Оглавление

[Введение. 3](#_Toc453077678)

[Постановка задачи. 4](#_Toc453077679)

[Теоретическая часть 5](#_Toc453077680)

[OpenCL 5](#_Toc453077681)

[Определение 5](#_Toc453077682)

[Программируемая пользователем вентильная матрица (FPGA) 5](#_Toc453077683)

[GPGPU 6](#_Toc453077684)

[Особенности языка 6](#_Toc453077685)

[Предпосылки появления OpenCL 6](#_Toc453077686)

[Решение проблемы 8](#_Toc453077687)

[Как обстоят дела на настоящий момент 9](#_Toc453077688)

[Заключение 10](#_Toc453077689)

[Медианная фильтрация 11](#_Toc453077690)

[Практическая часть. 15](#_Toc453077691)

[Описание алгоритма фильтрации изображения 15](#_Toc453077692)

[Приведение результатов работы программы 18](#_Toc453077693)

[Выводы. 20](#_Toc453077694)

[Список литературы. 21](#_Toc453077695)

# Введение.

# Постановка задачи.

# Теоретическая часть

## OpenCL

### Определение

OpenCL (от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Open Computing Language - открытый язык вычислений) - [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) для написания [компьютерных программ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), связанных с [параллельными вычислениями](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F) на различных [графических](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) GPU) и [центральных процессорах](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BD%D1%82%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) CPU), а также [FPGA](https://ru.wikipedia.org/wiki/FPGA). Вo фреймворк OpenCL входят [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), который базируется на стандарте [C99](https://ru.wikipedia.org/wiki/C99), и [интерфейс программирования приложений](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) API). OpenCL обеспечивает параллелизм на уровне инструкций и на уровне данных и является реализацией техники [GPGPU](https://ru.wikipedia.org/wiki/GPGPU). OpenCL является полностью [открытым стандартом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82), его использование не облагается лицензионными отчислениями.

Цель OpenCL состоит в том, чтобы дополнить [OpenGL](https://ru.wikipedia.org/wiki/OpenGL) и [OpenAL](https://ru.wikipedia.org/wiki/OpenAL), которые являются открытыми отраслевыми стандартами для [трёхмерной компьютерной графики](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D1%91%D1%85%D0%BC%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) и звука, пользуясь возможностями [GPU](https://ru.wikipedia.org/wiki/GPU). OpenCL разрабатывается и поддерживается некоммерческим консорциумом [Khronos Group](https://ru.wikipedia.org/wiki/Khronos_Group), в который входят много крупных компаний, включая [AMD](https://ru.wikipedia.org/wiki/AMD), [Apple](https://ru.wikipedia.org/wiki/Apple), [ARM](https://ru.wikipedia.org/wiki/ARM_%28%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F%29), [Intel](https://ru.wikipedia.org/wiki/Intel), [Nvidia](https://ru.wikipedia.org/wiki/Nvidia), [Sony Computer Entertainment](https://ru.wikipedia.org/wiki/Sony_Computer_Entertainment), [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems) и другие. [1]

### Особенности языка OpenCL

Ключевыми отличиями используемого языка от Си (стандарт ISO 1999 года) являются:

* Отсутствие поддержки указателей на [функции](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), [рекурсии](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F), [битовых полей](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5), массивов переменной длины (VLA), стандартных заголовочных файлов
* Расширения языка для параллелизма: векторные типы, синхронизация, функции для Work-items/Work-Groups
* Квалификаторы типов памяти: \_\_global, \_\_local, \_\_constant, \_\_private
* Иной набор встроенных функций

OpenCL новый стандарт для разработки приложений для гетерогенных систем. Изначально OpenCL задумывался как нечто большее: единый стандарт для написания приложений, которые должны исполняться в системе, где установлены различные по архитектуре процессоры, ускорители и платы расширения. [1]

### История появления OpenCL

Гетерогенные вычислительные системы в основном используются для высокопроизводительных вычислений, таких как моделирование физических процессов, кодирование видео и т.д. Ранее подобные задачи решались с помощью суперкомпьютера либо очень мощной настольной системы. С появлением технологий NVidia CUDA/AMD Stream стало возможным относительно просто писать программы, использующие вычислительные возможности GPU.

