УДК 004.043, 004.032.24, 004.932.4

ИНТЕГРИРОВАННАЯ ПРОГРАММНАЯ БИБЛИОТЕКА ДЛЯ ОБРАБОТКИ МЕДИЦИНСКИХ И ПРОМЫШЛЕННЫХ СНИМКОВ

Степанов Д.Н., Тищенко И.П.

Федеральное государственное бюджетное учреждение науки Институт программных систем им. А.К. Айламазяна Российской академии наук, Исследовательский центр мультипроцессорных систем (152021, Ярославская обл., Переславский р-н, с. Веськово, ул. Петра I, д. 4а), e-mail: mitek1989@mail.ru

Статья посвящена разработке и реализации библиотеки алгоритмов, которая является частью программно-инструментального комплекса высокопроизводительной обработки изображений медицинского и промышленного назначения. Библиотека позволяет работать со снимками различных форматов и является кросс-платформенной. Этот показатель отсутствует у многих существующих систем и комплексов схожего назначения. В библиотеке собраны реализации различных алгоритмов на графических процессорных устройствах (GPU), ориентированные на использование программно-аппаратной архитектуры CUDA. Подробно описаны некоторые функции библиотеки, которая объединяет возможности открытого программного обеспечения, что дает возможность ее практического использования в составе различных прикладных систем. Библиотека является легко расширяемой и содержит в себе функции для решения задач линейной алгебры, а также фильтрации, обработки и анализа изображений.

Ключевые слова: медицинские и промышленные снимки, унифицированная библиотека, GPU, CUDA, OpenCV.

INTEGRATED SOFTWARE LIBRARY FOR MEDICAL AND INDUSTRIAL IMAGES PROCESSING

Stepanov D.N., Tishchenko I.P.

Ailamazyan Program Systems Institute of the Russian Academy of Sciences, Multiprocessor System Research Center (152021, Yaroslavl region, Pereslavl area, Peter I st., 4a), e-mail: mitek1989@mail.ru

The article is devoted to development and implementation of software library, which is part of the computer appliance of high-performance image processing for medical and industrial purposes. The library allows to work with the images of different formats and is cross-platform. This attribute is absent in many existing systems and complexes of similar purpose. The library has the implementations of various algorithms on graphics processing units (GPU) were implemented in the software library, algorithms are carried out with the use of software and hardware architecture CUDA. Some of the functions of the library are described in detail, library combines the capabilities of different free software, it enables its practical use in various application systems. The library is easily expandable and contains functions for solving linear algebra problems, as well as filtering, image processing and analysis.

Key words: medical and industrial images, unified library, GPU, CUDA, OpenCV.

Введение

Коллектив исследователей Лаборатории интеллектуального управления Исследовательского центра мультипроцессорных систем ИПС имени А.К. Айламазяна РАН в рамках работ по государственному контракту разрабатывает программно-инструментальный комплекс (ПИК) высокопроизводительной обработки изображений медицинского и промышленного назначения. Актуальность темы научно-исследовательской работы (НИР) определяется стремительным развитием вычислительной техники и программного обеспечения (ПО), созданием на этой основе прикладных медицинских и промышленных систем диагностики и визуализации, а также различных систем поддержки принятия решений.

Авторами выполнен анализ различных программных и аппаратных комплексов и систем, предназначенных для решения задач в области обработки снимков медицинского и промышленного назначения (рентгеновские, ультразвуковые снимки, результаты компьютерной томографии и т.д.) Примерами таких комплексов служат: Magellan [7], TiViPE [9], система МИКРОКОН РАД [1]. Анализ показал, что производители комплексов предлагают конечному пользователю законченные решения, при этом упор делается на разработку аппаратной части, тогда как для обработки и анализа данных используются стандартные алгоритмы. В большинстве случаев предлагаемые решения предназначены для решения определенного узкого круга задач и обеспечивают раздельную работу либо с рентгеновскими, либо с ультразвуковыми снимками. Сравнивая зарубежные и отечественные системы, можно заметить некоторое отставание отечественных решений в области ПО для обработки медицинских снимков. В области неразрушающего контроля, напротив, комплексы российских производителей часто выбираются потребителями вместо зарубежных в силу лучшего соответствия отечественным ГОСТам.

Универсальный (интегрированный) комплекс – более предпочтительный вариант, поскольку возможности комплекса унифицированы для снимков разных типов. Среди этих возможностей: получение изображений из различных источников, их хранение, а также анализ, обработка и визуализация. При этом желательно, чтобы комплекс был кросс-платформенным – работал на компьютерах под управлением различных операционных систем.

