МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №3
по курсу «Программирование графических процессоров»

Классификация и кластеризация изображений на GPU

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

Москва, 2020

Условие

Цель работы: Научиться использовать GPU для классификации и кластеризации изображений. Использование константной памяти.
Вариант № 2 "Метод расстояния Махаланобиса"

Программное и аппаратное обеспечение

GPU:

1. Compute capability: 3.0
2. Графическая память: 2147155968
3. Разделяемая память: 49152
4. Константная память: 65536
5. Количество регистров на блок: 65536
6. Максимальное количество блоков: (65535, 65535, 65535)
7. Максимальное количество нитей: (1024, 1024, 64)
8. Количество мультипроцессоров: 6

Сведения о системе:

1. Процессор: Intel Core i7-Q720 1.60GHz
2. Память: 8 ГБ
3. HDD: 465 ГБ

Программное обеспечение:

1. OS: Windows 7
2. IDE: Visual Studio 2019
3. Компилятор: nvcc

Метод решения

Для некоторого пикселя p , номер класса jc определяется следующим образом: $jc = \arg\max_j \left[-(p - avg_j)^T * cov_j^{-1} * (p - avg_j)\right]$,где

$$avg_j = \frac{1}{np_j} \sum_{i=1}^{np_j} p * s_i^j$$

$$cov_j = \frac{1}{np_j - 1} \sum_{i=1}^{np_j} \left(p * s_i^j - avg_j\right) * \left(p * s_i^j - avg_j\right)^T$$

$p * s_i^j$ это $i-$ый пиксель из $j-$ой выборки

Avg и cov для каждого класса были вычислены на CPU и помещены в константную память, после на GPU производился поиск класса для каждого пикселя.

Описание программы

```
struct Image{
    int w;
    int h;
    uchar4* pixels;
    Image() {
        w = 0;
        h = 0;
        pixels = nullptr;
    }

    uchar4 GetPixel(int x, int y) {
        return pixels[x + y * w];
    }

    void ReadFromFile(string &input){

        int width;
        int height;
        ifstream inputFile(input, std::ios::in | std::ios::binary);

        if (inputFile.is_open()) {
            if (!inputFile.read((char*)&width, sizeof(width))) {
                cerr << "ERROR: can't read from file " << __LINE__ << endl;
                abort();
        }
        if (!inputFile.read((char*)&height, sizeof(height))) {
            cerr << "ERROR: can't read from file " << __LINE__ << endl;
            abort();
        }

        w = width;
        h = height;
        pixels = new uchar4[width * height];

        if (!inputFile.read((char*)pixels, width * height * sizeof(uchar4))) {
                cerr << "ERROR: can't read from file " << __LINE__ << endl;
                abort();
            }
            inputFile.close();
        }
        else {
            cerr << "ERROR: can't open file " << __LINE__ << endl;
            abort();
        }
    }
```

```
    void WriteToFile(string &output){

        std::ofstream outputFile(output, std::ios::out | std::ios::binary);

        if (outputFile.is_open()) {
            if (!outputFile.write((char*)&w, sizeof(w))) {
                cerr << "ERROR: can't open write " << __LINE__ << endl;
                abort();
            }
            if (!outputFile.write((char*)&h, sizeof(h))) {
                cerr << "ERROR: can't open write " << __LINE__ << endl;
                abort();
            }
            if (!outputFile.write((char*)pixels, w * h * sizeof(uchar4))) {
                cerr << "ERROR: can't open write " << __LINE__ << endl;
                abort();
            }
            outputFile.close();
        }
        else {
            cerr << "ERROR: can't open file " << __LINE__ << endl;
            abort();
        }
    }

    void Delete() {
        delete[] pixels;
        w = 0;
        h = 0;
        pixels = nullptr;
    }
};

__constant__ double g_avgs[32][3];
__constant__ double g_inv_covs[32][3][3];


__device__ double classifierFunc(uchar4* pixel, int classInd){

    double diff[3];
    double res[3];

    diff[0] = (double) pixel->x - g_avgs[classInd][0];
    diff[1] = (double) pixel->y - g_avgs[classInd][1];
    diff[2] = (double) pixel->z - g_avgs[classInd][2];

    res[0] = 0.0;
    res[1] = 0.0;
    res[2] = 0.0;
```

```
    for (int j = 0; j < 3; ++j) {
        for (int k = 0; k < 3; ++k) {
            res[j] += diff[k] * g_inv_covs[classInd][k][j];
        }
    }

    double answer = 0.0;

    for (int k = 0; k < 3; ++k) {
        answer += res[k] * diff[k];
    }
    answer *= -1.0;

    return answer;
}


__global__ void kernel(uchar4* pixels, int size, int PGP_NC) {

    int idX = blockIdx.x * blockDim.x + threadIdx.x;
    int offsetX = gridDim.x * blockDim.x;

    for (int i = idX; i < size; i = i + offsetX) {
        uchar4* pixel = &pixels[i];
        double maxVal = classifierFunc(pixel, 0);
        int maxInd = 0;

        for (int c = 1; c < PGP_NC; ++c){

            double tmpVal = classifierFunc(pixel, c);

            if (tmpVal > maxVal){
                maxVal = tmpVal;
                maxInd = c;
            }
        }

        pixel->w = (unsigned char) maxInd;
    }

}
```

