Задание.
Метод максимального правдоподобия.

Для некоторого пикселя p, номер класса jc определяется следующим образом:

$$jc = arg\ max_j \left[-(p - avg_j)^T * cov_j^{-1} * (p - avg_j) - log(|det(cov_j)|) \right]$$

Пример:

Входной файл	hex: in.data	hex: out.data
in.data out.data 2 4 1 2 1 0 2 2 2 1 4 0 0 0 1 1 1 2 0	03000000 03000000 A2DF4C00 F7C9FE00 9ED84500 B4E85300 99D14D00 92DD5600 A9E04C00 F7D1FA00 D4D0E900	03000000 03000000 A2DF4C01 F7C9FE00 9ED84501 B4E85301 99D14D01 92DD5600 A9E04C01 F7D1FA00 D4D0E900

На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. На следующей строке, число nc -- количество классов. Далее идут nc строчек описывающих каждый класс. В начале j-ой строки задается число np_j -- количество пикселей в выборке, за ним следуют np_j пар чисел -- координаты пикселей выборки. $nc \le 32$, $np_j \le 2^{19}$, $w*h \le 4*10^8$.

Оценка вектора средних и ковариационной матрицы:

$$avg_j = \frac{1}{np_j} \sum_{i=1}^{np_j} ps_i^j$$

$$cov_j = \frac{1}{np_j - 1} \sum_{i=1}^{np_j} (ps_i^j - avg_j) * (ps_i^j - avg_j)^T$$

где $ps_i^j = (r_i^j \ \ g_i^j \ \ b_i^j)^T$ -- i-ый пиксель из j-ой выборки.

Программное и аппаратное обеспечение

Device: GeForce GT 545
Размер глобальной памяти: 3150381056
Размер константной памяти : 65536
Размер разделяемой памяти: 49152
Регистров на блок: 32768
Максимум потоков на блок: 1024
Количество мультипроцессоров : 3
OS: Linux Mint 20 Cinnamon
Редактор: VSCode

Метод решения

Для начала необходимо посчитать значения средних по каждому из каналов и значения обратной матрицы ковариации для каждого из заданных признаков. Далее для каждого из пикселей изображения определяется его класс по формуле:

$$jc = arg\ max_j \left[-(p - avg_j)^T * cov_j^{-1} * (p - avg_j) - log(\|cov_j\|) \right]$$

Описание программы

Для выполнения программы я реализовал собственный класс изображения в методе которого и вызывался kernel. Этот класс не потерпел значительных изменений со времени выполнения второй лабораторной работы.

Для выполнения операции я инициализировал на этапе компиляции массив константной памяти необходимого размера, а именно максимального количества возможных классов, умноженного на размер, необходимый для хранения вычислительной информации для каждого из классов.

```
struct class_data{
float avg_red;
float avg_green;
float avg_blue;
float cov11;
float cov12;
float cov13;
float cov21;
float cov22;
float cov23;
float cov31;
float cov32;
float cov33;
float log_det;
};
// constant memory
__constant__ class_data computation_data[MAX_CLASS_NUMBERS];
```

Для копирования данных с host в этот участок памяти у меня есть следующий код:

```
throw_on_cuda_error(
 cudaMemcpyToSymbol(computation_data, cov_avg,
```

```
 MAX_CLASS_NUMBERS*sizeof(class_data), 0, cudaMemcpyHostToDevice)
);
```

После чего Сами же данные для вычисления я расчитываю на CPU, поскольку расчет на GPU не даст значимого прироста к производительности.

После чего я вызываю kernel с заданным количеством блоков и потоков, где я преобразую изображение в кластеризованное по описанному алгоритму:

```
dim3 threads = dim3(MAX_X, MAX_Y);
            dim3 blocks = dim3(BLOCKS_X, BLOCKS_Y);

classification<<<blocks, threads>>>(d_data, _height, _widht, indexes.size());
throw_on_cuda_error(cudaGetLastError());
```

В самом kernel мы вычисляем правдоподобие для пикселя по каждому из классов и записываем наиболее вероятный номер класса.

```
__global__ void classification(uint32_t* picture, uint32_t h, uint32_t w, uint8_t classes){
uint32_t idx = blockIdx.x * blockDim.x + threadIdx.x;
uint32_t idy = blockIdx.y * blockDim.y + threadIdx.y;“

uint32_t step_x = blockDim.x * gridDim.x;
uint32_t step_y = blockDim.y * gridDim.y;

// run for axis y
for(uint32_t i = idy; i < h; i += step_y){
// run for axis x
for(uint32_t j = idx; j < w; j += step_x){
// init very big num
float min = INT32_MAX;

uint32_t pixel = picture[i*w + j];
uint8_t ans_c = 0;

for(uint8_t c = 0; c < classes; ++c){
float red = RED(pixel);
float green = GREEN(pixel);
float blue = BLUE(pixel);
float metric = 0.0;

red -= computation_data[c].avg_red;
green -= computation_data[c].avg_green;
blue -= computation_data[c].avg_blue;

float temp_red = red*computation_data[c].cov11 +
```