Медицина и промышленность диктуют все более жесткие требования к скорости и качеству обработки данных. Использование традиционных процессоров общего назначения (CPU) не всегда позволяет выполнить эти требования, поэтому для решения задач обработки сигналов и изображений все чаще применяются графические процессорные устройства (GPU), обладающие существенно лучшей производительностью при решении задач определенных классов. А для решения наиболее ресурсоемких задач свое применение находят кластерные и гибридные системы, имеющие высокую производительность, относительно невысокую цену и обеспечивающие гибкость при выборе конфигурации.

При разработке ПИК предусматривается возможность расширяемости: список решаемых с его помощью задач не должен быть фиксированным, а набор функций должен быть при необходимости наращиваемым, в том числе путем добавления новых алгоритмов, использующих в своей работе уже имеющиеся базовые функции.

Все вышесказанное позволяет сформулировать задачу, которую необходимо решить в процессе разработки комплекса: одной из главных составных частей ПИК должна являться унифицированная программная библиотека типовых и базовых алгоритмов обработки

изображений и других данных. Причем основная часть алгоритмов должна быть реализована на GPU.

Архитектура CUDA

Существует несколько программно-аппаратных платформ, используемых при разработке прикладных систем и библиотек GPU-алгоритмов. Для создания унифицированной библиотеки было решено выбрать архитектуру CUDA, так как она является наиболее распространенной, стабильной, и для нее существует большое количество инструментальных средств, облегчающих создание GPU-приложений. В силу особенностей архитектуры CUDA наиболее предпочтительным является реализация алгоритмов для решения задач, в которых возможна декомпозиция на множество подзадач, а каждую подзадачу можно решать независимо. Поэтому CUDA широко применяется для решения задач обработки массивов различной размерности (векторы, матрицы, изображения).

Архитектура унифицированной библиотеки

Разрабатываемая библиотека состоит из набора функций языка C++. Необходимо было выбрать способ представления векторов, матриц и медицинских и промышленных изображений без привязки к каким-то конкретным форматам. Было решено использовать возможности библиотеки OpenCV [8], которая предоставляет достаточно универсальные структуры данных для работы с одномерными и двумерными массивами. В OpenCV для хранения матриц и изображений, которые располагаются в оперативной памяти, используется класс Mat; для хранения данных, располагающихся в памяти GPU – класс GpuMat. Предусмотрена поддержка автоматического выделения/освобождения памяти для обоих классов, имеется много встроенных методов для работы с объектами обоих классов, в том числе функции копирования данных между GPU и оперативной памятью, а также между двумя матрицами, располагающимися на GPU. В то же время имеется возможность работы на низком уровне, с использованием указателей на данные, располагающиеся в основной памяти или на GPU. Эта возможность используется при написании собственных CUDA-функций, а также для вызова необходимых функций из других программных библиотек, ориентированных на работу с указателями (например, библиотека MAGMA).

Большинство функций, предназначенных для обработки двумерных массивов на GPU, устроены следующим образом: на вход поступает один или несколько массивов (исходные данные), результат записывается в один или несколько других массивов (впрочем, часть операций могут выполняться в режиме in place, когда результат записывается в исходные данные). И каждый массив может располагаться как в оперативной памяти, так и в памяти GPU. Где именно – это решает прикладной программист, который должен сам решить где, как и в какой последовательности осуществлять обработку данных. И поэтому часть

функций реализована в нескольких вариантах: когда исходный набор данных расположен в оперативной памяти, и когда он располагается в памяти GPU. То же самое относится и к тому набору двумерных массивов, в которые будет записан результат вычислений. Рассмотрим функцию, предназначенную для сглаживания изображений на GPU методом Гаусса, она существует в четырех вариантах:

```
void smoothGauss( const GpuMat& src, GpuMat& dst, Size ksize, double sigma1, double sigma2);
void smoothGauss( const GpuMat& src, Mat& dst, Size ksize, double sigma1, double sigma2);
void smoothGauss( const Mat& src, GpuMat& dst, Size ksize, double sigma1, double sigma2);
void smoothGauss( const Mat& src, Mat& dst, Size ksize, double sigma1, double sigma2);
```

Первый аргумент каждой функции – исходное изображение, оно может храниться как в памяти хоста, так и в памяти устройства. Второй аргумент – сглаженное изображение, память под него будет выделена или на хосте, или на устройстве (или уже была заранее выделена прикладным программистом). Остальные аргументы определяют вид дискретизированной функции Гаусса, которая будет использоваться для сглаживания.