Результаты

CPU

w * h	10^6	10^7	10^7	10^8	10^8
nc	32	16	32	8	16
Time (sec)	3.083	16.134	30.973	82.243	159.975

GPU <<< (1, 32)>>>

w * h	10^6	10^7	10^7	10^8	10^8
nc	32	16	32	8	16
Time (mil sec)	394	2049	3939	11053	20496

GPU <<< (1, 64)>>>

w * h	10^6	10^7	10^7	10^8	10^8
nc	32	16	32	8	16
Time (mil sec)	199	1031	1986	5544	10318

GPU <<< (32, 32)>>>

w * h	10^6	10^7	10^7	10^8	10^8
nc	32	16	32	8	16
Time (mil sec)	39	200	393	1042	1999

GPU <<< (32, 64) >>>

w * h	10^6	10^7	10^7	10^8	10^8
nc	32	16	32	8	16
Time (mil sec)	34	175	348	892	1754

GPU <<< (64, 32) >>>

w * h	10^6	10^7	10^7	10^8	10^8
nc	32	16	32	8	16
Time (mil sec)	34	174	344	889	1742

GPU <<< (256, 256) >>>

w * h	10^6	10^7	10^7	10^8	10^8
nc	32	16	32	8	16
Time (mil sec)	33	167	331	842	1668

GPU <<< (512, 512) >>>

w * h	10^6	10^7	10^7	10^8	10^8
nc	32	16	32	8	16
Time (mil sec)	33	167	330	849	1682

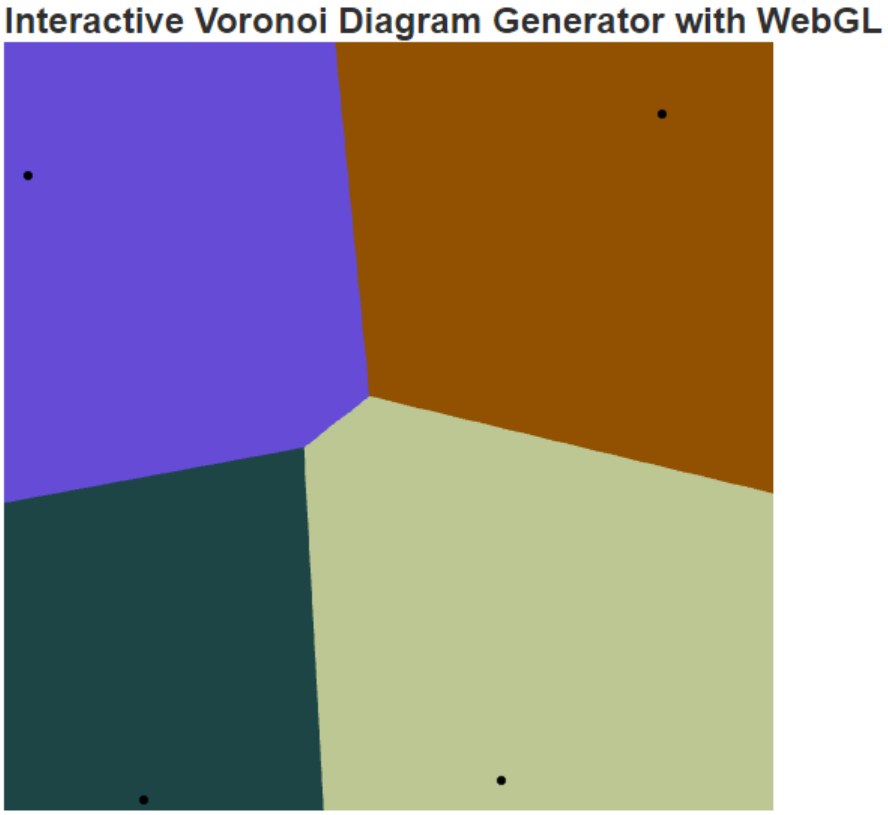
GPU <<< (1024, 1024) >>>

w * h	10^6	10^7	10^7	10^8	10^8
nc	32	16	32	8	16
Time (mil sec)	34	170	338	855	1697

