2. С.В. Цаплин и др. отчёт НИОКР «Проведение сравнительных испытаний масел закалочных производства ООО «НЗМП» с использованием информационно-измерительной системы определения охлаждающей способности закалочных сред (разработка СамГУ) и прибора «IVF SmartQuench»», (Швеция), применяемого в ОАО «АВТОВАЗ», СамГУ, Самара 2012г., 60с.

3. Цаплин С.В. Информационно-измерительная система исследования охлаждающей способности закалочных сред [Текст] / С.В. Цаплин, С.А. Болычев, Б.С.Мишагин, Д.В.Шеманаев // Металловедение и термическая обработка металлов. – 2014. – №5.

А.А. Цыганов

GPGPU ОПТИМИЗАЦИЯ ВЫЧИСЛЕНИЙ МЕТРИКИ СХОЖЕСТИ КАДРОВ ТРЕХМЕРНОГО ВЕКТОРНОГО ВИДЕО

(Самарский государственный технический университет)

Использование потоковых процессоров графических ускорителей и платформы *CUDA* позволяет добиться значительного прироста производительности по сравнению с расчетами на процессорах общего назначения при решении задач в области компьютерного зрения, в частности для определения схожести изображений. В статье рассматриваются методы, применяемые для решения задачи вычисления метрики схожести кадров трехмерного векторного видео. Приведены результаты исследования производительности *GPGPU* реализации расчетов значений метрики.

Метрика схожести изображений

Для воспроизведения видео в трехмерном векторном формате важной задачей является определение типов параметров шейдерных программ, содержащихся в видео потоке. Это может быть осуществлено на основе сравнения растровых представлений исходного кадра видео и кадра, модифицированного с использованием предположения о типе параметров. Исходный кадр и модифицированный кадр проходят процесс растеризации, результатом которого являются два изображения I_o и I_m соответственно. Они сравниваются с помощью метрики схожести изображений.

Алгоритм вычисления метрики осуществляет обработку изображений в несколько шагов. Из исходных изображений методом рассеивания рассчитываются цветовые гистограммы $H(I_o)$ и $H(I_m)$. Первичная оценка расстояния между изображениями выполняется с помощью расстояния Бхаттачарья $D_B(H_o, H_m)$. Метрика уточняется с помощью сравнения множеств контрольных точек на исходных изображениях [1]. Множества контрольных точек P_o и P_m, получаемые из изображений I_o и I_m соответственно, используются для вычисления расстояния $D_S(P_o, P_m)$. Для обнаружения точек используется метод *SURF* [2].

Вычисление гистограмм

Вычисление компоненты метрики D_B выполняется с помощью гистограмм $H(I_O)$ и $H(I_S)$ соответствующих изображений. Расчет гистограмм на *GPU*

может быть выполнен как с использованием классических шейдерных программ, так и с использованием технологии *CUDA* для вычислений общего назначения на графическом процессоре. Использование технологии *CUDA* описано в работах Подложнюка [3] и Шамса [4]. Эти алгоритмы обеспечивают более высокую производительность, чем те, что основаны на использовании обычных средств графических программных интерфейсов, как это показано в работе Нугтерена и соавторов.

Так как основной задачей метода является ускорение расчета метрики, то наиболее подходящими являются методы на основе *CUDA*. В частности, метод Подложнюка реализован в *CUDA SDK*. Метод кэш-эффективен и не содержит этапов выгрузки данных в общую память, что позволяет его интегрировать в процесс вычисления компонент метрики.

В этом методе исходные данные разделяются на блоки между исполняемыми на GPU потоками. Результат обработки данных каждым потоком сохраняется в индивидуальной гистограмме. В финальном проходе все гистограммы, созданные разными потоками, объединяются в одну. Для эффективного использования общей памяти потоков каждая индивидуальная гистограмма создается для группы потоков, называемой тросом. Это позволяет хранить в памяти гистограммы большего объема, вплоть до 6 килобайт на аппаратной архитектуре *G80.*

На основе полученных гистограмм вычисляется расстояние Бхаттачарья для двух статистических множеств. Вычисление суммы произведений элементов гистограмм реализуется с помощью свертки массивов исходных данных на *GPU* при использовании оптимизированного метода параллельной свертки на для *CUDA.*

Поиск ключевых точек

Вторая компонента метрики D_S рассчитывается с использованием алгоритма *SURF*. С его помощью осуществляется поиск двух множеств ключевых точек *P* и *P'*, имеющихся на оригинальном и модифицированном кадрах соответственно (рисунок 1).

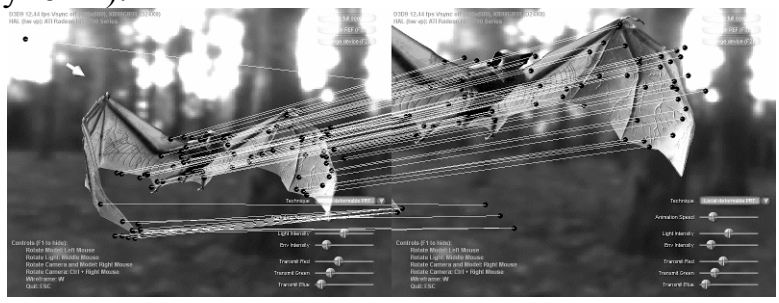

Рис. 1. Сопоставление ключевых точек

Обнаружение ключевых точек в *SURF* осуществляется с помощью аппроксимации определителя матрицы Гессе. Аппроксимация выполняется наложением блочных фильтров на изображение. Это позволяет эффективно использовать интегральное представление изображения.

