**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 №7

***Изучение алгоритма повышения яркости***

Выполнила: студент гр. M3101

Семина А. Д.

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.

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");

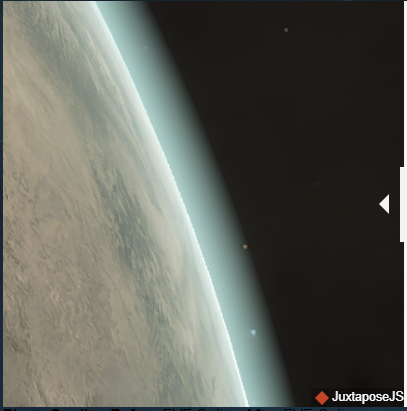
C++: std::cerr

**Теоретическая часть**

Процесс коррекции изображения и повышения его резкости играет важную роль в работе с любыми визуальными материалами. Кинорежиссёры, фотографы и художники, работающие над созданием игровых текстур, ежедневно прибегают к этой технике в стремлении достичь максимального качества картинки. Но всё важно сделать правильно, ведь, если допустить ошибку и перестараться с резкостью, изображение будет выглядеть неестественно и на нём начнут проявляться артефакты.

Почему так случается? Потому что именно так и происходит повышение резкости — за счёт увеличения контрастности между пикселями, что особенно заметно на границе объектов. Другими словами, когда один пиксель становится ярче, другой становится темнее.

Слева внизу находится обычный неотредактированный снимок экрана. Справа — то же самое изображение, но на него был дважды наложен фильтр резкости в Adobe Photoshop. Этот пример наглядно показывает, что может произойти, если неправильно использовать увеличение резкости. На правом краю планеты появились артефакты, переход цветов перестал быть плавным, а на поверхности планеты отчётливо видны пиксели.



Эти кадры хорошо демонстрируют, как неумелое повышение резкости может навредить качеству двухмерного изображения. Но если речь идёт о трёхмерном пространстве, в котором камера и корабли постоянно двигаются, всё будет выглядеть гораздо хуже.

В итоге на ситуацию обратили внимание специалисты компании AMD, которые в рамках своего набора инструментов GPUOpen FidelityFX создали бесплатный шейдер для повышения резкости с открытым исходным кодом. Он называется Contrast Adaptive Sharpening (контрастно-адаптивная резкость) и совместим с видеокартами любых производителей.

Как следует из названия, Contrast Adaptive Sharpening (контрастно-адаптивная резкость), или CAS, в ходе повышения резкости в первую очередь учитывает контрастность пикселей. Границы и переходы больше не страдают от обилия артефактов, как на изображении выше, и при этом новый шейдер способен очень точно повышать резкость в сценах с большими различиями в контрастности.

**А вот и сам алгоритм (с better diagonals, без масштабирования):**

Для каждого пикселя (е) выбираем окрестность 3x3

a b c

d(e)f

g h i

Затем вычисляем «мягкий» минимум MIN и максимум MAX, которые дают представление о локальном контрасте

a b c b

d e f \* 0.5 + d e f \* 0.5

g h i h

--- 1.0 ^

| | <-- S1

--- MAX v

|

--- MIN ^

| | <-- S2

--- 0.0 v

S1 – расстояние от полученного MAX до верхнего предельного значения цвета, т.е. 1, S2 – расстояние от нижнего предельного значения цвета, т.е. 0, до MIN.

Базовая величина повышения резкости A = sqrt (min(S1, S2) / MAX).

A варьируется от 0 (нет резкости) до 1 (полная резкость).

Далее A масштабируется в зависимости от полученной на входе резкости, а именно умножается на - 1 / mix( 8, 5, резкость), где mix (x, y, a) = x \* (1 - a) + y \* a (для A = 1 получаются значения от -1/5 до -1/8).

Конечный ​​фильтр выглядит так:

0 A 0

A 1 A

0 A 0

Затем локальная окрестность пикселя е умножается на веса фильтра, суммируется и делится на сумму весов.

Для высокого качества вычисляется вес фильтра на канал, для низкого используют вес фильтра зеленого канала для вычисления коэффициента А для всех каналов.

**Экспериментальная часть**

Язык программирования: C++ 14

Данные считываются в формате, указанном в описании работы. Если данные не соответствуют формату, то выводится ошибка и программа завершает свою работу.

Все данные я считываю в массив image (цвета 0..255), для P5 я записываю пиксели image в массив RGB (цвета 0..1), для P6 - в массивы R, G, B (цвета 0..1) для каждого канала. Далее выполняется алгоритм, описанный в теории, и значения получившихся пикселей записываются обратно в image.