Подобные программы создавались и раньше, но именно NVidia CUDA обеспечила рост популярности GPGPU за счет облегчения процесса создания приложений, выполняемых графическим процессором. Первые GPGPU приложения в качестве ядер (kernel в CUDA и OpenCL) использовали шейдеры, а данные запаковывались в текстуры. Таким образом, необходимо было хорошо разбираться в OpenGL или DirectX. Чуть позже появился язык Brook, который упрощал процесс написание программы (на основе этого языка создавалась AMD Stream (в ней используется Brook+)).

Компьютер, на котором производятся вычисления, на аппаратном уровне может иметь процессоры x86, x86-64, Itanium, SpursEngine (Cell), NVidia GPU, AMD GPU, VIA (S3 Graphics) GPU и т.д. Для каждого из этих типов процессов существует свой SDK, свой язык программирования и программная модель. То есть если нужно, чтобы программа расчета какого-либо физического процесса работала на простой рабочей станции, суперкомпьютере BlueGene, или компьютере, оборудованном двумя ускорителями NVidia Tesla - будет необходимо переписывать достаточно большую часть программы, так как каждая из платформ в силу своей архитектуры имеет набор жестких ограничений. Написание одной и той же программы под разные платформы является очень дорогой и трудоемкой задачей.

Для решения этой проблемы было решено создать некий единый стандарт для программ, исполняющихся в гетерогенной среде. Это означает, что программа, вообще говоря, должна быть способна исполняться на компьютере, в котором установлены одновременно GPU NVidia и AMD, Toshiba SpursEngine и т.д.

Для разработки открытого стандарта решено было привлечь Khronos Group, которые разрабатывали такие стандарты как OpenGL и OpenML. В разработке и финансировании так же участвовали AMD, IBM, Activision Blizzard, Intel, NVidia и т.д.

Компания NVidia не афишировала свое участие в проекте, и быстрыми темпами наращивала функциональность и производительность CUDA. В то время как несколько ведущих инженеров NVidia участвовали в создании OpenCL. Вероятно, именно участие NVidia в большой мере определило синтаксическую и идеологическую схожесть OpenCL и CUDA, что облегчает для программиста переход от одного языка к другому.

Первая версия стандарта была опубликована в конце 2008 года.

Драйвер для OpenCL был выпущен NVidia и прошел проверку на совместимость со стандартом, но доступен только для ограниченного круга людей - зарегистрированных разработчиков.

Реализация OpenCL для NVidia была достаточно легкой задачей, так как основные идеи сходны. CUDA и OpenCL - некоторое расширение языка С, со сходным синтаксисом, используют одинаковую программную модель в качестве основной - Data Parallel (SIMD). Так же OpenCL поддерживает Task Parallel programming model - модель, когда одновременно могут выполняться различные kernel (work-group содержит один элемент). О схожести двух технологий говорит так же то, что NVidia опубликовала специальный документ, в котором описано как писать для CUDA так, чтобы потом легко перейти на OpenCL.

# // TODO

### Как обстоят дела на настоящий момент

Основной проблемой реализации OpenCL от NVidia является низкая производительность по сравнению с CUDA, но с каждым новым релизом драйверов производительность OpenCL под управлением CUDA все ближе подбирается к производительности CUDA приложений. По заявлениям разработчиков такой же путь проделала и производительность самих CUDA приложений – от сравнительно невысокой на ранний версиях драйверов до впечатляющей в настоящее время.  
А что же делала в этот момент AMD? Ведь именно AMD (как сторонник открытых стандартов – закрытый PhysX vs. открытый Havoc; дорогой Intel Thread Profiler vs. бесплатный AMD CodeAnalyst) делала большие ставки на новую технологию, учитывая что AMD Stream не удавалось хоть сколь-нибудь соревноваться в популярности с NVidia CUDA – виною тому отставание Stream от CUDA в техническом плане.   
Летом 2009 года компания AMD сделала заявление о поддержке и соответствии стандарту OpenCL в новой версии Stream SDK. На деле же оказалось, что поддержка была реализована только для CPU. Да, именно так, это ничему не противоречит – OpenCL стандарт для гетерогенных систем и ничего не мешает Вам запустить kernel на CPU, более того – это очень удобно в случае если в системе нет другого OpenCL устройства. В таком случае программа будет продолжать работать, только медленнее. Или же вы можете задействовать все вычислительные мощности, которые есть в компьютере – как GPU так и CPU, хотя на практике это не имеет особого смысла, так как время исполнения kernel’ов которые исполняются на CPU будет намного больше тех что исполняются на GPU – скорость процессора станет узким местом. Зато для отладки приложений это более чем удобно.  
Поддержка OpenCL для графических адаптеров AMD так же не заставила себя долго ждать – по последним сообщениям компании версия для графических чипов сейчас находится на стадии подтверждения соответствия спецификациям стандарта. После чего она станет доступна всем желающим.  
Так как OpenCL должен работать поверх некоторой специфической для железа оболочки, а значит для того чтобы можно этот стандарт действительно стал единым для различных гетерогенных систем – надо чтобы соответствующие оболочки (драйверы) были выпущены и для IBM Cell и для Intel Larrabie. Пока от этих гигантов IT ничего не слышно, таким образом OpenCL остается еще одним средством разработки для GPU на ряду с CUDA, Stream и DirectX Compute.  
Apple также заявляет о поддержке OpenCL, которая, впрочем, обеспечивается за счет NVidia CUDA.  
Также в настоящее время сторонними разработчиками предлагается:

