МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»

Институт информатики, математики и электроники
Факультет информатики
Кафедра технической кибернетики

**Отчет по научно-исследовательской работе бакалавра**

**на тему**

# РЕАЛИЗАЦИЯ МЕТОДОВ КЛАСТЕРИЗАЦИИ С ПРИМЕНЕНИЕМ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ МОБИЛЬНЫХ УСТРОЙСТВ

*Выполнил:* Логинов П.В.

*Группа:* 6409

*Научный руководитель:* Суханов С.В.

*Оценка:* _____

*Дата:* _____

Самара 2017

# РЕФЕРАТ

**Отчет по научно-исследовательской работе бакалавра:** 24 с., 5 рисунков, 0 таблиц, 5 источников, одно приложение.

СЕГМЕНТАЦИЯ ИЗОБРАЖЕНИЙ, МЕТОДЫ КЛАСТЕРИЗАЦИИ, МЕТОД КЛАСТЕРИЗАЦИИ k-СРЕДНИХ, K-MEANS, МЕТОД ОТЦУ, OPENCL, ANDROID

Объектом исследования является реализация алгоритмов кластеризации изображений с помощью OpenCL на мобильных устройствах в среде операционной системы Android.

Цель работы – изучение особенностей реализации метода k-средних с помощью OpenCL на мобильных устройствах в среде операционной системы Android.

Рассмотрены особенности работы программы, использующей OpenCL-инструкции, на центральном и графическом процессорах мобильных устройств под управлением операционной системы Android и сделан вывод о том, что работа рассматриваемого алгоритма на мобильных устройствах возможна.

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Мобильные устройства имеют значительные аппаратные ограничения производительности вследствие небольших габаритов и требований к автономности, не позволяющих установку эффективной системы охлаждения и значительного потребления энергии, то есть не позволяющих использование высокопроизводительных компонентов. Такие задачи, как распознавание объектов в видеопотоке в реальном времени, семантический анализ текста, быстрый рендеринг трехмерной графики, представляют сложность для процессоров мобильных устройств, хотя являются очень востребованными для задач бизнеса. В связи с этим сообщество разработчиков стремится использовать эти вычислительные ресурсы максимально эффективно.

В данной работе была поставлена задача исследовать возможность реализации параллельных вычислений на центральном и графическом процессорах мобильных устройств. Для исследования была выбрана операция сегментации изображений с применением метода кластеризации k-средних.

# 1 ОБЗОР МЕТОДА КЛАСТЕРИЗАЦИИ K-СРЕДНИХ

## 1.1 Постановка задачи кластеризации

Кластеризация (классификация) - это разделение множества входных векторов на группы (кластеры, классы) по степени схожести друг с другом. Формальная постановка задачи выглядит следующим образом:

- $X = \{x_1, \dots, x_n\}$ – конечная выборка объектов;
- $Y = \{y_1, \dots, y_m\}$ – множество кластеров;
- $r = r(x_i, x_j)$ – метрика на выборке $X$ (функция, определяющая расстояние между элементами выборки).

Задача: каждому элементу выборки $X$ поставить в соответствие один кластер из множества $Y$ так, чтобы минимизировать $r$ для объектов каждого кластера. Эта задача является не разрешимой однозначно и сильно зависит от входных данных.

## 1.2 Метод k-средних

Метод k-средних (k-means) – метод кластеризации, целью которого является присоединение объектов к кластеру таким образом, чтобы минимизировать суммарное среднеквадратичное отклонение внутри кластера, вычисляемое следующим образом [1]:

$$\sigma = \sum_{i=1}^{k} \sum_{x_j \in Y_i} (x_j - c_i)^2,$$

где $c_i$ – координата центра кластера $Y_i$.

Пошаговое описание алгоритма метода k-средних:

1. Случайным образом выбираются центры кластеров.
2. Для каждого объекта вычисляются расстояния до центров кластеров. Объект присоединяется к ближайшему кластеру.
3. Вычисляются новые центры кластеров как центр области, в которую вписан кластер.