**Выводы**

В ходе проделанной работы были получены следующие результаты:

Время работы алгоритма - O(w \* h), где w и h – высота и ширина изображения. Сделать быстрее не получится, так как хотя бы раз мы должны пройти по всем пикселям изображения.

**Листинг**

**Название исходного файла : main.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <algorithm>

#include <vector>

#include <string>

#include <cmath>

using namespace std;

int quantity;

double sharpen;

char symbol;

int number, h, w, max\_color;

vector<unsigned char> image;

//P6

vector<unsigned char> R\_, G\_, B\_;

vector<double> R, G, B;

//P5

vector<double> RGB;

FILE \*fin, \*fout;

double Sat(double c)

{

return min(1.0, max(0.0, c));

}

double mix(double x, double y, double a)

{

return x \* (1 - a) + y \* a;

}

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

{

//lab7.exe <input> <output> <sharpen>

if (argc == 4)

{

try

{

sharpen = stof(argv[3]);

if(sharpen < 0 || sharpen > 1)

{

cerr << "sharpen < 0 || sharpen > 1";

return 1;

}

}

catch(...)

{

cerr << "Invalid arguments";

return 1;

}

}

else

{

cerr << "Invalid number of arguments";

return 1;

}

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

if (fin == NULL)

{

cerr << "Input file can't be open";

return 1;

}

quantity = fscanf(fin, "%c%d\n%d%d\n%d\n", &symbol, &number, &w, &h, &max\_color);

if (quantity != 5 || symbol != 'P' || (number != 5 && number != 6) || h <= 0 || w <= 0 || max\_color <= 0 || max\_color > 255)

{

fclose(fin);

cerr << "Invalid ppm file";

return 1;

}

if (number == 5)

{

image.resize(h \* w, 0);

RGB.resize(h \* w, 0);

quantity = fread(&image[0], sizeof(unsigned char), image.size(), fin);

fclose(fin);

if (quantity != h \* w)

{

cerr << "Invalid pgm file: quantity != h \* w";

return 1;

}

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

{

RGB[i] = image[i] / 255.;

}

}

else

{

image.resize(3 \* h \* w, 0);

R.resize(h \* w, 0);

G.resize(h \* w, 0);

B.resize(h \* w, 0);

R\_.resize(h \* w, 0);

G\_.resize(h \* w, 0);

B\_.resize(h \* w, 0);

quantity = fread(&image[0], sizeof(unsigned char), image.size(), fin);

fclose(fin);

if (quantity != 3 \* h \* w)

{

cerr << "Invalid pnm file: quantity != 3 \* h \* w";

return 1;

}

for(int i = 0; i < quantity; i += 3)

{

R[i / 3] = image[i] / 255.;

G[i / 3] = image[i + 1] / 255.;

B[i / 3] = image[i + 2] / 255.;

R\_[i / 3] = image[i] ;

G\_[i / 3] = image[i + 1];

B\_[i / 3] = image[i + 2];

}

}

//Contrast Adaptive Sharpening with better diagonals

if (number == 5)

{

for (int y = 1; y < h - 1; ++y)

for (int x = 1; x < w - 1; ++x)

{

int a = (y - 1) \* w + x - 1;

int b = (y - 1) \* w + x;

int c = (y - 1) \* w + x + 1;

int d = y \* w + x - 1;

int e = y \* w + x;

int f = y \* w + x + 1;

int g = (y + 1) \* w + x - 1;

int h = (y + 1) \* w + x;

int i = (y + 1) \* w + x + 1;

// Soft min and max.

// a b c b

// d e f \* 0.5 + d e f \* 0.5

// g h i h

// These are 2.0x bigger (factored out the extra multiply).

double min\_ = min(RGB[d], min(RGB[e], min(RGB[f], min(RGB[b], RGB[h]))));

double min\_2 = min(min\_, min(RGB[a], min(RGB[c], (min(RGB[g], RGB[i])))));

min\_ += min\_2;

double max\_ = max(RGB[d], max(RGB[e], max(RGB[f], max(RGB[b], RGB[h]))));

double max\_2 = max(max\_, max(RGB[a], max(RGB[c], (max(RGB[g], RGB[i])))));

max\_ += max\_2;

double rcpM = 1. / max\_;

double amp= Sat(min(min\_, 2.0 - max\_) \* rcpM);

amp = sqrt(amp);

// Filter shape.