* [OpenTK](http://www.opentk.com/) — библиотека-обертка над OpenGL, OpenAL и OpenCL для .Net.
* [PyOpenCL](http://mathema.tician.de/software/pyopencl) – обертка над OpenCL для Pyton.
* [Java](http://code.google.com/p/nativelibs4java/wiki/OpenCL) обертка для OpenCL.

### Заключение

Технология OpenCL представляет интерес для различных компания IT сферы – от разработчиков игр до производителей чипов, а это означает что у нее большие шансы стать фактическим стандартом для разработки высокопроизводительных вычислений, отобрав этот титул у главенствующей в этом секторе CUDA.  
  
В будущем я планирую более подробную статью о самом OpenCL, описывающую что из себя представляет эта технология, ее особенности, достоинства и недостатки.  
Спасибо за внимание.

## Программируемая пользователем вентильная матрица (FPGA)

Программируемая пользователем вентильная матрица (ППВМ, англ. Field-Programmable Gate Array, FPGA) - полупроводниковое устройство, которое может быть сконфигурировано производителем или разработчиком после изготовления; отсюда название: «программируемая пользователем». ППВМ программируются путём изменения логики работы принципиальной схемы, например с помощью исходного кода на языке проектирования (типа VHDL), на котором можно описать эту логику работы микросхемы. ППВМ является одной из архитектурных разновидностей программируемых логических интегральных схем (ПЛИС).

ППВМ могут быть модифицированы практически в любой момент в процессе их использования. Они состоят из конфигурируемых логических блоков, подобных переключателям с множеством входов и одним выходом (логические вентили или gates). В цифровых схемах такие переключатели реализуют базовые двоичные операции AND, NAND, OR, NOR и XOR. В большинстве современных микропроцессоров функции логических блоков фиксированы и не могут модифицироваться. Принципиальное отличие ППВМ состоит в том, что и функции блоков, и конфигурация соединений между ними могут меняться с помощью специальных сигналов, посылаемых схеме. В некоторых специализированных интегральных схемах (ASIC) используются логические матрицы, аналогичные ППВМ по структуре, однако они конфигурируются один раз в процессе производства, в то время как ППВМ могут постоянно перепрограммироваться и менять топологию соединений в процессе использования. Однако такая гибкость требует существенного увеличения количества транзисторов микросхемы.

## GPGPU

GPGPU (также GPGP, GP²U, англ. General-purpose computing for graphics processing units, неспециализированные вычисления на графических процессорах) -техника использования графического процессора видеокарты, который обычно имеет дело с вычислениями только для компьютерной графики, чтобы выполнять расчёты в приложениях для общих вычислений, которые обычно проводит центральный процессор. Это стало возможным благодаря добавлению программируемых шейдерных блоков и более высокой арифметической точности растровых конвейеров, что позволяет разработчикам ПО использовать потоковые процессоры для не-графических данных.