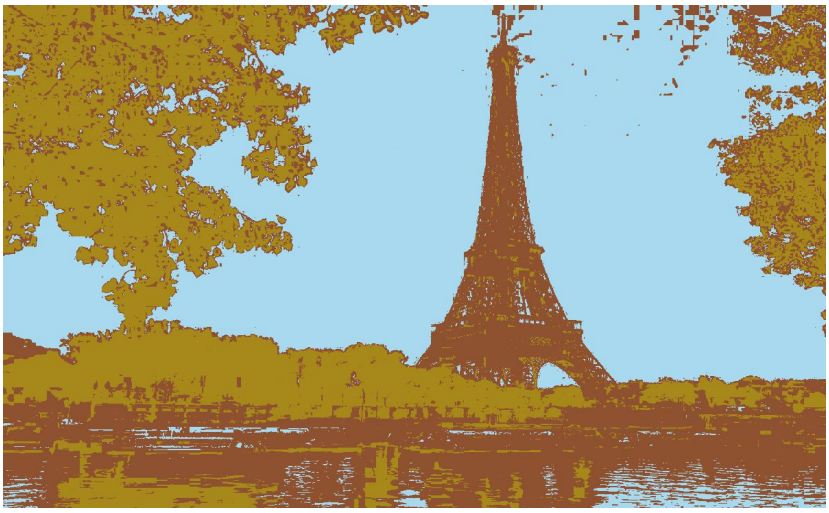
```
green*computation_data[c].cov21 + blue*computation_data[c].cov31;

float temp_green = red*computation_data[c].cov12 +
green*computation_data[c].cov22 + blue*computation_data[c].cov32;

float temp_blue = red*computation_data[c].cov13 +
green*computation_data[c].cov23 + blue*computation_data[c].cov33;

// dot + log(|covl)
metric = (temp_red*red + temp_green*green + temp_blue*blue + computation_data[c].log_det);
if(metric < min){
ans_c = c;
min = metric;
}
}

#ifndef __WITH_IMG__
pixel ^= ((uint32_t) ans_c) << 24;
#endif

picture[i*w + j] = pixel;
}
}
}
```

После вызова kernel я копирую данные в массив и освобождаю выделенную память.

Результаты

Для иллюстрации результатов работы алгоритма я выбрал 2 изображения:

После чего я применил конвертер в заданный в задании формат, который я взял из материалов от преподавателя. После чего я применил к ним свою программу, только вместо номера класса на место alpha канала я записываю avg по наиболее вероятному классу. Полученные результаты:



В обоих изображениях я выделял несколько точек из трех классов. На лицо некоторые неточности, которые устраняются с добавлением новых классов и новых точек для них.

Таблица замеров времени выполнения

CPU или конфигурация	Размер изображения	Время выполнения
CPU	800 1200	99.141
CPU	1600 2560	290.946
<<<(4, 4), (4, 4)>>>	800 1200	7.38314
<<<(4, 4), (4, 4)>>>	1600 2560	29.7029
<<<(4, 4), (16, 16)>>>	800 1200	1.37098
<<<(4, 4), (16, 16)>>>	1600 2560	5.76128
<<<(4, 4), (32, 32)>>>	800 1200	1.22864
<<<(4, 4), (32, 32)>>>	1600 2560	5.04371
<<<(16, 16), (4, 4)>>>	800 1200	5.1625
<<<(16, 16), (4, 4)>>>	1600 2560	21.9541
<<<(16, 16), (16, 16)>>>	800 1200	2.1607
<<<(16, 16), (16, 16)>>>	1600 2560	6.9639
<<<(16, 16), (32, 32)>>>	800 1200	1.24474
<<<(16, 16), (32, 32)>>>	1600 2560	4.74365
<<<(32, 32), (4, 4)>>>	800 1200	5.01814
<<<(32, 32), (4, 4)>>>	1600 2560	21.536
<<<(32, 32), (16, 16)>>>	800 1200	1.11478
<<<(32, 32), (16, 16)>>>	1600 2560	4.46362
<<<(32, 32), (32, 32)>>>	800 1200	1.50403
<<<(32, 32), (32, 32)>>>	1600 2560	5.12384