4. Шаги 2-3 повторяются, пока центры кластеров не перестанут меняться, т.е. пока алгоритм не сойдется.

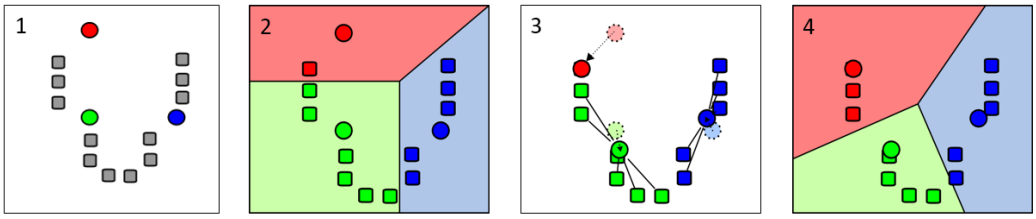


Рисунок 1 – Графическая демонстрация алгоритма

На рисунке 1 представлена визуализация работы алгоритма.

Вычислительная сложность алгоритма - $O(nkI)$, где $n$ – размер входных данных, $k$ – количество кластеров, $I$ – количество итераций до схождения алгоритма.

Преимущества метода:

- всегда сходится локально;
- прост в реализации.

Недостатки:

- не гарантируется глобальная сходимость;
- оптимальный выбор первого приближения происходит случайно;
- необходимо знать число кластеров до выполнения кластеризации;
- алгоритм не справится с ситуацией, когда объект может принадлежать нескольким кластерам.

Существует множество модификаций метода k-средних:

- k-means++ – начальные координаты центров векторов выбираются не случайно, а с вероятностями, пропорциональными квадратам расстояния между центрами.
- MiniBatch – вычисления производятся не для каждого элемента входной выборки, а для некоторых усредненных элементов. Используется, когда входные данные слишком большого объема.
- c-means – для каждого объекта вычисляет принадлежность к кластерам с некоторой вероятностью.

### 1.3 Применение метода k-средних для сегментации изображений

Метод k-средних позволяет сегментировать изображение на несколько кластеров [2]. Объектами исходной выборки в таком случае будут считать пиксели. Каждый пиксель может быть описан пятью числами: двумерными координатами в сетке изображения и значениями трех каналов цветовой модели RGB: $p = (x, y, R, G, B)$.

В таком случае метрика может быть обозначена следующим образом:

$$r(x_i,\ x_j) = \{d = x_j - x_i\} = \|d\| = \sqrt{{d_x}^2 + {d_y}^2 + {d_r}^2 + {d_g}^2 + {d_b}^2}.$$

Данная метрика является евклидовой мерой в пятимерном пространстве.

# 2 ОБЗОР OpenCL

OpenCL - это фреймворк для создания параллельных программ, особенностью которого является гомогенность, позволяющая выполнять одну программу практически на любом вычислительном устройстве (на центральном и графическом процессорах, однокристальных системах, суперкомпьютерах с распределенными вычислительными узлами и других устройствах) и распределять вычисления между устройствами в вычислительных системах. OpenCL обеспечивает параллелизм на уровнях инструкций и данных [3].

Фреймворк включает в себя язык программирования OpenCL C, являющийся диалектом языка C и основанный на стандарте C99, и интерфейс прикладного программирования (API). OpenCL распространяется по свободной лицензии.

Разработка фреймворка была начата компаниями Apple и AMD, а в 2008 году контроль за разработкой и поддержкой был передан консорциуму Khronos Group, и участие в них стали принимать также Intel, Nvidia, Sony и другие крупные компании. OpenCL используется для ускорения вычислительных операций во многих областях: в обработке изображений и видеопотока, компьютерном зрении, графических движках и т. д.