## Медианная фильтрация

Все линейные алгоритмы фильтрации приводят к сглаживанию резких перепадов яркости изображений, прошедших обработку. Этот недостаток, особенно существенный, если потребителем информации является человек, принципиально не может быть исключен в рамках линейной обработки. Дело в том, что линейные процедуры являются оптимальными при [гауссовском распределении](http://sernam.ru/book_tp.php?id=25) сигналов, помех и наблюдаемых данных. Реальные изображения, строго говоря, не подчиняются данному распределению [вероятностей](http://edu.sernam.ru/book_kiber1.php?id=227). Причем, одна из основных причин этого состоит в наличии у изображений разнообразных границ, перепадов яркости, переходов от одной текстуры к другой и т. п. Поддаваясь локальному гауссовскому описанию в пределах ограниченных участков, многие реальные изображения в этой связи плохо представляются как глобально гауссовские объекты. Именно это и служит причиной плохой передачи границ при линейной фильтрации.

Вторая особенность линейной фильтрации - ее оптимальность, как только что упоминалось, при гауссовском характере помех. Обычно этому условию отвечают шумовые помехи на изображениях, поэтому при их подавлении линейные алгоритмы имеют высокие показатели. Однако, часто приходится иметь дело с изображениями, искаженными помехами других типов. Одной из них является импульсная помеха. При ее воздействии на изображении наблюдаются белые или (и) черные точки, хаотически разбросанные по кадру. Применение линейной фильтрации в этом случае неэффективно - каждый из входных импульсов (по сути - [дельта-функция](http://edu.sernam.ru/book_p_math2.php?id=139)) дает отклик в виде [импульсной характеристики](http://scask.ru/book_brts.php?id=37) фильтра, а их совокупность способствует распространению помехи на всю площадь кадра.

Удачным решением перечисленных проблем является применение медианной фильтрации, предложенной Дж. Тьюки в 1971 г. для анализа экономических процессов. Наиболее полное исследование медианной фильтрации применительно к обработке изображений представлено в сборнике [3.9]. Отметим, что медианная фильтрация представляет собой эвристический метод обработки, ее алгоритм не является математическим решением строго сформулированной задачи. Поэтому исследователями уделяется большое внимание анализу эффективности обработки изображений на ее основе и сопоставлению с другими методами.

При применении [медианного фильтра](http://sernam.ru/book_prett2.php?id=8) (МФ) происходит последовательная обработка каждой точки кадра, в результате чего образуется последовательность оценок. В идейном отношении обработка в различных точках независима (этим МФ похож на масочный фильтр), но в целях ее [ускорения](http://sernam.ru/book_phis_t1.php?id=19) целесообразно алгоритмически на каждом шаге использовать ранее выполненные вычисления.

При медианной фильтрации используется двумерное окно (апертура фильтра), обычно имеющее центральную симметрию, при этом его центр располагается в текущей точке фильтрации. На рис. 3.10 показаны два примера наиболее часто применяемых вариантов окон в виде креста и в виде квадрата. Размеры апертуры принадлежат к числу параметров, оптимизируемых в процессе анализа эффективности алгоритма. Отсчеты изображения, оказавшиеся в пределах окна, образуют рабочую выборку текущего шага.

|  |  |
| --- | --- |
| http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image001.gif | http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image002.gif |
| а) | б) |
| Рис. 3.10. Примеры окон при медианной фильтрации | |

Двумерный характер окна позволяет выполнять, по существу, двумерную фильтрацию, поскольку для образования оценки привлекаются данные как из текущих строки и столбца, так и из соседних. Обозначим рабочую выборку в виде одномерного массива http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image003.gif; число его элементов равняется размеру окна, а их расположение произвольно. Обычно применяют окна с нечетным числом точек http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image004.gif (это автоматически обеспечивается при центральной симметрии апертуры и при вхождении самой центральной точки в ее состав). Если упорядочить последовательность http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image005.gif по возрастанию, то ее медианой будет тот элемент выборки, который занимает центральное положение в этой упорядоченной последовательности. Полученное таким образом число и является продуктом фильтрации для текущей точки кадра. Понятно, что результат такой обработки в самом деле не зависит от того, в какой последовательности представлены элементы изображения в рабочей выборке http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image006.gif. Введем формальное обозначение описанной процедуры в виде:

http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image007.gif.                      (3.48)

Рассмотрим пример. Предположим, что выборка имеет вид: http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image008.gif, а элемент 250, расположенный в ее центре, соответствует текущей точке фильтрации http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image009.gif (рис. 3.10). Большое значение яркости в этой точке кадра может быть результатом воздействия импульсной (точечной) помехи. Упорядоченная по возрастанию выборка имеет при этом вид http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image010.gif{45,55,75,99,104,110,136,158,250}, следовательно, в соответствии с процедурой (3.48), получаем http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image011.gif. Видим, что влияние “соседей” на результат фильтрации в текущей точке привело к “игнорированию” импульсного выброса яркости, что следует рассматривать как эффект фильтрации. Если импульсная помеха не является точечной, а покрывает некоторую локальную область, то она также может быть подавлена. Это произойдет, если размер этой локальной области будет меньше, чем половина размера апертуры МФ. Поэтому для подавления импульсных помех, поражающих локальные участки изображения, следует увеличивать размеры апертуры МФ.