// 0 w 0

// w 1 w

// 0 w 0

//double peak=AF1\_AU1(const1.x);

double sharp = - 1. / (mix(8.0,5.0, Sat(sharpen)));

double peak = sharp;

double wRGB = amp \* peak;

double rcpWeight = 1. / (1.0 + 4.0 \* wRGB);

image[y \* w + x] = (unsigned char)(255 \* Sat((RGB[b] \* wRGB + RGB[d] \* wRGB + RGB[f] \* wRGB + RGB[h] \* wRGB + RGB[e]) \* rcpWeight));

}

}

else

{

for (int y = 1; y < h - 1; ++y)

for (int x = 1; x < w - 1; ++x)

{

int a = (y - 1) \* w + x - 1;

int b = (y - 1) \* w + x;

int c = (y - 1) \* w + x + 1;

int d = y \* w + x - 1;

int e = y \* w + x;

int f = y \* w + x + 1;

int g = (y + 1) \* w + x - 1;

int h = (y + 1) \* w + x;

int i = (y + 1) \* w + x + 1;

double min\_R = min(R[d], min(R[e], min(R[f], (min(R[b], R[h])))));

double min\_G = min(G[d], min(G[e], min(G[f], (min(G[b], G[h])))));

double min\_B = min(B[d], min(B[e], min(B[f], (min(B[b], B[h])))));

double min\_R2 = min(min\_R, min(R[a], min(R[c], (min(R[g], R[i])))));

double min\_G2 = min(min\_G, min(G[a], min(G[c], (min(G[g], G[i])))));

double min\_B2 = min(min\_B, min(B[a], min(B[c], (min(B[g], B[i])))));

min\_R += min\_R2;

min\_G += min\_G2;

min\_B += min\_B2;

double max\_R = max(R[d], max(R[e], max(R[f], (max(R[b], R[h])))));

double max\_G = max(G[d], max(G[e], max(G[f], (max(G[b], G[h])))));

double max\_B = max(B[d], max(B[e], max(B[f], (max(B[b], B[h])))));

double max\_R2 = max(max\_R, max(R[a], max(R[c], (max(R[g], R[i])))));

double max\_G2 = max(max\_G, max(G[a], max(G[c], (max(G[g], G[i])))));

double max\_B2 = max(max\_B, max(B[a], max(B[c], (max(B[g], B[i])))));

max\_R += max\_R2;

max\_G += max\_G2;

max\_B += max\_B2;

// Soft min and max.

// a b c b

// d e f \* 0.5 + d e f \* 0.5

// g h i h

// These are 2.0x bigger (factored out the extra multiply).

double rcpMR = 1. / max\_R;

double rcpMG = 1. / max\_G;

double rcpMB = 1. / max\_B;

double ampR= Sat(min(min\_R, 2.0 - max\_R) \* rcpMR);

double ampG= Sat(min(min\_G, 2.0 - max\_G) \* rcpMG);

double ampB = Sat(min(min\_B, 2.0 - max\_B) \* rcpMB);

ampR = sqrt(ampR);

ampG = sqrt(ampG);

ampB = sqrt(ampB);

// Filter shape.

// 0 w 0

// w 1 w

// 0 w 0

//double peak=AF1\_AU1(const1.x);

double sharp = - 1. / (mix(8.0,5.0, Sat(sharpen)));

double peak = sharp;

double wR = ampR \* peak;

double wG = ampG \* peak;

double wB = ampB \* peak;

double rcpWeightR = 1. / (1.0 + 4.0 \* wR);

double rcpWeightG = 1. / (1.0 + 4.0 \* wG);

double rcpWeightB = 1. / (1.0 + 4.0 \* wB);

R\_[y \* w + x] = (unsigned char)(255 \* Sat((R[b] \* wR + R[d] \* wR + R[f] \* wR + R[h] \* wR + R[e]) \* rcpWeightR));

G\_[y \* w + x] = (unsigned char)(255 \* Sat((G[b] \* wG + G[d] \* wG + G[f] \* wG + G[h] \* wG + G[e]) \* rcpWeightG));

B\_[y \* w + x] = (unsigned char)(255 \* Sat((B[b] \* wB + B[d] \* wB + B[f] \* wB + B[h] \* wB + B[e]) \* rcpWeightB));

}

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

{

image[3 \* i] = R\_[i];

image[3 \* i + 1] = G\_[i];

image[3 \* i + 2] = B\_[i];

}

}

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

if (fout == NULL)

{

cerr << "Output file can't be open";

return 1;

}

fprintf(fout, "%c%d\n%d %d\n%d\n", symbol, number, w, h, max\_color);

int quantity\_out = fwrite(&image[0], sizeof(unsigned char), image.size(), fout);

fclose(fout);

if (quantity != quantity\_out)

{

cerr << "Quantity of output file != width \* height";

return 1;

}

return 0;

}