## 2.1 Структура

Ключевыми элементами архитектуры OpenCL являются *хост (host)* и *исполняющее устройство (device)*. Хост - это устройство, контролирующее выполнение OpenCL-инструкций и осуществляющее взаимодействие с клиентским кодом. В вычислительной системе хост может существовать только в единственном экземпляре. Исполняющее устройство - это абстракция для устройства, выполняющего OpenCL-инструкции. Исполняющие устройства содержат внутри себя *вычислительные модули*

*(compute units)*, в которые входят *обрабатывающие элементы (processing elements)*.

Типичными примерами исполняющих устройств OpenCL являются центральные процессоры, графические процессоры и аппаратные модули, такие как процессоры обработки сигналов. На рисунке 1 изображена структурная схема платформы OpenCL.

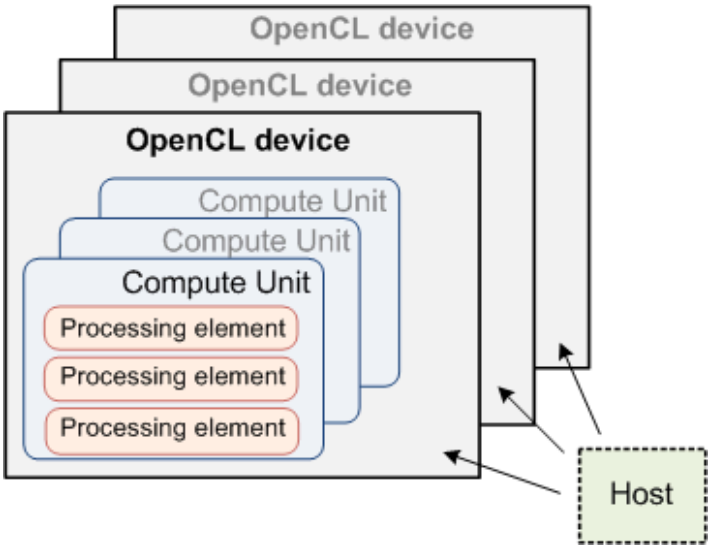


Рисунок 2 - Представление платформы OpenCL

### 2.2 Организация вычислений

Программа хоста пишется на языках C и C++ и описывает вызовы функций, включенных в OpenCL API, для обмена данными с исполняющим устройством посредством объектов OpenCL: устройств, очередей, буферов и т.д.

Программа исполняющего устройства пишется на языке OpenCL C. Программа может работать с несколькими вычислительными модулями. Программа исполняющего устройства может загружаться в текстовой форме и компилироваться во время выполнения или загружаться в скомпилированном виде в память через программу хоста. Программа исполняющего устройства называется *ядром (kernel)*.

Входные и выходные данные для работы вычислительных модулей находятся в буферной памяти.

Каждый обрабатывающий элемент вычислительного модуля работает для некоторой точки итерационного пространства, имеющей уникальный идентификатор. Итерационное пространство в OpenCL представляет собой одно-, дву- или трехмерную сетку NDRange. Например, при обработке растрового изображения используется двумерная сетка, каждый элемент которой связан с одним конкретным пикселем этого изображения. Экземпляр ядра для одного элемента итерационного пространства называется *рабочим элементом (work-item)*. Множество рабочих элементов разбивается на *рабочие группы (work-group)*. Все рабочие элементы, включенные в одну группу, гарантированно выполняются параллельно. На рисунке 2 изображен пример разбиения итерационного пространства размером 15x15 рабочих элементов на рабочие группы размером 5x5.