Вычисление интегрального представления на *GPU* является самым длительным этапом работы алгоритма *SURF* и может быть осуществлено с помо-

щью алгоритма пирамиды моментов, как это описано в работе Террибери и соавторов [5].

Само построение интегрального изображения является задачей префиксной суммы. Алгоритм пирамиды моментов предлагает решение этой задачи на *GPU* в два этапа. На первом этапе осуществляется проход снизу вверх, в ходе которого строится пирамида изображений, каждое из которых разбивается на четыре вдвое меньших по ширине и высоте, чем на предыдущем уровне.

Используя интегральное изображение, ключевые точки определяются путем поиска экстремумов определителя матриц Гессе. Для этого применяются блочные фильтры, описанные в работе Бэя и соавторов [1]. Для их вычисления на *GPU* требуется всего 17 текстурных выборок на пиксель. Нахождения локального максимума Гессианы производится методом соседних точек 3x3x3.

Исследование производительности

Для определения выигрыша производительности *GPGPU* реализации по сравнению с реализацией для процессоров общего назначения, было проведено экспериментальное исследование с различными источниками графической информации. Источниками выступали приложения, выбранные по статистике сервисов потокового видео вещания.

Первая серия опытов нацелена на оценку зависимости времени составления профиля от длительности записи. Результаты представлены на рисунке 2. Как видно, время работы системы возрастает незначительно, так как в более длительных записях почти не появляется новых шейдерных программ. Однако, происходит значительное сокращение времени выполнения при использовании *GPU* реализации в 8-12 раз.

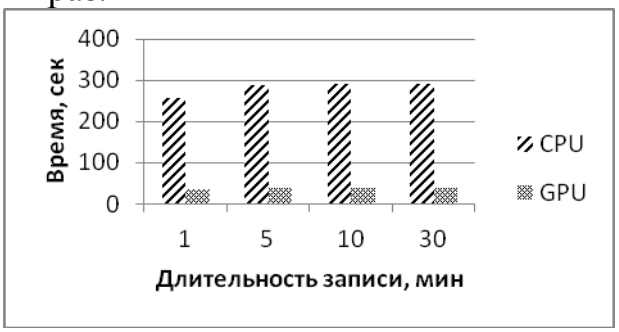

Рис. 2. Диаграмма зависимости времени от длительности записи

Состав шейдерных программ в каждом приложении очень разнороден. Основным свойством, влияющим на сложность анализа конкретной шейдерной программы, является количество ее параметров, представляющих интерес для алгоритма. Для оценки влияния этого количества на время обработки каждой шейдерной программы проведена серия экспериментов над теми же источниками графической информации, что и в предыдущем случае.

Значения являются усредненными по всем шейдерным программам с заданным количеством параметров матричного типа в десятиминутной записи. Результаты представлены на рисунке 3.

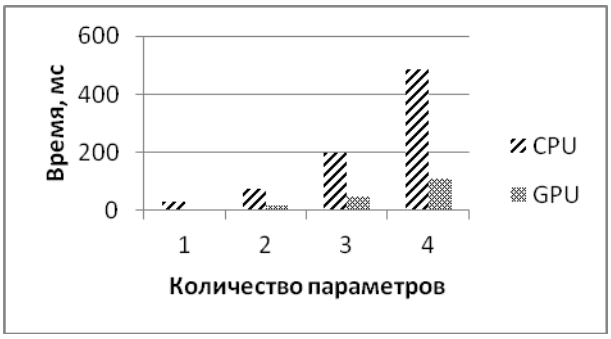

Рис. 3. Диаграмма зависимости времени от количества параметров

Число распознаваемых параметров экспоненциально влияет на длительность их распознавания. Скорость обработки данных методом сильно зависит от сложности системы рендеринга источника видеопотока. Однако вычисления с помощью *GPGPU* способны сократить затраты времени в 8-12 раз. Это позволяет осуществлять сравнение кадров векторного видео и последующее составление профиле за сроки, приемлемые для практического использования описанных методов.

Литература

1. Cornelis N. Fast Scale Invariant Feature Detection and Matching on Programmable Graphics Hardware / N. Cornelis, L. Van Gool // IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008. P. 1 – 8.

2. Herbert Bay. Speeded-Up Robust Features (SURF) / Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool // New York, USA: Computer Vision and Image Understanding, 2008. – Vol. 110. – P. 346 – 359.

3. Podlozhnyuk V. Histogram calculation in CUDA. Technical report, NVIDIA,
2007.URL: http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/histogram64/doc/histogram.pdf

4. Ramtin Shams, R. A. Kennedy. Efficient Histogram Algorithms for NVIDIA CUDA Compatible Devices. – Australia, Gold Coast. – ICSPCS, 2007. – P. 418 – 422.

5. Timothy B. Terriberry. GPU Accelerating Speeded-Up Robust Features / Timothy B. Terriberry, Lindley M. French, John Helmsen // Proceedings of the Fourth International Symposium on 3D Data Processing, Visualization and Transmission. – Georgia Institute of Technology, Atlanta, GA, USA, 2008. – P. 355 – 362.

6. Kari Pulli. Real-time computer vision with OpenCV / Kari Pulli (NVIDIA), Anatoly Baksheev, Kirill Kornyakov, Victor Eruhimov // Communications of the ACM. – ACM: New York, NY, USA, 2012. – P. 61 – 69.