Необходимо учитывать, что если данные располагаются в памяти хоста, то вызов соответствующей функции приведет к созданию временных буферов памяти на GPU и копированию данных в них. По окончании работы функции память будет освобождена. Если все данные уже располагаются на GPU, то лишних операций по выделению/освобождению памяти не будет. Такой вариант может быть полезен, если программисту необходимо выполнить последовательную обработку данных с помощью нескольких GPU-функций.

Состав унифицированной библиотеки

В рамках выполнения НИР были рассмотрены алгоритмы, которые применяются в системах обработки и визуализации медицинских и промышленных изображений. Основные задачи, решаемые с помощью этих алгоритмов – поиск переломов, новообразований и инородных тел в медицине и поиск дефектов конструкций в промышленности. В основе решения указанных задач лежит применение различных алгоритмов сегментации, кластеризации, сравнения изображений, фильтрации, распознавания образов, сжатия данных, математической морфологии, выделения связных областей, интегральные преобразования изображений (их представление в частотной области с помощью дискретного преобразования Фурье, вейвлет-преобразования) и т.д. В свою очередь, многие алгоритмы распознавания, сравнения и сжатия изображений опираются на базовые алгоритмы линейной алгебры, такие как алгоритмы вычисления собственных значений, алгоритмы обращения матрицы, алгоритмы решения систем линейных алгебраических уравнений, арифметические операции с векторами и матрицами и др.

Все это позволило определить, какой функционал должен быть заложен в унифицированную библиотеку. По мере развития ПИК список доступных функций будет расширяться, в том числе и за счет создания новых алгоритмов с использованием уже имеющихся базовых функций.

При создании библиотеки использовались возможности бесплатного ПО (часть – с открытыми исходными кодами), которое позволяет использовать возможности GPU. Приведем список функций библиотеки, реализованных с применением сторонних средств:

- с использованием библиотеки MAGMA [6]:
  - вычисление собственных значений и собственных векторов;
  - вычисление обратных матриц (для матриц общего вида или же симметричных и положительно определенных);
  - решение обычных и переопределенных систем линейных алгебраических уравнений;
  - вычисление ковариационной матрицы;
- с использованием библиотеки OpenCV:
  - прямое и обратное преобразование Фурье для изображений;
  - сглаживание изображений с помощью функции Гаусса;
  - фильтрация изображений с помощью алгоритма MeanShift;
  - морфологические операции: эрозия, наращивание, открытие, закрытие;
  - выделение контуров с помощью методов Кэнни и Собеля;
  - выделение контуров с помощью масок Робертса: используется вспомогательная функция из библиотеки OpenCV, позволяющая проводить фильтрацию (свертку) изображения с произвольным прямоугольным ядром;
  - выделение контуров с помощью масок Превитт: используется вспомогательная функция из OpenCV, позволяющая проводить сепарабельную фильтрацию изображения с двумя одномерными масками (их произведение задает ядро оператора Превитт);
  - повышение резкости изображений: задача сводится к свертке исходного изображения со следующей маской: $Kern = Id + coef \cdot (Id - G)$, где $coef$ – некоторый коэффициент, Id – квадратная матрица, в ее центре находится единица, остальные элементы равны нулю. Матрица G – матрица гауссиана.
  - арифметические операции с матрицами и изображениями: поэлементное сложение, вычитание, умножение, деление;
  - цветояркостные преобразования (перевод изображения из одного цветового пространства в другое);
  - бинаризация изображений по порогу;

- вычисление гистограммы изображения; выравнивание гистограммы (повышение яркости изображения и одновременно повышение контрастности);
- масштабирование изображений;
- сегментация с помощью алгоритма MeanShift;

- с использованием исходного кода утилиты gpudwt [5]: прямые и обратные вейвлет-преобразования изображений с помощью биортогональных вейвлетов CDF 5/3 и CDF 9/7. Данный тип вейвлетов применяется, к примеру, в алгоритме сжатия изображений JPEG2000;
- с использованием исходного кода демонстрационных программ для архитектуры CUDA от компании NVIDIA: сглаживание изображений с помощью двухстороннего (bilateral) фильтра;
- с использованием исходного кода утилиты gpuquickshift [4]: сегментация изображений с помощью алгоритма QuickShift. Потребовалось модифицировать код утилиты и добавить несколько функций-оберток, чтобы обеспечить единый стиль программного интерфейса для всех функций библиотеки;
- с использованием исходного кода утилит, взятых с ресурса [3]: маркировка связных компонент на полутоновых изображениях. Код утилит был модифицирован и скомпилирован в динамически подгружаемую библиотеку. Также были написаны несколько функций-оберток для обеспечения единого стиля программного интерфейса.

