МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Факультет «Информационные технологии и прикладная математика»
Кафедра «Вычислительная математика и программирование»

Лабораторная работа №3
по курсу «Программирование графических процессоров»

Классификация и кластеризация изображений на GPU.

Студент: Лысенко Д.А.
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2019

Условие

Вариант 5. Метод k-средних.

Входные данные. На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. На следующей строке, число nc -- кол-во кластеров. Далее идут nc строчек описывающих начальные центры кластеров. Каждая i-ая строчка содержит пару чисел -- координаты пикселя который является центром.

$nc \leq 32$.

Программное и аппаратное обеспечение

Compute capability: 5.0
Name: GeForce GTX 960M
Total Global Memory: -2147483648
Shared memory per block: 49152
Registers per block: 65536
Warp size: 32
Max threads per block: (1024, 1024, 64)
Max block : (2147483647, 65535, 65535)
Total constant memory: 65536
Multiprocessors count: 5
Процессор (CPU) – Intel Core i5-6300HQ 2.30GHz; 8 ГБ RAM;
OS Microsoft Windows 10 (x64)
CUDA V10.1

Метод решения

Для реализации метода к-средних мы выделяем константную память, в которую записываем текущие центры кластеров. Далее мы запускаем ядро, которое параллельно вычисляет для всех пикселей расстояние до центров каждого кластера и относит каждый пиксель к кластеру с наименьшим расстоянием. Далее мы пересчитываем центры кластеров и запускаем ядро заново. Программа прекращает свою работу, когда центры кластеров перестают изменятся.

Описание программы

Ядро программы, где происходит классификация пикселей изображения.

```
__global__ void KMeans(uchar4 * __restrict__ img, const uint32_t w, const uint32_t h, const uint32_t nc) {
    uint32_t idx = threadIdx.x + blockIdx.x * blockDim.x;
    uint32_t idy = threadIdx.y + blockIdx.y * blockDim.y;
    uint32_t offsetx = blockDim.x * gridDim.x;
    uint32_t offsety = blockDim.y * gridDim.y;
    for (uint32_t i = idx; i < w; i += offsetx) {
        for (uint32_t j = idy; j < h; j += offsety) {
            double distanceMin = calculateDistance(img[j * w + i], centerClusters[0]);
            uint32_t clusterNumber = 0;
            for (uint32_t k = 1; k < nc; ++k) {
                double distanceTmp = calculateDistance(img[j * w + i], centerClusters[k]);
                if (distanceTmp < distanceMin) {
                    distanceMin = distanceTmp;
                    clusterNumber = k;
                }
            }
```

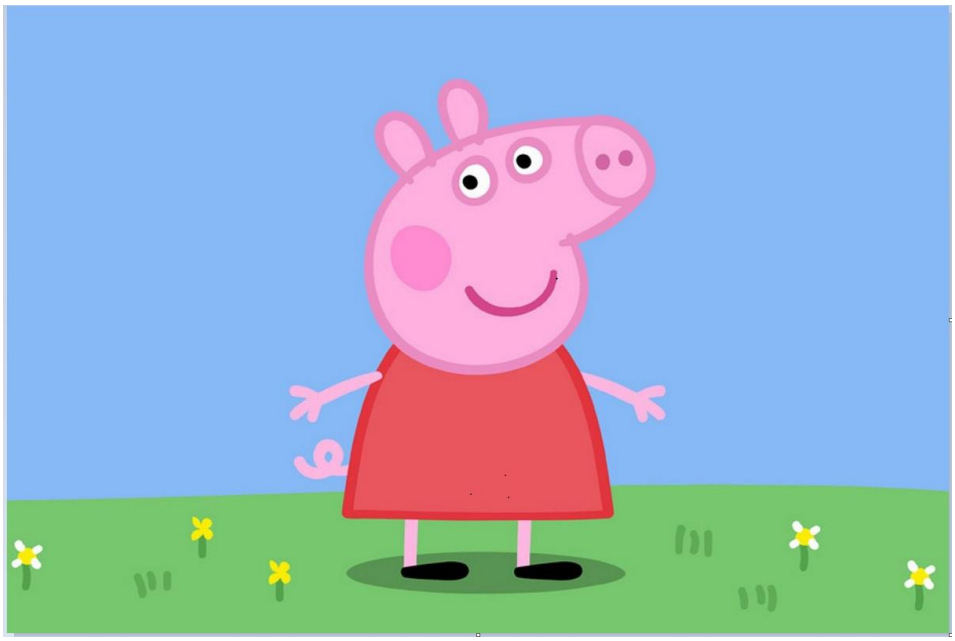
```
            img[j * w + i].w = clusterNumber;
        }
    }
}
```