Из (3.48) следует, что действие МФ состоит в “игнорировании” экстремальных значений входной выборки - как положительных, так и отрицательных выбросов. Такой принцип подавления помехи может быть применен и для ослабления шума на изображении. Однако исследование подавления шума при помощи медианной фильтрации показывает, что ее эффективность при решении этой задачи ниже, чем у линейной фильтрации [3.9].

Результаты экспериментов, иллюстрирующие работу МФ, приведены на рис. 3.11. В экспериментах применялся МФ, имеющий квадратную апертуру со

стороной равной 3. В левом ряду представлены изображения, искаженные помехой, в правом - результаты их медианной фильтрации. На рис. 3.11.а и рис. 3.11.в показано исходное изображение, искаженное импульсной помехой. При ее наложении использовался [датчик случайных чисел](http://edu.sernam.ru/book_kiber1.php?id=351) с равномерным на интервале [0, 1] законом распределения, вырабатывающий во всех точках кадра независимые [случайные числа](http://edu.sernam.ru/book_kiber2.php?id=537). Интенсивность помехи задавалась вероятностью http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image012.gif ее возникновения в каждой точке. Если для случайного числа http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image013.gif, сформированного в точке http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image009.gif, выполнялось условие http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image014.gif, то яркость изображения http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image010.gifhttp://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image015.gif в этой точке  замещалась числом 255, соответствующим максимальной яркости (уровню белого). На рис. 3.11.а действием импульсной помехи искажено 5 % (http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image012.gif=0.05), а на рис. 3.11.в - 10 % элементов изображения. Результаты обработки говорят о практически полном подавлении помехи в первом случае и о ее значительном ослаблении во втором.

// TODO добавить результаты фильтрации

Рис. 3.11.д показывает изображение, искаженное независимым гауссовским шумом при отношении сигнал/шум http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image022.gifдБ, а рис. 3.11.е - результат его фильтрации [медианным фильтром](http://sernam.ru/book_prett2.php?id=8). Условия данного эксперимента позволяют сравнивать его результаты с результатами рассмотренной выше линейной фильтрации. В таблице 3.1 приведены данные, дающие возможность такого сравнения. Для различных методов фильтрации в этой таблице приводятся значения относительного среднего квадрата ошибок http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image023.gif и коэффициента ослабления шума http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image024.gif для случая, когда отношение сигнал/шум на входе фильтра составляет  -5 дБ.

Табл.3.1. Сравнение эффективности подавления шума при [фильтрации изображений](http://sernam.ru/book_ot.php?id=31), http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image022.gif дБ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | масочный фильтр с оптимальн. КИХ | масочный фильтр с равномерн. КИХ | двумерный рекуррентн. фильтр | двумерный фильтр Винера | [медианный фильтр](http://sernam.ru/book_prett2.php?id=8) |
| http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image023.gif | 0.309 | 0.395 | 0.29 | 0.186 | 0.539 |
| http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image024.gif | 10.2 | 8.0 | 10.9 | 17.0 | 5.86 |

Наибольшей эффективностью обладает двумерный [фильтр Винера](http://sernam.ru/tau_31.php), уменьшающий средний квадрат ошибок в 17 раз. [Медианный фильтр](http://sernam.ru/book_prett2.php?id=8) имеет наименьшую из всех рассмотренных фильтров эффективность, ему соответствует http://www.sernam.ru/archive/arch.php?path=../htm/book_kir/files.book&file=kir_25.files/image024.gif=5.86. Тем не менее, это число свидетельствует о том, что и при его помощи удается значительно снизить уровень шума на изображении.

