:
    3. Затем нужно вычислить базовую величину резкости base:
    4. И также необходимо вычислить вес ядра:

Здесь sharpen – параметр резкости, передаваемый алгоритму в качестве аргумента.

* + 1. В результате, фильтр будет выглядеть следующим образом:

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

А новое значение пикселя будет вычисляться по формуле:

Где (2), (4), (5), (6), (8) – значения яркости пикселей в выбранном канале.

**Экспериментальная часть**

Выполнено полное решение лабораторной работы.

Язык программирования: С++17.

Реализация программной части выполнена в точности по формулам из теоретической.

Для удобства работа всегда ведется со всеми тремя каналами, но для случая с однотонным пространством часть действий пропускается.

После применения алгоритма составляющие цвета необходимо снова привести к первоначальному диапазону значений ([0; 255]).

**Выводы**

В результате данной работы был изучен алгоритм повышения резкости CAS, позволяющий заметно улучшать качество изображений.

**Листинг**

**PNM\_ex.h**

#pragma once

#include <vector>

#include <string>

**class** **PNM\_Header** {

**public:**

**char** format;

**int** width;

**int** height;

**int** depth;

PNM\_Header();

PNM\_Header(**FILE** \*fin);

};

**class** **PNM\_File** {

**public:**

PNM\_Header header;

std::vector <**unsigned** **char**> data;

**int** size = **0**;

PNM\_File();

PNM\_File(**FILE** \*fin);

**void** **write**(**FILE** \*fout);

};

**PNM\_ex.cpp**

#include "PNM\_ex.h"

PNM\_Header::PNM\_Header() {};

PNM\_Header::PNM\_Header(**FILE** \*fin) {

**char** checker;

**int** check = fscanf(fin, "%c%c**\n**%d %d**\n**%d**\n**", &checker, &format, &width, &height, &depth);

**if** (check != **5** || checker != 'P' || format != '5' && format != '6'

|| width <= **0** || height <= **0** || depth != **255**) {

**throw** std::exception("Invalid data format!**\n**");

}

}

PNM\_File::PNM\_File() {};

PNM\_File::PNM\_File(**FILE** \*fin) {

header = PNM\_Header(fin);

**if** (header.format == '5') {

size = header.width \* header.height;

}

**else** {

size = header.width \* header.height \* **3**;

}

data.resize(size);

**int** check = fread(&data[**0**], **sizeof**(**unsigned** **char**), size, fin);

**if** (check != size) {

**throw** std::exception("Invalid data format: width or**\a**nd height are wrong!**\n**");

}

}

**void** PNM\_File::write(**FILE** \*fout) {

fprintf(fout, "P%c**\n**%d %d**\n**%d**\n**", header.format, header.width, header.height, header.depth);

fwrite(&data[**0**], **sizeof**(**unsigned** **char**), size, fout);

fclose(fout);

}

**Functs.h**

#pragma once

#include <algorithm>

#include "PNM\_ex.h"

**const** **int** INF = **1e9**;

**const** **double** EPS = **1e-3**;

**class** **component** {

**public:**

**double** a;

**double** b;

**double** c;

component();

component(**double**, **double**, **double**);

};

component **get\_component**(PNM\_File &file, **int** i, **int** j);

**void** **write\_component**(PNM\_File &file, **int** i, **int** j, component &new\_color);

**void** **cas**(PNM\_File &file, **double** sharpen);

**Functs.cpp**

#include <algorithm>

#include <vector>

#include <exception>

#include "PNM\_ex.h"

#include "Functs.h"

component::component(): a(**0**), b(**0**), c(**0**) {}

component::component(**double** a, **double** b, **double** c): a(a), b(b), c(c) {}

component get\_component(PNM\_File &file, **int** i, **int** j) {

**int** w = file.header.width;

component color;

**if** (file.header.format == '6') {

color.a = **double**(file.data[i \* w \* **3** + j \* **3** + **0**]) / **255**;

color.b = **double**(file.data[i \* w \* **3** + j \* **3** + **1**]) / **255**;

color.c = **double**(file.data[i \* w \* **3** + j \* **3** + **2**]) / **255**;

}

**else** {

color.a = **double**(file.data[i \* w + j]) / **255**;

color.b = -INF;

color.c = -INF;

}

**return** color;

}

**void** write\_component(PNM\_File &file, **int** i, **int** j, component &new\_color) {

**int** w = file.header.width;

component color;

**if** (file.header.format == '6') {

file.data[i \* w \* **3** + j \* **3** + **0**] = new\_color.a;

file.data[i \* w \* **3** + j \* **3** + **1**] = new\_color.b;

file.data[i \* w \* **3** + j \* **3** + **2**] = new\_color.c;