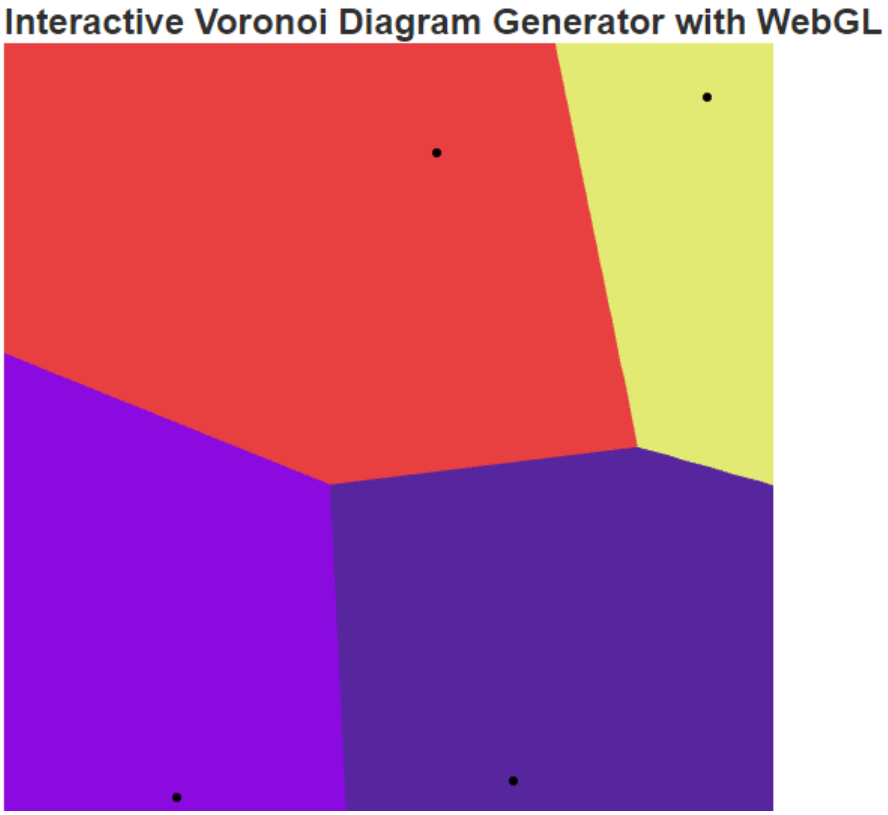
Значение Avg для каждого класса.

```
user19@server-i72:~/lab3/Images$ ./lab3
input.data
copy.data
4
8
840 450 930 480 980 540 971 648 887 663 836 646 800 600 792 542
6
667 179 630 217 511 386 444 471 630 598 898 144
6
390 8   392 615 391 608 391 606 391 604 379 336
6
1500 1200 1500 1300 1500 1100 1600 1300 1600 1200 1600 1100

210.125
2.625
140.875


233.833
235.333
238


1.16667
24.6667
40.8333


1.5
139.167
171.5

user19@server-i72:~/lab3/Images$
```