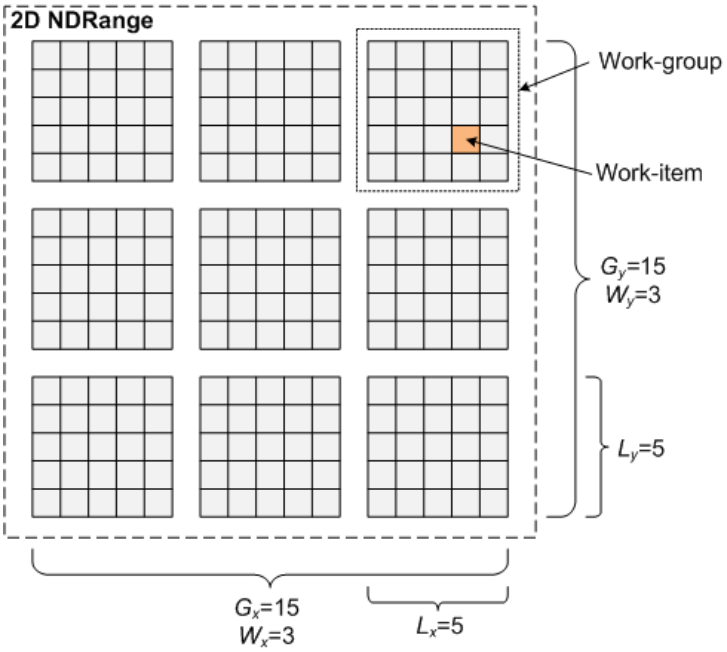


Рисунок 3 - Схема двумерного NDRange

Взаимодействие между хостом и исполняющим устройством OpenCL происходит посредством команд, помещенных хостом в *очередь команд (command queue)*. Для каждого исполняющего устройства создается отдельная очередь.

### 2.3 Организация памяти

За работу с памятью в OpenCL отвечают *объекты памяти*, инкапсулирующие в себе регионы памяти. В программе хоста структуры данных должны быть преобразованы в объекты памяти, а в ядрах можно получать доступ к ним посредством указателей.

Данные представляют собой либо массив (буфер), либо изображение. Обработка этих данных в ядре происходит по-разному. Важно учитывать, что на графических процессорах корректно обрабатываться могут только квадратные изображения, т.е. изображения, обладающие одинаковым размером по обеим осям.

### 2.4 Особенности работы OpenCL в Android-приложениях

Серьезным ограничением для мобильных устройств на базе операционной системы Android является тот факт, что приложения выполняются в виртуальной машине. Это дает преимущества в безопасности и простоте и скорости разработки, но очень сильно урезает вычислительные способности устройства. Для решения этой проблемы компания Google разрешила разработчикам использовать инструкции, выполняемые прямо на процессорах устройств, представив в 2009 году Android Native Development Kit [4].

На данный момент почти все Android-устройства имеют выделенный графический процессор, помимо центрального, и содержат в системе реализацию OpenCL. А для устройств на базе процессоров Intel Atom компания Intel даже представила SDK. Использование OpenCL для мобильной разработки позволяет значительно ускорить вычисления на мобильном устройстве.

На рисунке 3 показана высокоуровневая архитектура приложения OpenCL для Android. В таком приложении должны присутствовать две части: основная, выполняемая в виртуальной машине Android (Dalvik или ART), и нативная, выполняемая непосредственно на устройстве. Основная часть

управляет пользовательским интерфейсом, загружает текст программы OpenCL из файла, компилирует ее и вызывает функцию из нативной части для выполнения фактической работы программы OpenCL. Связь между основной и нативной частями выполняется через JNI (Java Native Interface), что характерно для любого приложения Android, которое включает в себя нативные модули.