Далее приведен список функций унифицированной библиотеки, которые были реализованы самостоятельно с применением архитектуры CUDA, и подробности реализации каждой функции.

- Вычисление ковариационной матрицы: используется в методе главных компонент. Исходные данные – набор из M векторов размерности N: $\left[\vec{X}_1, \vec{X}_2, \ldots \vec{X}_M\right]^T$, их можно представить в виде прямоугольной матрицы A, каждая строка которой – вектор из исходного набора, т.е. матрица имеет M строк и N столбцов. Вычисление ковариационной матрицы можно свести к операции перемножения двух матриц: $C = \frac{1}{M} B^T B$, где $B_{ij} = \frac{A_{ij} - P_j}{\sigma_j}$, P_i и σ_i – средние значения и среднеквадратические отклонения по каждой компоненте соответственно. Для перемножения матриц на GPU использовались возможности библиотеки MAGMA.
- Метод главных компонент: суть метода состоит в вычислении координат (проецировании) тестового набора векторов размерности N в базисе, образованном подмножеством из $P < N$ собственных векторов ковариационной матрицы, которая была вычислена по некоторым исходным данным. Выбираются собственные векторы, соответствующие наибольшим

собственным значениям ковариационной матрицы. Пусть тестовый набор состоит из M векторов, набор можно представить в виде матрицы A, в которой каждый вектор занимает отдельную строку. Подмножество используемых собственных векторов также можно представить в виде матрицы S (каждая строка – отдельный собственный вектор). Тогда операцию проецирования векторов можно представить в виде перемножения матриц:

$$A_{new} = B \cdot S^T, B = \begin{pmatrix} X_{11} - P_1 & X_{12} - P_2 & \dots & X_{1N} - P_N \\ X_{21} - P_1 & X_{22} - P_2 & \dots & X_{2N} - P_N \\ \dots & \dots & \dots & \dots \\ X_{M1} - P_1 & X_{M2} - P_2 & \dots & X_{MN} - P_N \end{pmatrix} = A - \begin{pmatrix} 1_1 \\ 1_2 \\ \vdots \\ 1_M \end{pmatrix} \begin{pmatrix} P_1 & P_2 & \dots & P_N \end{pmatrix},$$

где $\vec{P} = (P_1, P_2, \dots P_N)$ – вектор из средних значений по каждой компоненте. Процесс вычисления матрицы B распараллелен, каждый элемент матрицы A обрабатывается независимо. Для улучшения производительности элементы вектора $\vec{P}$ записываются в текстурную память GPU, поскольку доступ к ней кэшируется, а для обработки значений некоторого столбца матрицы используется одно и то же значение P_i. Для перемножения матриц используется библиотека MAGMA.

- Прямое и обратное вейвлет-преобразования с использованием вейвлетов Хаара: параллелизм достигается за счет независимой обработки каждого пикселя. Исходное изображение разбивается на фрагменты. Все GPU-нити, обрабатывающие изображение в рамках одного фрагмента, загружают фрагмент в разделяемую (shared) память. После загрузки фрагмента происходит заполнение результирующего изображения. Использование разделяемой памяти позволяет уменьшить количество чтений из глобальной памяти, которая обладает большей латентностью.
- Фильтрация шумов в частотной области, основанная на быстром преобразовании Фурье: задача сводится к поэлементному перемножению матрицы, которая является Фурье-образом исходного изображения, и матрицы *H(u, v),* соответствующей частотной функции фильтра. Фильтр низких частот имеет следующий вид:

$$H(u,v) = \begin{cases} 1, \textit{ если } D(u,v) < D_0 \\ 0, \textit{иначе} \end{cases},$$

где $D(u,v)$ – расстояние до центра частотной плоскости, а D_0 – числовой параметр, определяющий вид фильтра. Фильтр высоких частот получается путем инверсии идеального низкочастотного фильтра: $H'(u,v) = 1 - H(u,v)$. Также реализована частотная фильтрация, основанная на функции Гаусса. Низкочастотный гауссовский фильтр имеет вид

$H(u,v) = e^{-\frac{D^2(u,v)}{2D_0^2}}$. Высокочастотный фильтр Гаусса также получается путем инверсии низкочастотного.