}

**else** {

file.data[i \* w + j] = new\_color.a;

}

}

**double** get\_ker(**double** bright, std::vector<**double**> ring, std::vector<**double**> circle, **double** sharpen) {

**if** (bright == -INF) {

**return** -INF;

}

**double** ring\_min = bright;

**double** ring\_max = bright;

**for** (**auto** p : ring) {

ring\_min = std::min(p, ring\_min);

ring\_max = std::max(p, ring\_max);

}

**double** area\_min = ring\_min;

**double** area\_max = ring\_max;

**for** (**auto** p : circle) {

area\_min = std::min(p, area\_min);

area\_max = std::max(p, area\_max);

}

area\_min += ring\_min;

area\_max += ring\_max;

**double** base = sqrt(std::min(area\_min, **2** - area\_max) / area\_max);

**double** weight = base \* (-**1.**) / (**8** \* (**1** - sharpen) + **5** \* sharpen);

**double** res = bright;

**for** (**auto** p : ring) {

res += p \* weight;

}

**return** res / (**1** + **4** \* weight);

}

**void** cas(PNM\_File &file, **double** sharpen) {

**int** h = file.header.height;

**int** w = file.header.width;

std::pair<**int**, **int**> shift\_circle[] = { {-**1**, -**1**}, {-**1**, **1**}, {**1**, -**1**}, {**1**, **1**} };

std::pair<**int**, **int**> shift\_ring[] = { {-**1**, **0**}, {**0**, -**1**}, {**1**, **0**}, {**0**, **1**} };

**for** (**int** i = **0**; i < h; i++) {

**for** (**int** j = **0**; j < w; j++) {

component color = get\_component(file, i, j);

std::vector<**double**> ring[**3**];

std::vector<**double**> circle[**3**];

**for** (**auto** shift : shift\_ring) {

**if** (i + shift.first >= **0** && i + shift.first < h &&

j + shift.second >= **0** && j + shift.second < w) {

component shift\_color = get\_component(file, i + shift.first, j + shift.second);

ring[**0**].push\_back(shift\_color.a);

ring[**1**].push\_back(shift\_color.b);

ring[**2**].push\_back(shift\_color.c);

}

}

**for** (**auto** shift : shift\_circle) {

**if** (i + shift.first >= **0** && i + shift.first < h &&

j + shift.second >= **0** && j + shift.second < w) {

component shift\_color = get\_component(file, i + shift.first, j + shift.second);

circle[**0**].push\_back(shift\_color.a);

circle[**1**].push\_back(shift\_color.b);

circle[**2**].push\_back(shift\_color.c);

}

}

component new\_color;

new\_color.a = get\_ker(color.a, ring[**0**], circle[**0**], sharpen);

new\_color.b = get\_ker(color.b, ring[**1**], circle[**1**], sharpen);

new\_color.c = get\_ker(color.c, ring[**2**], circle[**2**], sharpen);

**if** (new\_color.a < **0**) {

new\_color.a = **0**;

}

**if** (new\_color.b < **0** && new\_color.b != -INF) {

new\_color.b = **0**;

}

**if** (new\_color.c < **0** && new\_color.c != -INF) {

new\_color.c = **0**;

}

**if** (new\_color.a > **1**) {

new\_color.a = **1**;

}

**if** (new\_color.b > **1**) {

new\_color.b = **1**;

}

**if** (new\_color.c > **1**) {

new\_color.c = **1**;

}

new\_color.a = round(new\_color.a \* **255**);

new\_color.b = round(new\_color.b \* **255**);

new\_color.c = round(new\_color.c \* **255**);

write\_component(file, i, j, new\_color);

}

}

}

**main.cpp**

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <map>

#include "PNM\_ex.h"

#include "Functs.h"

#include <stdlib.h>

**using** **namespace** std;

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

**if** (argc != **4**) {

cerr << "Wrong number of arguments!";

**return** **1**;

}

**FILE** \*fin = fopen(argv[**1**], "rb");

**if** (!fin) {

cerr << "Input file error!**\n**";

cerr << argv[**1**];

**return** **1**;

}

PNM\_File file;

try {

file = PNM\_File(fin);

}

**catch** (exception err) {

fclose(fin);

cerr << err.what();

**return** **1**;

}

fclose(fin);

cas(file, atof(argv[**3**]));

**FILE** \*fout = fopen(argv[**2**], "wb");

**if** (!fout) {

cerr << "Output file error!";

**return** **1**;

}

file.write(fout);

**return** **0**;

}