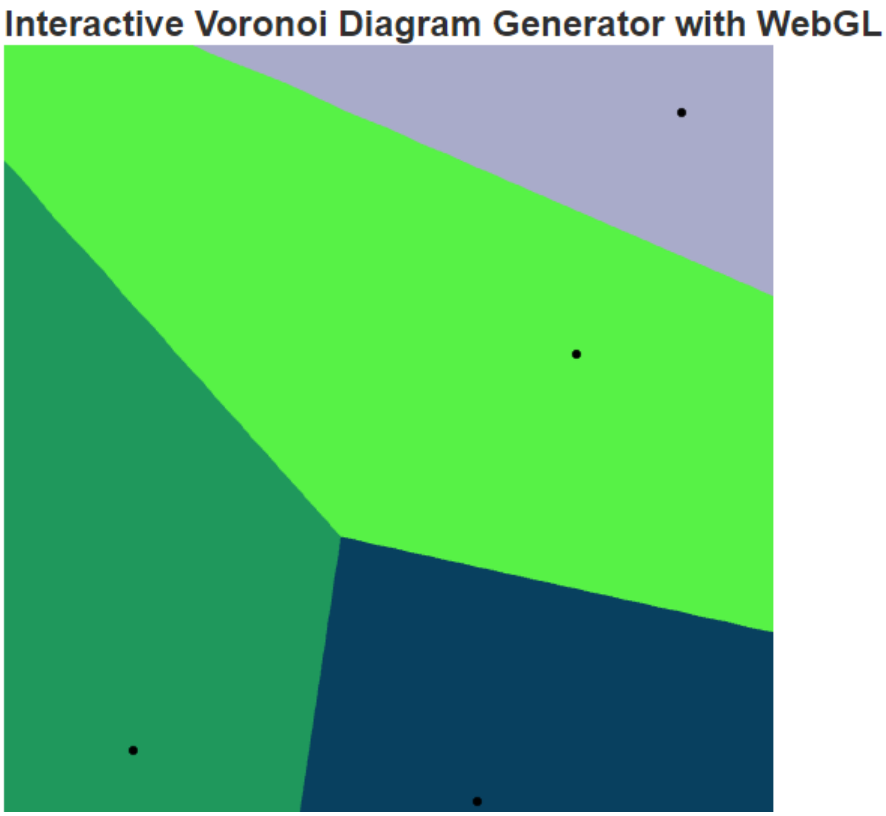
Ординаты Red Абсциссы Green



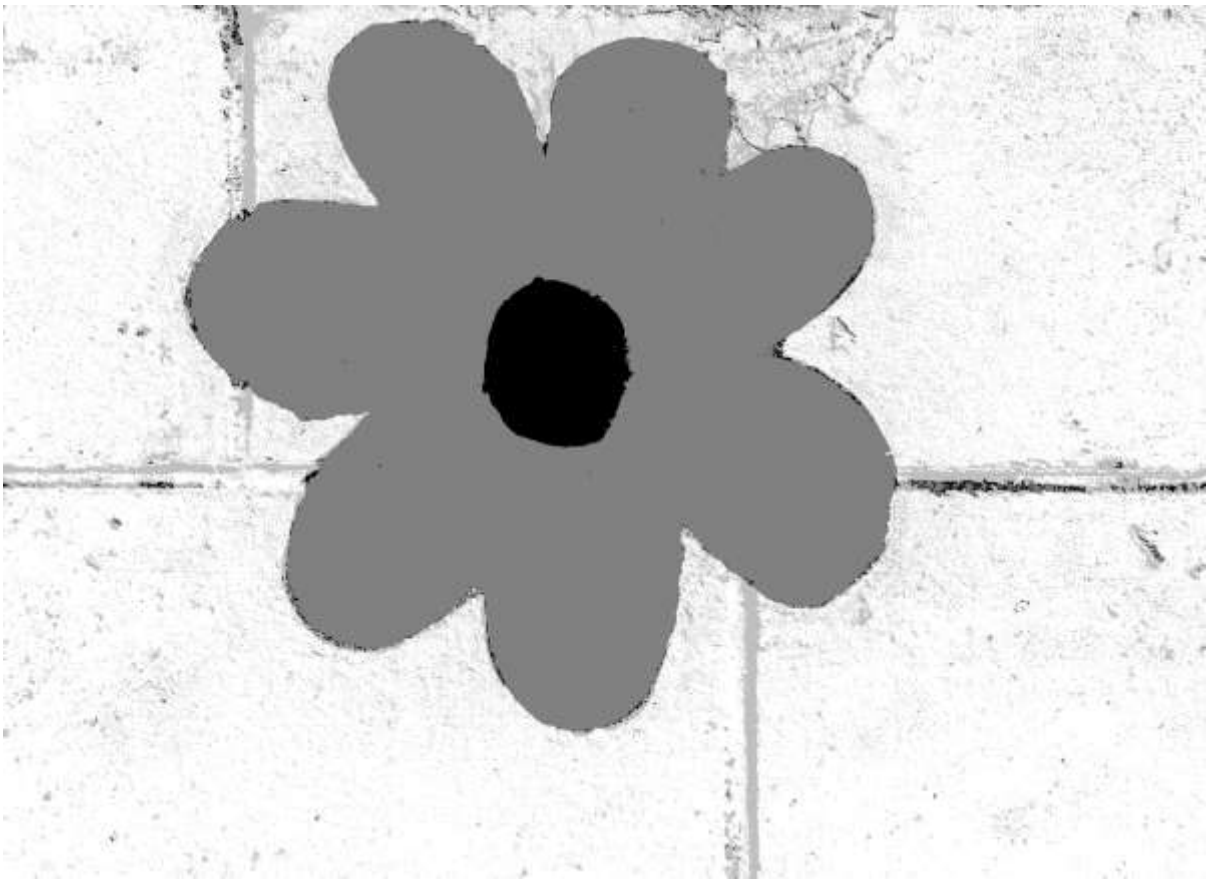
Ординаты Red Абсциссы Blue



Ординаты Green Абсциссы Blue



Результаты работы программы:



Выводы

В результате выполнения лабораторной работы изучил применение метода Махаланобиса для задачи классификации. Результаты классификации методом не самые качественные. Он уступает в качестве методу максимального правдоподобия. Задача классификации очень хорошо распараллеливается, так что не удивительно, что многие Data science инженеры ведут свои расчеты на GPU.