МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №2**
**по курсу «Программирование графических процессоров»**

# Обработка изображений на GPU. Фильтры.

Выполнил: Р.Р. Гаптулхаков
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2022

## Условие

Необходимо реализовать размытие по Гауссу с линейной сложностью от радиуса размытия(двухпроходный алгоритм).

**Вариант 7.** Поэлементное вычисление модуля вектора.
**Входные данные.** На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. На следующей строке, целое число r -- радиус размытия, $w*h \leq 5 * 10^7$, $0 \leq r \leq 1024$.

**Выходные данные.** Бинарный файл

## Программное и аппаратное обеспечение

Таблица 1 — Характеристики графического процессора

| | |
|---|---|
| **Compute capability** | 2.1 |
| **Name** | GeForce GT 545 |
| **Total Global Memory** | 3150381056 |
| **Shared memory per block** | 49152 |
| **Registers per block** | 32768 |
| **Warp size** | 32 |
| **Max threads per block** | (1024, 1024, 64) |
| **Max block** | (65535, 65535, 65535) |
| **Total constant memory** | 65536 |
| **Multiprocessors count** | 3 |

Таблица 2 — Используемое ПО

| | |
|---|---|
| **Operating system** | Windows 11 + ssh Ubuntu 16.04.6 LTS |
| **IDE** | Visual Studio Code |
| **Compiler** | nvcc |

Таблица 3 — Характеристики процессора

| | |
|---|---|
| **Name** | Intel(R) Core(TM) i7-3770 CPU |
| **Architecture** | x86_64 |
| **CPU(s)** | 8 |
| **Thread(s) per core** | 2 |
| **Core(s) per socket** | 4 |
| **CPU max MHz** | 3900 |
| **CPU min MHz** | 1600 |
| **CPU MHz** | 1800 |
| **L1d cache** | 32K |
| **L1i cache** | 32K |
| **L2 cache** | 256K |
| **L3 cache** | 8192K |

Таблица 4 — Оперативная память

| | |
|---|---|
| **Size** | 16 G |

Таблица 5 — Постоянная память

| | |
|---|---|
| **Size** | 1000 G |

## Метод решения

Быстрое размытие по Гауссу выполняется в два этапа. С начала проходимся фильтром по горизонтали, затем по вертикали. Фильтр — окно с радиусом r, каждый элемент которого имеет свой вес. Для каждого пикселя изображения в отдельном потоке вычисляем сумму произведения веса фильтра на соответствующий пиксели.

Коэффициенты в фильтре вычисляются по следующей формуле:

$$y(m,n) = \frac{1}{2\pi r}\sum_{u,v} e^{\frac{-(u^2+v^2)}{2r^2}} x(m+u, n+v)$$

Или если преобразовать:

$$f(m,n) = \frac{1}{r\sqrt{2\pi}}\sum_{u,v} e^{\frac{-u^2}{2r^2}} x(m+u,n)$$

$$y(m,n) = \frac{1}{2\pi r^2}\sum_{u,v} e^{\frac{-(u^2+v^2)}{2r^2}} x(m+u,n+v) =$$

