# ДЕТЕКТИРОВАНИЕ ЛИЦ ЛЮДЕЙ НА ИЗОБРАЖЕНИИ С ИСПОЛЬЗОВАНИЕМ ГРАФИЧЕСКИХ УСКОРИТЕЛЕЙ

***А.С. Голубев, И.И. Зиновьев***

*Владимирский госуниверситет им. А.Г. и Н.Г. Столетовых*

Предложен вариант алгоритма для детектирования лиц людей на изображении с использованием графических ускорителей. Данный алгоритм был проверен на практике в реальных условиях и подтвердил свою работоспособность при анализе видеопотока высокого разрешения в реальном времени.

## Введение

С развитием современной техники видеонаблюдения повышаются требования к быстродействию алгоритмов детектирования лиц людей на изображении в связи с увеличением разрешения анализируемых кадров. Современные центральные процессоры (CPU) уже не справляются с этой задачей в реальном времени. Одним из возможных решений этой проблемы является использование графических ускорителей (GPU). В работе предлагается вариант реализации алгоритма детектирования лиц людей на изображении, представленного в 2001 году Полом Виолой (Paul Viola) и Майклом Джонсом (Michael Jones) [5], для архитектуры NVIDIA CUDA.

## 1. Особенности реализации алгоритма

Подходы к распараллеливанию алгоритма Виола-Джонса подробно рассмотрены в работе [1]. Мы выбрали вариант, в котором каждый поток анализирует свой участок изображения. В качестве основы взята реализация алгоритма Виола-Джонса для CPU из открытой библиотеки OpenCV [6].

Рассмотрим особенности предлагаемой нами реализации алгоритма детектирования.

### 1.1. Размещение исходных данных в памяти GPU

Исходное изображение загружается в текстурную память графического ускорителя, что позволяет реализовать эффективный алгоритм его масштабирования путем выборки из текстуры с аппаратной билинейной фильтрацией.

Т.к. каскад классификаторов является неизменным и необходимым всем потокам одновременно, то наиболее эффективно хранить его в константной памяти. Размещение классификатора в памяти графического ускорителя осложняется сложной иерархией его структур в библиотеке OpenCV и ограниченным размером константной памяти. Поэтому формат хранения каскада был преобразован с использованием одномерных массивов и минимально возможных типов данных. Сам процесс приведения каскада к новому формату не вносит значительных временных затрат, т.к. производится один раз при загрузке каскада. Такая реорганизация каскада позволила хранить в константной памяти графического ускорителя до 2150 слабых классификаторов.

Следует заметить, что все буферы в памяти GPU выделяются один раз на этапе инициализации детектора, что немаловажно при обработке видеопотока.

### 1.2. Использование вычислительных ресурсов GPU

Особенностью нашей реализации по сравнению с другими [1, 2, 4] является использование цельного ядра детектирования (рис. 1). Это позволяет значительно снизить число обращений в медленную глобальную память за счет размещения данных в разделяемой памяти.

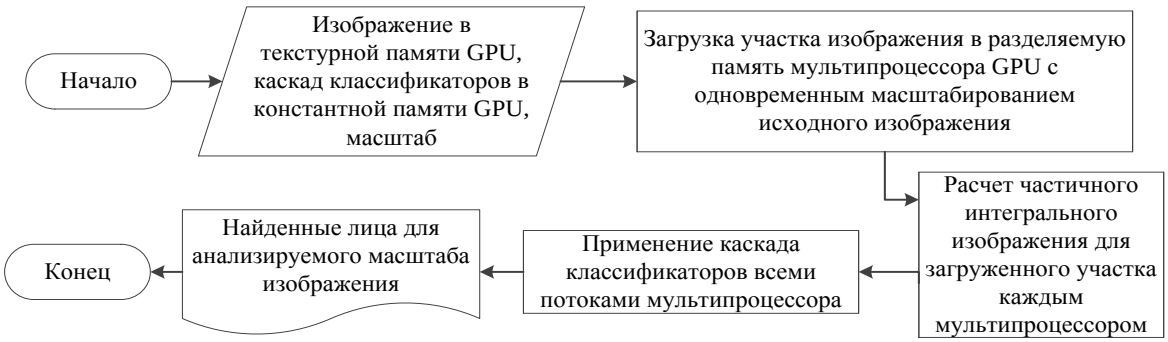


Рис. 1. Алгоритм работы ядра детектирования

Для достижения поставленной цели в разделяемую память мультипроцессора из текстурной памяти считывается участок изображения, достаточный для обеспечения исходными данными всех его потоков. Затем производится вычисление независимых интегральных изображений для частей изображения, уже размещенных в разделяемой памяти мультипроцессоров. Каждый мультипроцессор сначала выполняет суммирование своего участка изображения по строкам, затем – по столбцам. Затем каждый поток независимо от других анализирует свой участок изображения на наличие на нем лица.