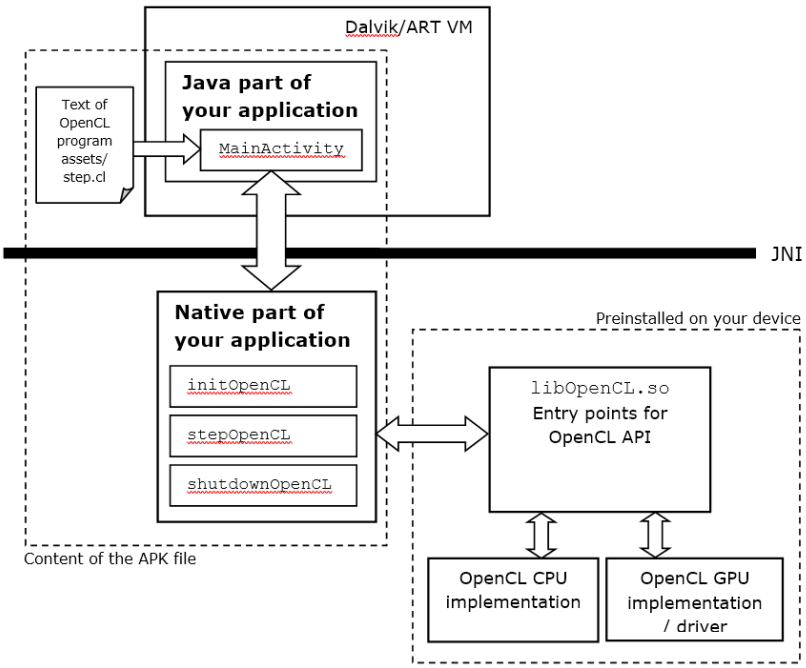


Рисунок 4 - Схема архитектуры OpenCL-приложения для Android

Нативная часть работает с библиотекой libOpenCL.so, которая передает вызовы OpenCL API реализации OpenCL для конкретного устройства, например, CPU или GPU. Библиотека libOpenCL.so должна находиться на целевом устройстве до того, как приложение будет развернуто. Как правило, на современных устройствах с операционной системой Android эта библиотека входит в стандартную поставку и передает вызовы как CPU, так и GPU.

Нативная часть состоит из следующих частей:

- initOpenCL - для инициализации объектов OpenCL;
- stepOpenCL - для коммуникации между хостом и исполняющими устройствами;

- shutdownOpenCL - для сброса объектов OpenCL.

# 3 РЕАЛИЗАЦИЯ АЛГОРИТМА СЕГМЕНТАЦИИ ИЗОБРАЖЕНИЯ

Для решения задачи распознавания отдельных объектов на изображении необходимо предпринять следующие шаги:

1. Привести изображение к монохромному (черно-белому) виду. Это легко сделать, изменив значения цвета каждого пикселя на среднее арифметическое значение трех каналов RGB:
$$c = (c_r + c_g + c_b)/3.$$
2. Разбить пиксели монохромного изображения на два класса яркости (полезные и фоновые) с применением метода Отцу [5]. Цель этого метода - вычисление порога яркости $t$ таким образом, чтобы минимизировать дисперсию яркости пикселей внутри классов, т.е. максимизировать дисперсию между классами. Дисперсия может быть описана следующим выражением
$$\sigma_b^2(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2,$$
где $\omega_1(t)$ и $\omega_2(t)$ – вероятности двух классов, разделенных порогом $t$, $\mu_1(t)$ и $\mu_2(t)$ – средние арифметические значения яркости в классе. Эти значения могут пересчитываться итеративно, и будут выражены при этом рекурсивно через значения на предыдущей итерации. Нобуюки Отцу предложил быстрый алгоритм максимизации, который сводится к нахождению максимального значения $\sigma_b^2(t)$. Результатом работы будет величина порога. Далее входное изображение можно разбить на две части, отнеси пиксели с яркостью ниже порога $t$ к одному классу, с яркостью выше – к другому.
3. Провести кластеризацию пикселей полученных на предыдущем шаге классов и определить отдельные объекты, к которым они принадлежат, с помощью метода k-средних.

Разбиение на классы яркости применяется в предположении, что в большинстве случаев пиксели разных классов относятся к разным объектам. Вообще говоря, это неверно, но может оказаться эффективным для изображений с равномерным освещением. Кроме того, метод Отцу требует дополнительных вычислений, что уменьшает производительность. Таким образом, шаги 1 и 2 можно опустить для обеспечения лучшей производительности и провести кластеризацию исходной сетки пикселей.