$$= \frac{1}{r\sqrt{2\pi}}\sum_{v} e^{\frac{-v^2}{2r^2}} f(m,n+v)$$

## Описание программы

Для оптимального выполнения программы вычислим на CPU коэффициенты для фильтра. И запишим их в константную память, так как там есть кэш.

```
__constant__ double coefficients[1025];

__host__ double calculate_coefficients(int r){
    const double PI = acos(-1.0);
    double sum = 0.0, tmp;
    double constant = 1.0 / (sqrt(2.0 * PI) * r);
    double coeffs[1025];
    for (int i = 0; i <= r; ++i){
        tmp = constant * exp((-(double)(i * i)) / (double)(2 *
r * r));
        sum += 2 * tmp;
        coeffs[i] = tmp;
    }
    sum -= constant * exp((-(0.0 * 0.0)) / (2 * r * r)); //
Так как два раза учитываем центральный элемент
    for (int i = 0; i <= r; ++i){
        coeffs[i] /= sum;
    }
    CSC(cudaMemcpyToSymbol(coefficients, coeffs, (1025) *
sizeof(double)));
    return sum;
```

```
}
```
Затем проходим по горизонтали и для каждого пикселя вычисляем новое значение с использованием константной памяти.
```
__global__ void horizontal(uchar4 *out, int w, int h, int r,
double sum){
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    int idy = blockDim.y * blockIdx.y + threadIdx.y;
    int offsetx = blockDim.x * gridDim.x;
    int offsety = blockDim.y * gridDim.y;
    int x, y;
    uchar4 pixel;
    for(y = idy; y < h; y += offsety){
        for(x = idx; x < w; x += offsetx){
            double coeff;
            double4 out_p;
            out_p.x = 0;
            out_p.y = 0;
            out_p.z = 0;
            for (int i = -r; i <= r; ++i) {
                coeff = coefficients[abs(i)];
                int y_ = max(0, min(y, h));
                int x_ = max(0, min(x + i, w));

                pixel = tex2D(tex, x_, y_);

                out_p.x += pixel.x * coeff;
                out_p.y += pixel.y * coeff;
                out_p.z += pixel.z * coeff;
            }
            out[y * w + x] = make_uchar4(out_p.x, out_p.y,
out_p.z, 255);
        }
    }
}
```

Копируем данные на хост и с новыми данными проходим фильтром по вертикали.
```
__global__ void vertical(uchar4 *out, int w, int h, int r,
double sum){
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    int idy = blockDim.y * blockIdx.y + threadIdx.y;
    int offsetx = blockDim.x * gridDim.x;
    int offsety = blockDim.y * gridDim.y;
    int x, y;
    uchar4 pixel;
```

```
    for(y = idy; y < h; y += offsety){
        for(x = idx; x < w; x += offsetx) {
            double coeff;
            double4 out_p;
            out_p.x = 0;
            out_p.y = 0;
            out_p.z = 0;
            for (int i = -r; i <= r; ++i) {
                coeff = coefficients[abs(i)];
                int y_ = max(0, min(y + i, h));
                int x_ = max(0, min(x, w));

                pixel = tex2D(tex, x_, y_);

                out_p.x += pixel.x * coeff;
                out_p.y += pixel.y * coeff;
                out_p.z += pixel.z * coeff;
            }
            out[y * w + x] = make_uchar4(out_p.x, out_p.y,
out_p.z, 255);

        }
    }
}
```

## Результаты

Входная картинка:


Выходная картинка:

r = 7

Таблица 6 — Результаты

| **Количество потоков** | **512x768** |
|---|---|
| $<<<16^2, 32^2>>>$ | 6.656992 |
| $<<<32^2, 32^2>>>$ | 7.040032 |
| $<<<16^2, 16^2>>>$ | 6.627360 |
| $<<<8^2, 16^2>>>$ | 6.775296 |
| $<<<16^2, 8^2>>>$ | 7.942880 |
| $<<<8^2, 32^2>>>$ | 6.646464 |
| $<<<32^2, 8^2>>>$ | 7.842400 |
| CPU | 253.503 |

## Выводы

Реализованная программа широко применяется для устранения различных шумов с изображений. Например в программе Photoshop, инструмент размытия по Гауссу является одним из основных. Также мне стало лучше понятно, как работают сверточные слои в нейронных сетях. Модель может выучить сверточный слой, который размывает изображения для упрощения дальнейшей своей работы. Я долго не мог понять, почему после обработки изображения, я не получал желаемого результата. Проблема заключалась в альфа канале. После выбора нужного значения альфа канала программа стала работать, хотя на тесты не затрагивают его при проверке. С различным количеством потоков результаты сильно не отличаются, но видно сильное ускорение по сравнению с вычислением на CPU.