- Сглаживание изображений с помощью фильтра Kuwahara [10]: фильтр позволяет сгладить изображения без размытия контуров (в отличие от фильтра Гаусса). Для ускорения вычислений, для исходного изображения предварительно вычисляются два интегральных изображения. Это позволяет быстро находить суммы значений и суммы квадратов значений пикселей внутри произвольной прямоугольной области внутри изображения. Оба интегральных изображения загружаются в текстурную память GPU, что позволяет уменьшить время доступа к их элементам, поскольку текстурная память кэшируется.
- Медианная фильтрация: как и во многих других типах фильтрации, каждый пиксель результирующего изображения обрабатывается независимо, а исходное изображение загружается в текстурную память. Задача нахождения медианы – частный случай задачи поиска *k*-й порядковой статистики.
- Нахождение полезного сигнала под уровнем шумов в режиме накопления данных: в этой задаче в качестве исходных данных выступает не одно, а несколько изображений одинакового размера. Все изображения получены из одного источника и содержат случайную шумовую составляющую. Существуют различные методы фильтрации шумов, основанные на аккумуляции: обычное среднее, медиана, усеченное среднее. Пусть имеется набор из N изображений: $M_1(x,y), M_2(x,y), \dots M_N(x,y)$. В случае обычного среднего результирующее изображение M_{res} формируется с использованием следующей формулы: $M_{res}(i,j) = \frac{1}{N}\sum_{k=1}^{N} M_k(i,j)$, в случае медианы: $M_{res}(i,j) = median\left(M_1(i,j) \dots M_N(i,j)\right)$, в случае усеченного среднего: $M_{res}(i,j) = \frac{1}{N_1}\sum_{k=1}^{N} S_k$, где S – отсортированный массив, образованный элементами $M_1(i,j), \dots M_N(i,j)$, из которого были удалены $d/2$ наибольших и $d/2$ наименьших значений (d – параметр работы алгоритма). Исходные изображения предварительно загружаются в глобальную память GPU, а параллелизм достигается за счет независимой обработки каждого пикселя исходного изображения.

Также предлагается следующий подход: для фильтрации выбираются не только значения пикселей $M_1(i,j), \dots M_N(i,j)$, а целая трехмерная область в виде параллелепипеда размером $R \cdot R \cdot N$, образованная следующими пикселями:

$\{M_k(p,q), k=1\ldots N, p=(i-R_1)\ldots(i+R_1), q=(j-R_1)\ldots(j+R_1)\}$, $R_1=\left\lceil\frac{R}{2}\right\rceil$, где R – настраиваемый параметр. В этом случае исходные изображения загружаются в текстурную память, так как для обработки каждого пикселя используются его соседи.

- Поиск характерных областей на изображениях с помощью анализа спектрографических текстур: оригинальный алгоритм анализа спектрографических текстур [2] был разработан силами Лаборатории интеллектуального управления. Текстурный классификатор, лежащий в основе алгоритма, не рассматривает точки на снимке как отдельные совокупности спектральных значений. Вместо этого оцениваются группы близлежащих точек, формирующие спектрографические текстуры. Точки результирующего изображения (каждой точке соответствует геометрический центр анализируемой текстуры) окрашиваются в тот или иной цвет в зависимости от степени близости к региону интереса. Набор регионов интереса задается экспертом как обучающая выборка. Применение классификатора спектрографических текстур обеспечивает нахождение сложно выявляемых объектов, что недостижимо для классификаторов, работающих с отдельными точками изображения.

Параллелизм достигается за счет независимой обработки каждого пикселя. Исходное изображение разбивается на блоки, каждый блок разделяется в разделяемую память, доступную всем GPU-нитям, которые обрабатывают все пиксели этого блока. Использование разделяемой памяти позволяет сократить число обращений к глобальной памяти.

- Бинаризация с помощью адаптивного метода Оцу: метод состоит из трех операций – вычисление гистограммы исходного изображения, вычисление значения порога и собственно бинаризация с помощью пороговой обработки. Первая и третья операции выполняются с помощью уже имеющихся функций, вторая операция выполняется на CPU, так как общий объем данных и общий объем вычислений невелики, что делает неэффективным ее реализацию на GPU.
- Преобразование яркости и контраста, инвертирование изображений: задача сводится к линейному преобразованию значений каждого пикселя исходного изображения, по формуле $y=ax+b$. Кроме того, возможно проводить модификацию исходного изображения, без дополнительных затрат памяти. Наличие этой и двух последующих функций особенно актуально для медицинских и промышленных снимков, поскольку очень часто они отличаются слабой яркостью или контрастностью. Исходное и результирующее изображения загружаются в глобальную память GPU, каждый пиксель обрабатывается независимо.
- Гамма-коррекция: значение каждого пикселя исходного изображения преобразуется по формуле $y=cx^{\lambda}$.