Была разработана программа, сегментирующая изображения обоими описанными способами. Программа состоит из двух частей: хоста (консольное Win32-приложение, написанное на языке C++) и ядра. Исходный код представлен в приложении А.

Результаты работы программы представлены на рисунке 5. Каждое изображение сегментировано на 10 кластеров, обозначенных разными цветами.

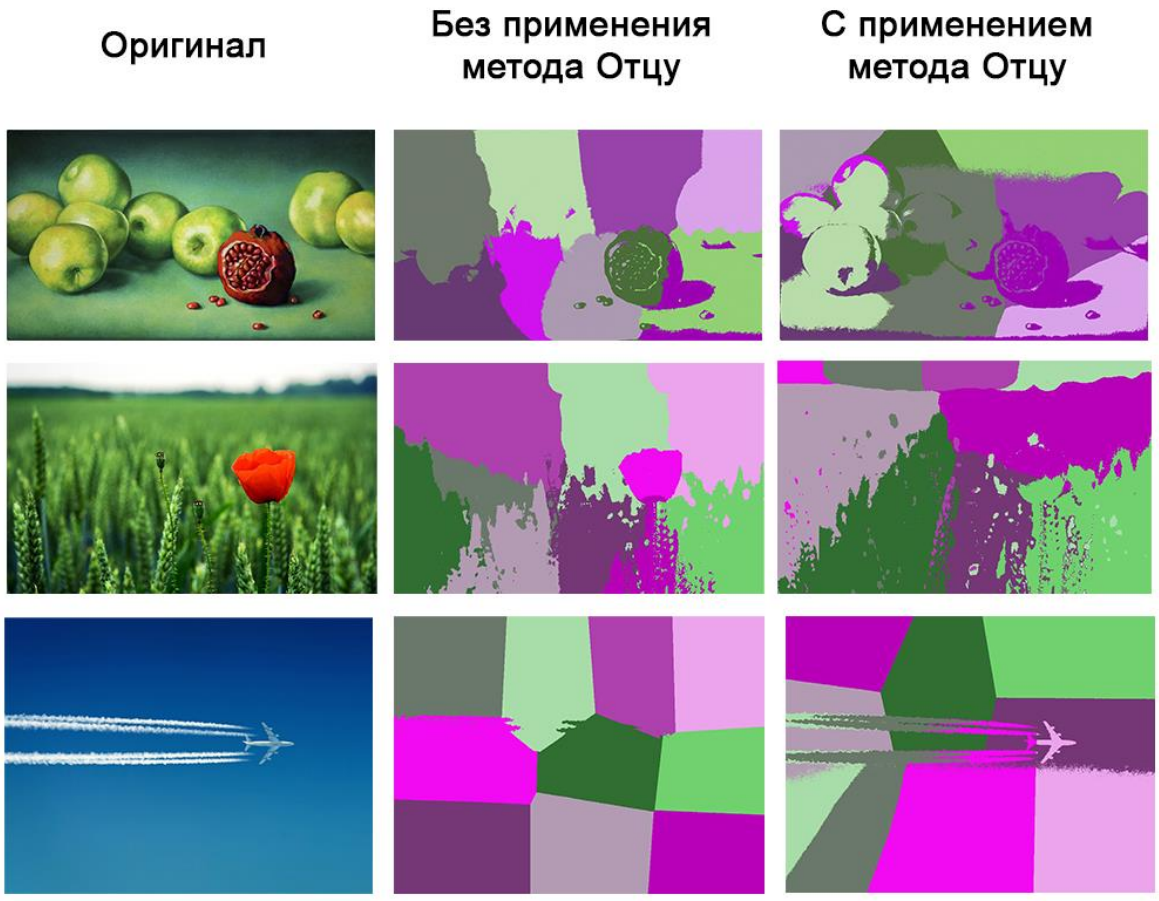


Рисунок 5 – Результаты обработки изображений

Как видно, сегментация с применением метода Отцу для разбиения на классы яркости и без него дают различные результаты:

