ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное образовательное учреждение высшего профессионального образования ордена Трудового Красного Знамени

«МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ»

МКиИТ

**ЛАБОРАТОРНАЯ РАБОТА №3**

По дисциплине «Распределенные и параллельные вычисления»

На тему **«Размытие изображения на CUDA»**

Выполнил студент 2 курса группы МБД2032:

Трояновский Семён Владиславович

Москва

2021

**Задача лабораторной работы:**

Цель лабораторной – реализовать эффективный алгоритм размытия входного изображения. Как и в лабораторной по свертке изображения, изображение кодируется в формате RGB значениями типа float.

**Ход выполнения:**

1. **Алгоритм для расчётов**

Как и при выполнении второй работы, сначала нужно определиться с алгоритмом, который будет выполнять размытие. В моем случае этот алгоритм выглядит так:



1. **Полный код**

Полный код метода выглядит примерно также как и в предыдущем случае. Однако, есть и существенная разница: здесь мы обрабатываем каждый канал цвета отдельно, а потом собираем снова в общую композицию:

def image\_blur(input\_image: str, output\_image: str):

# ################################################# #

# load image in to array and extract color channels #

# ################################################# #

try:

img = Image.open(input\_image)

input\_array = np.array(img)

red\_channel = input\_array[:, :, 0].copy()

green\_channel = input\_array[:, :, 1].copy()

blue\_channel = input\_array[:, :, 2].copy()

except FileNotFoundError:

sys.exit("Cannot load image file")

# ######################################## #

# generate gaussian kernel (size of N \* N) #

# ######################################## #

sigma = 2 # standard deviation of the distribution

kernel\_width = int(3 \* sigma)

if kernel\_width % 2 == 0:

kernel\_width = kernel\_width - 1 # make sure kernel width only sth 3,5,7 etc

# create empty matrix for the gaussian kernel #

kernel\_matrix = np.empty((kernel\_width, kernel\_width), np.float32)

kernel\_half\_width = kernel\_width // 2

for i in range(-kernel\_half\_width, kernel\_half\_width + 1):

for j in range(-kernel\_half\_width, kernel\_half\_width + 1):

kernel\_matrix[i + kernel\_half\_width][j + kernel\_half\_width] = (

np.exp(-(i \*\* 2 + j \*\* 2) / (2 \* sigma \*\* 2))

/ (2 \* np.pi \* sigma \*\* 2)

)

gaussian\_kernel = kernel\_matrix / kernel\_matrix.sum()

# ####################################################################

# calculate the CUDA threads/blocks/gird base on width/height of image

# ####################################################################

height, width = input\_array.shape[:2]

dim\_block = 32

dim\_grid\_x = math.ceil(width / dim\_block)

dim\_grid\_y = math.ceil(height / dim\_block)

# load CUDA code

kernel = '''

\_\_global\_\_ void applyFilter(const unsigned char \*input, unsigned char \*output, const unsigned int width, const unsigned int height, const float \*kernel, const unsigned int kernelWidth) {

const unsigned int col = threadIdx.x + blockIdx.x \* blockDim.x;

const unsigned int row = threadIdx.y + blockIdx.y \* blockDim.y;

if(row < height && col < width) {

const int half = kernelWidth / 2;

float blur = 0.0;

for(int i = -half; i <= half; i++) {

for(int j = -half; j <= half; j++) {

const unsigned int y = max(0, min(height - 1, row + i));

const unsigned int x = max(0, min(width - 1, col + j));

const float w = kernel[(j + half) + (i + half) \* kernelWidth];

blur += w \* input[x + y \* width];

}

}

output[col + row \* width] = static\_cast<unsigned char>(blur);

}

}

'''

# mod = compiler.SourceModule(open('gaussian\_blur.cu').read())

mod = compiler.SourceModule(kernel)

apply\_filter = mod.get\_function('applyFilter')

# ##################

# apply the filter

# ##################

# start time

time\_started = timeit.default\_timer()

for channel in (red\_channel, green\_channel, blue\_channel):

apply\_filter(

drv.In(channel),

drv.Out(channel),

np.uint32(width),

np.uint32(height),

drv.In(gaussian\_kernel),

np.uint32(kernel\_width),

block=(dim\_block, dim\_block, 1),

grid=(dim\_grid\_x, dim\_grid\_y)

)

# end time

time\_ended = timeit.default\_timer()

####################################################################

# create the output array with the same

# shape and type as the input array

#####################################################################

output\_array = np.empty\_like(input\_array)

output\_array[:, :, 0] = red\_channel

output\_array[:, :, 1] = green\_channel

output\_array[:, :, 2] = blue\_channel

# save result image

Image.fromarray(output\_array).save(output\_image)

# display total time

print('Total processing time: ', time\_ended - time\_started, 's')

1. **Проверка результата**

Также нужно проверить что алгоритм отработал корректно и действительно получилось размытие, которого мы добивались:

Так выглядит оригинал:



А так версия с размытием:



Эффект размытия заметен.

1. **Ответы на вопросы.**

(1) Сколько операций с плавающей точкой выполняется в вашем ядре? ОБЪЯСНИТЬ.

*В правильном решении каждое ядро в среднем производит одно сложение с плавающей точкой и одно деление*

*для каждого пикселя.*

(2) Сколько чтений из глобальной памяти производится вашим ядром? ОБЪЯСНИТЬ.

*Одно чтение глобальной памяти на пиксель. Так как изображение монохромное это соответствует одному чтению*

*числа с плавающей точкой.*

(3) Сколько записей в глобальную память производится вашим ядром? ОБЪЯСНИТЬ.

*Одна запись в глобальную память на пиксель. Так как изображение монохромное это соответствует одной записи*

*числа с плавающей точкой.*

(4) Опишите, какие оптимизации можно сделать в вашем ядре, чтобы достичь увеличения производительности.

*Можно представить изображение с помощью беззнаковых символов (байтов). Тогда каждый пиксель будет занимать 1 байт против 4 в числах с плавающей точкой.*

1. **Вывод**

В данной работе я рассмотрел и реализовал практический пример программирования с использованием карты CUDA и выполнил размытие изображения.