Главным недостатком такого подхода является большой размер полученного ядра детектирования и, следовательно, большое число регистров, требуемое ему для работы. Но правильный выбор параметров для выполнения ядра позволяет снизить влияние этого фактора (рис. 2).

### 1.3. Получение результатов из памяти GPU

В процессе детектирования каждый поток сохраняет результат своей работы в выделенную на этапе инициализации область глобальной памяти. Для ускорения работы реализована их асинхронная загрузка в оперативную память компьютера (технология «zero copy»). После завершения анализа очередного кадра в оперативную память выгружаются все оставшиеся к этому моменту промежуточные результаты. Дальнейшая обработка полученных результатов, связанная с группировкой лиц, найденных в разных масштабах, производится на центральном процессоре.

## 2. Апробация предложенной реализации алгоритма детектирования

Для наглядной оценки скорости работы полученной реализации алгоритма (обозначим как gpu) проведем сравнение с многопоточными вариантами данного алгоритма, представленными в библиотеке OpenCV, как для центрального процессора (OCV-cpu), так и для графического ускорителя (OCV-gpu). Эксперимент состоит в следующем: один и тот же видеофайл продолжительностью 6 минут, полученный с камер видеонаблюдения, установленных на турникетах метрополитена, анализировался всеми вариантами алгоритма. Эксперимент проводился на компьютере, в состав которого входят 2 процессора Intel Xeon E5520 2.27 Ghz (в сумме 8 физических или 16 логических ядер) и видеокарта NVIDIA GTX 470 (448 CUDA-ядер). При этом фиксировались следующие показатели: минимальное ($T_{min}$), максимальное ($T_{max}$) и среднее ($T_{avg}$) время,

затраченное на анализ одного кадра, процент загрузки центрального процессора ($P_{cpu}$) и процессора графического ускорителя ($P_{gpu}$), объем используемой оперативной памяти ($M_{cpu}$) и памяти графического ускорителя ($M_{gpu}$). В ходе эксперимента для всех вариантов алгоритма были установлены следующие параметры: минимальный размер лица 20x20 пикселей, шаг масштабирования 1.2, коэффициент группировки 3, каскад классификаторов haarcascade_frontalface_alt2.xml из состава библиотеки OpenCV. Результаты эксперимента приведены в табл. 1.