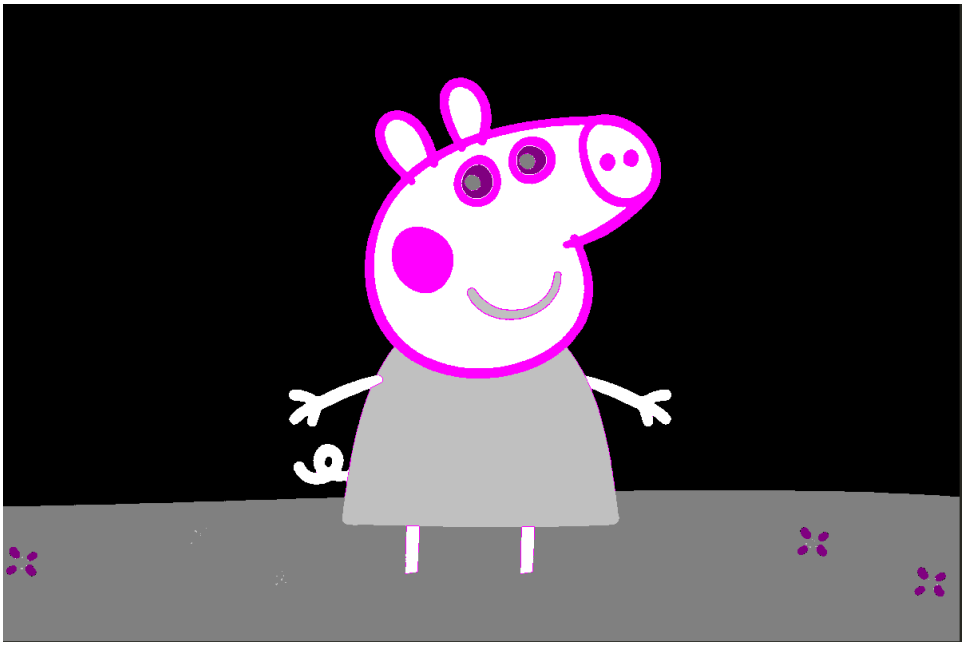
Пересчет центров кластеров:

```
double3 * __restrict__ centerClustersHost, const uint32_t w, const uint32_t h, const uint32_t
nc) {
    uint64_t countElementOnCluster[32] = { 0 };
    ulonglong3 sumElementOnCluster[32] = { make_ulonglong3(0, 0, 0) };
    bool notEqual = false;
    for (uint32_t i = 0; i < w*h; ++i) {
        if (imgNew[i].w != img[i].w) notEqual = true;
        countElementOnCluster[imgNew[i].w]++;
        sumElementOnCluster[imgNew[i].w].x += imgNew[i].x;
        sumElementOnCluster[imgNew[i].w].y += imgNew[i].y;
        sumElementOnCluster[imgNew[i].w].z += imgNew[i].z;

    }
    for (uint32_t i = 0; i < nc; ++i) {
        centerClustersHost[i].x = (double)sumElementOnCluster[i].x / (double)countElementOn-
Cluster[i];
        centerClustersHost[i].y = (double)sumElementOnCluster[i].y / (double)countElementOn-
Cluster[i];
        centerClustersHost[i].z = (double)sumElementOnCluster[i].z / (double)countElementOn-
Cluster[i];
    }
    return notEqual;
}
```

Тест





Оценка производительности

Количество кластеров	CUDA <(8,8),(8,8)> TIME	CUDA <(16,16),(16,16)> TIME	CUDA <(32,32),(32,32) > TIME	CPU TIME
4	138.18	131.10	135.83	392
16	417.71	404.13	408.75	1688

Вывод

Алгоритм к-средних является достаточно простым в реализации, мы просто относим пиксели к кластерам с наименьшим расстоянием, пересчет кластеров происходит в параллельном режиме т.к. при запуске в параллельном режиме нам приходится записывать информацию, необходимую для пересчета центров кластеров в одно место памяти, что приводит к конфликтам и прироста производительности не наблюдается.