Вместе с тем, как говорилось выше, и что демонстрирует рис. 3.11.е, медианная фильтрация в меньшей степени сглаживает границы изображения, чем любая линейная фильтрация. Механизм этого явления очень прост и заключается в следующем. Предположим, что апертура фильтра находится вблизи границы, разделяющей светлый и темный участки изображения, при этом ее центр располагается в области темного участка. Тогда, вероятнее всего, рабочая выборка будет содержать большее количество элементов с малыми значениями яркости, и, следовательно, медиана будет находиться среди тех элементов рабочей выборки, которые соответствуют этой области изображения. Ситуация меняется на противоположную, если центр апертуры смещен в область более высокой яркости. Но это и означает наличие чувствительности у МФ к перепадам яркости.

# Практическая часть.

Была написана программа фильтрации изображения медианным алгоритмом. Алгоритм был реализован с использованием технологии OpenCL, а так же с использованием стандартных средств C++.

### Описание алгоритма фильтрации изображения

Для каждого пикселя берем окно размером “Глубина фильтрации”\*“Глубина фильтрации”. В стандартном изображении каждый пиксель кодируется тремя цветами RGB. Для качественной фильтрации шумов необходимо фильтровать пиксели по всем трем цветам. Т.е. в действительности нам необходимо провести медианную фильтрацию 3 раза для каждого пикселя.

1. Сначала берем окно для красной компоненты.
2. Расставляем все интенсивности по возрастанию.
3. Величина интенсивности красной компоненты обрабатываемого пикселя будет равна центральному элементу отсортированного окна.
4. Повторяем эту процедуру для зеленого и синего цвета

После этого мы получаем готовый пиксель. Для сортировки был использован алгоритм …, который достаточно быстрый и простой в реализации, что не маловажно, т.к. графические устройства не поддерживают рекурсию и динамическое выделение памяти.

Листинг написанной программы, которая выполняется ядром устройства OpenCL:

\_\_kernel void Filter(

\_\_global unsigned int \*in,

\_\_global unsigned int \*out,

int edge)

{

const int x = get\_global\_id(0); // Получаем индекс в 0 измерение

const int y = get\_global\_id(1); // Получаем индекс в 1 измерение

const int width = get\_global\_size(0); // Получаем размерность 0 измерения

const int height = get\_global\_size(1); // Получаем размерность 1 измерения

if ((x >= width) || (y >= height)) return; // Проверяем что индексы не вышли за диаппазон

unsigned int tmp[1000]; // Создадим массив для фильтрующего окна

unsigned char colorTmp[1000]; // Массив для цветов

int tmpSize = edge \* edge;

unsigned int pixel = 0x000000;

// Берем окно размером edge x edge

for(int l = -edge/2; l < edge/2; l++)

{

int line = l;

if(l + y >= height)

{

line = height - (l + y);

}

else if(y + l < 0)

{

line = -(y + l);

}

for(int r = -edge/2; r < edge/2; r++)

{

int raw = r;

if(r + x >= width)

{

raw = width - (r + x);

}

else if(r + x < 0)

{

raw = -(r + x);

}

tmp[(l + edge/2) \* edge + (r + edge/2)] = in[(width \* (y + line)) + (x + raw)];

}

}

// Красный

for(int i = 0; i < tmpSize; i++)

{

colorTmp[i] = RED(tmp[i]);

}

sort(&colorTmp, tmpSize);

pixel = pixel + OUTRED(colorTmp[(edge \* edge - 1) / 2]);

// Зеленый

for(int i = 0; i < tmpSize; i++)

{

colorTmp[i] = GREEN(tmp[i]);

}

sort(&colorTmp, tmpSize);

pixel = pixel + OUTGREEN(colorTmp[(edge \* edge - 1) / 2]);

// Синий

for(int i = 0; i < tmpSize; i++)

{

colorTmp[i] = BLUE(tmp[i]);

}

sort(&colorTmp, tmpSize);

pixel = pixel + OUTBLUE(colorTmp[(edge \* edge - 1) / 2]);

// Записываем в пиксель медиану (центральный пиксель)

out[width \* y + x] = pixel;

}

Листинг алгоритма с использованием стандартных средств C++:

void COpenCLImageFilterDlg::LAFilter(unsigned int\* in, unsigned int\* out, int width, int height, int edge)

{

for(int y = 0; y < height; y++)

{

for(int x = 0; x < width; x++)