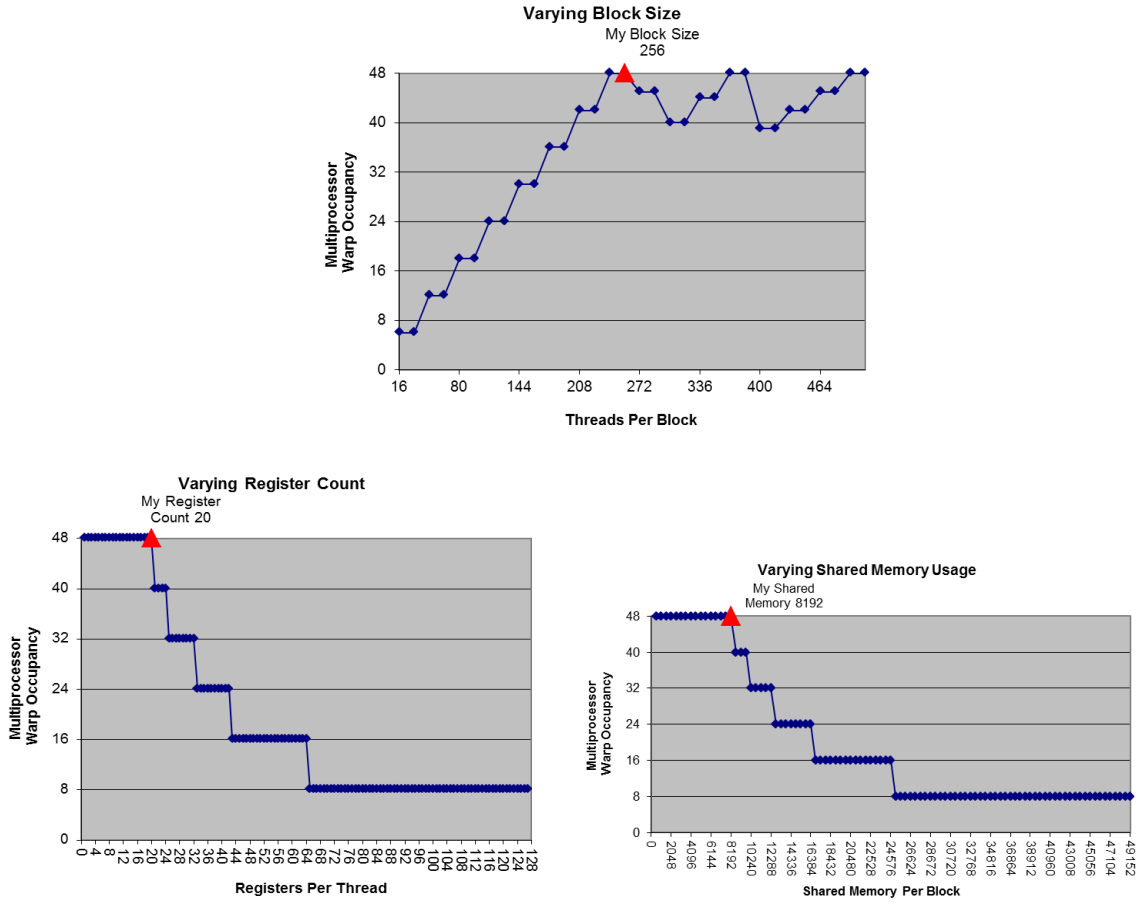


Рис. 2. Выбранные параметры выполнения ядра детектирования (с использованием NVIDIA Occupancy Calculator)

Таблица 1. Сравнительные характеристики разных реализаций алгоритма Виола-Джонса

| Реализация | $T_{min}$, мс | $T_{max}$, мс | $T_{avg}$, мс | $P_{cpu}$, % | $P_{gpu}$, % | $M_{cpu}$, МБ | $M_{gpu}$, МБ |
|---|---|---|---|---|---|---|---|
| OCV-cpu | 410,23 | 585,03 | 438,49 | 60 | - | 79,348 | - |
| OCV-gpu | 171,29 | 213,91 | 195,78 | 7 | 76 | 184,908 | 201 |
| gpu | 46,73 | 82,46 | 69,91 | 7 | 59 | 89,344 | 63 |

Следует отметить, что предложенная нами реализация алгоритма детектирования оказалась как быстрее, так и эффективнее в использовании ресурсов графического ускорителя.

Сравнение эффективности работы рассмотренных выше программных реализаций алгоритма Виола-Джонса проводилось по относительному показателю положительных (True positive rate) и ложных (False positives) обнаружений при анализе изображений из

базы Face Detection Data Set and Benchmark (FDDB) [3]. В ходе эксперимента для алгоритма Виола-Джонса были установлены параметры, аналогичные описанным выше, изменялся только коэффициент группировки в пределах от 1 до 7. Полученные результаты приведены на рис. 3.

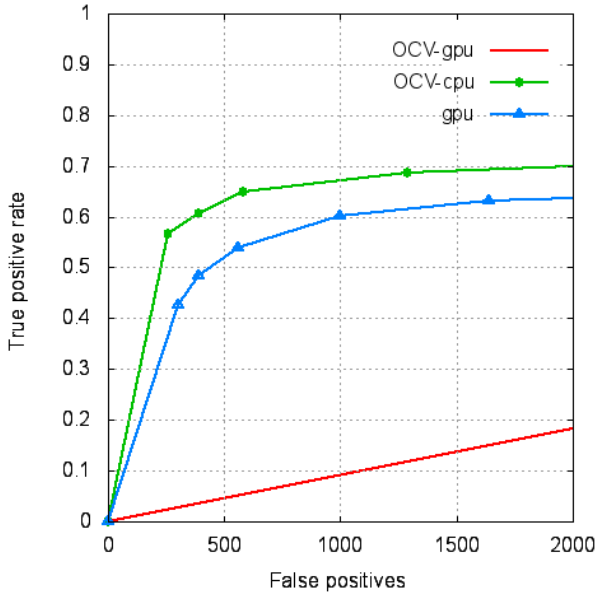


Рис. 3. Сравнение программных реализаций по базе FDDB (Discrete Score)

Предложенная в работе реализация алгоритма продемонстрировала сравнимые результаты с вариантом для центрального процессора.

**Заключение**

Полученные результаты демонстрируют, что предложенный нами вариант адаптации алгоритма Виола-Джонса для графического ускорителя применим на практике и успешно конкурирует с другими подходами.

Данная работа выполнена в рамках НИР по госзаданию «Наука» (Регистрационный номер: 8.3303.2011 от 23.11.2011).

**Литература**

1. Harvey J.P. GPU acceleration of object classification algorithms using NVIDIA CUDA. Master's thesis, Rochester Institute of Technology. 2009.
2. Hefenbrock D., Oberg J., Thanh N.T.N., Kastner R., Baden S.B. Accelerating viola-jones face detection to FPGA-level using GPUs // In Proceedings of the 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM '10, Washington, DC, USA. IEEE Computer Society. 2010. P. 11–18.
3. Jain V., Learned-Miller E. FDDB: A Benchmark for Face Detection in Unconstrained Settings. Technical Report UM-CS-2010-009, Dept. of Computer Science, University of Massachusetts, Amherst. 2010.
4. Obukhov A. Haar classifiers for object detection with CUDA. GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics. Addison Wesley, 2004. P. 517–544.
5. Viola P., Jones M. Robust real-time face detection // International Journal of Computer Vision. 2004. 57(2). P. 137–154.
6. Документация открытой библиотеки OpenCV – [http://opencv.willowgarage.com/wiki/].