1. На изображении преобладает светло-зеленый цвет, на котором выделяется красный объект - гранат. При сегментации без применения метода Отцу теряются контуры всех фоновых объектов, а гранат попадает сразу в два кластера. Во втором случае гранат со своей тенью попадает в отдельный кластер, и контуры фоновых объектов не теряются.
2. Аналогично с первым изображением, на светло-зеленом фоне находится ярко-красный объект. Однако в этом случае точнее срабатывает алгоритм без применения метода Отцу – бутон красного цветка попадает в отдельный кластер, тогда как во втором случае результат получается нерелевантным. Это происходит потому, что на первом изображении контрастируют яркие и неяркие объекты (тени), и они легко подвергаются разделению на классы, а на втором преобладают светлые (яркие) пиксели.
3. Сегментация с разбиением по методу Отцу выделяет самолет и его конденсационный след, причем самолет полностью относит к одному кластеру, результат работы без разбиения выдает нерелевантный результат.

# ЗАКЛЮЧЕНИЕ

В данной работе были изучены возможности фреймворка OpenCL, проведено исследование возможности применения метода кластеризации k-средних для сегментации изображений с целью нахождения на них отдельных объектов.

Также были рассмотрены особенности работы программы, использующей OpenCL-инструкции, на центральном и графическом процессорах мобильных устройств под управлением операционной системы Android и сделан вывод о том, что работа рассматриваемого алгоритма на мобильных устройствах возможна.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Coates A. Learning Feature Representations with k-means //Neural networks: Tricks of the trade. – Berlin, Heidelberg: Springer, 2012. – P. 561-580.

2 Ту Дж. Принципы распознавания образов / Дж. Ту, Р. Гонсалес. – М.: Мир, 1978. – 414 с.

3 OpenCL Programming Guide / Aaftab Munshi et al. – Addison-Wesley, 2011. – 648 p.

4 Cinar O. Getting Started with C++ on Android //Pro Android C++ with the NDK. – Apress, 2012. – P. 1-39.

5 Otsu N. A threshold selection method from gray-level histograms //IEEE transactions on systems, man, and cybernetics, 1979. – Т. 9 (1). – P. 62-66.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

## Код ядра:

```
__kernel void grayscale(__read_only image2d_t input, __write_only image2d_t output, sampler_t _sampler) {
	int2 currentPosition = (int2)(get_global_id(0), get_global_id(1));
	uint4 currentPixel;
	uint4 calculatedPixel;
	currentPixel = read_imageui(input, _sampler,  currentPosition);
	calculatedPixel = currentPixel.x * 0.07 + currentPixel.y * 0.72 + currentPixel.z * 0.21;
	write_imageui(output, currentPosition, calculatedPixel);
}
inline int get_random(int *m_z, int *m_w)
{
	(*m_z) = ((*m_z) & 512) + ((*m_z) >> 6);
	(*m_w) = ((*m_w) & 512) + ((*m_w) >> 5);
	return ((*m_z) << 16) + (*m_w);
}
__kernel void kmeans(__read_only image2d_t input, __write_only image2d_t output, sampler_t _sampler, unsigned int clustersCount, __global const int2 *centroids, __global int* mask) {
	int id = get_global_id(0);
	int2 currentPosition = (int2)(id, get_global_id(1));
	uint4 currentPixel = read_imageui(input, _sampler, currentPosition);
	uint4 bitmap[512][512];
	for (int x = 0; x < 512; x++) {
		for (int y = 0; y < 512; y++) {
			uint4 pixel = read_imageui(input, _sampler, (int2)(x, y));
			bitmap[x][y] = pixel;
		}
	}
	for (int x = 0; x < 512; x++) {
		for (int y = 0; y < 512; y++) {
			uint4 pixel = bitmap[x][y];
			write_imageui(output, (int2)(x, y), pixel);
		}
	}
	int2 sums_per_cluster[3];
	int pixels_count_per_cluster[3];
```