{

int tmpSize = edge \* edge;

unsigned char \*colorTmp = new unsigned char[tmpSize]; // Массив для цветов

unsigned int \*tmp = new unsigned int[tmpSize]; // Создадим массив для фильтрующего окна

unsigned int pixel = 0x000000;

// Берем окно размером edge x edge

for(int l = -edge/2; l < edge/2; l++)

{

int line = l;

if(l + y >= height)

{

line = height - (l + y);

}

else if(y + l < 0)

{

line = -(y + l);

}

for(int r = -edge/2; r < edge/2; r++)

{

int raw = r;

if(r + x >= width)

{

raw = width - (r + x);

}

else if(r + x < 0)

{

raw = -(r + x);

}

tmp[(l + edge/2) \* edge + (r + edge/2)] = in[(width \* (y + line)) + (x + raw)];

}

}

// Красный

for(int i = 0; i < tmpSize; i++)

{

colorTmp[i] = RED(tmp[i]);

}

sort(colorTmp, tmpSize);

pixel = pixel + OUTRED(colorTmp[(edge \* edge - 1) / 2]);

// Зеленый

for(int i = 0; i < tmpSize; i++)

{

colorTmp[i] = GREEN(tmp[i]);

}

sort(colorTmp, tmpSize);

pixel = pixel + OUTGREEN(colorTmp[(edge \* edge - 1) / 2]);

// Синий

for(int i = 0; i < tmpSize; i++)

{

colorTmp[i] = BLUE(tmp[i]);

}

sort(colorTmp, tmpSize);

pixel = pixel + OUTBLUE(colorTmp[(edge \* edge - 1) / 2]);

// Записываем в пиксель медиану (центральный пиксель)

out[width \* y + x] = pixel;

delete [] tmp;

delete [] colorTmp;

}

}

}

### Приведение результатов работы программы

Для демонстрации целесообразности использования данной технологии сравним скорость выполнения одного и того же алгоритма с использованием OpenCL на разных платформах/устройствах и с использованием стандартных средств C++.

Конфигурация компьютера, на котором выполнялось сравнение:

CPU Intel Core i5-2450M

GPU AMD Radeon HD 7400M

ОЗУ 6Гб

Устройства поддерживающие OpenCL: CPU, GPU

Результаты фильтрации изображения размером 584x329 px с разным уровнем зашумленности представлены в таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Уровень шума, % | Глубина фильтрации, px | Устройство | Время, мс |
| 15% | 4 | CPU | 146.899 |
| 15% | 4 | GPU | 273.904 |
| 15% | 4 | CPU & GPU | 162.921 |
| 15% | 4 | Линейный алгоритм | 766.343 |
| 50% | 8 | CPU | 1186.602 |
| 50% | 8 | GPU | 3544.097 |
| 50% | 8 | CPU & GPU | 1773.141 |
| 50% | 8 | Линейный алгоритм | 5128.284 |

Таблица 1. Результаты фильтрации изображения размером 584x329 px

Результаты фильтрации изображения размером 128x697 px с разным уровнем зашумленности представлены в таблице 2.

|  |  |  |  |
| --- | --- | --- | --- |
| Уровень шума, % | Глубина фильтрации, px | Устройство | Время, мс |
| 15% | 4 | CPU | 639.554 |
| 15% | 4 | GPU | 932.504 |
| 15% | 4 | CPU & GPU | 492.247 |
| 15% | 4 | Линейный алгоритм | 3052.037 |
| 50% | 8 | CPU | 8412.208 |
| 50% | 8 | GPU | 14822.143 |
| 50% | 8 | CPU & GPU | 7448.263 |
| 50% | 8 | Линейный алгоритм | 30735.625 |

Таблица 2. Результаты фильтрации изображения размером 128x697 px

Как видно из таблиц, фильтрация изображения с использованием технологии OpenCL проходит до 5-6 раз быстрее, чем с использованием линейных алгоритмов С++.

# Выводы.

# Список литературы.

1. OpenCL: <https://ru.wikipedia.org/wiki/OpenCL> (Дата обращения: 07.06.2016 г.).

2. Программируемая пользователем вентильная матрица: <https://ru.wikipedia.org/w/index.php?title=FPGA&redirect=no> (Дата обращения: 07.06.2016 г.).

3. GPGPU: <https://ru.wikipedia.org/wiki/GPGPU> (Дата обращения: 07.06.2016 г.).

4.