- Изменение яркости изображения с помощью логарифмической функции: значение каждого пикселя исходного изображения преобразуется по формуле $y = c \cdot \ln(x+1)$.

- Преобразование полутонового изображения в цветное: раскрашивание изображений в псевдоцвета может быть полезно тем, что человек способен находить нужные ему области на цветных изображениях быстрее, чем на полутоновых. Каждому оттенку серого цвета (всего их 256) сопоставляется определенный цвет из цветового пространства RGB или другого. На входе также имеется некоторый набор из N базовых цветов ($N < 256$). Затем диапазон цветов расширяется до 256, с применением линейной интерполяции. Формируется вектор *colors* из 256 троек: $R_0, G_0, B_0, R_1, G_1, B_1, \ldots R_{255}, G_{255}, B_{255}$. Значения пикселей результирующего изображения вычисляются по следующей формуле: $dst(x, y) = colors(src(x, y))$. Компоненты вектора *colors* вычисляются на CPU, так как объем вычислений и данных невелик. Затем вектор копируется в константную память GPU, так как доступ к константной памяти кэшируется и обладает меньшей латентностью, чем глобальная или текстурная память.

Заключение

Предлагаемая программная библиотека является важным компонентом разрабатываемого программно-инструментального комплекса, предназначенного для высокопроизводительной обработки изображений медицинского и промышленного назначения. Библиотека содержит как типовые (базовые), так и оригинальные алгоритмы обработки изображений, а также алгоритмы линейной алгебры. При создании библиотеки использовались возможности бесплатного стороннего ПО, для реализации части функций на GPU использовалась архитектура CUDA. Все использованное стороннее ПО является кросс-платформенным. В дальнейшем планируется провести всестороннее тестирование функций библиотеки, а также включить ее в программно-инструментальный комплекс. Состав библиотеки будет расширяться и дополняться новыми функциями.

Работа поддержана Министерством образования и науки Российской Федерации (Государственный контракт № 14.514.11.4056 по теме «Разработка программно-инструментального комплекса высокопроизводительной обработки изображений медицинского и промышленного назначения»)

Список литературы

1. Официальный сайт МИКРОКОН РАД. - URL: http://microkon.rosbizinfo.ru/products/37.html (дата обращения: 26.06.2013).

2. Фраленко В.П. Анализ спектрографических текстур данных дистанционного зондирования Земли // Искусственный интеллект и принятие решений. – 2010. – № 2. – С. 11-15.

3. Connected Component Labeling. - URL: https://github.com/foota/ccl (дата обращения: 26.06.2013).

4. Fulkerson B., Soatto S. Really quick shift: Image segmentation on a GPU // In Proceedings of the Workshop on Computer Vision using GPUs, held with the European Conference on Computer Vision. – 2010. – September. – 9 p. - URL: http://vision.ucla.edu/~brian/papers/fulkerson10really.pdf (дата обращения: 26.06.2013).

5. Gpudwt utility. - URL: https://code.google.com/p/gpudwt/ (дата обращения: 26.06.2013).

6. MAGMA project. - URL: http://icl.cs.utk.edu/magma/ (дата обращения: 26.06.2013).

7. Official site of Magellan image processing software. - URL: http://www.imagingdynamics.com/content/view/18/22/ (дата обращения: 26.06.2013).

8. Official site of OpenCV library. - URL: http://opencv.willowgarage.com (дата обращения: 26.06.2013).

9. Official site of TiViPE software. URL: - http://www.tivipe.com/index.php?option=com_content&view=article&id=66&Itemid=81 (дата обращения: 26.06.2013).

10. The Kuwahara Filter. URL: http://en.wikipedia.org/wiki/User:DinoVgk (дата обращения: 26.06.2013)

Рецензенты:

Славин Олег Анатольевич, д.т.н., заведующий лабораторией, ФГБУН «Институт системного анализа» Российской академии наук (ИСА РАН), г. Москва.

Хачумов Вячеслав Михайлович, д.т.н., профессор, заведующий лабораторией, ФГБУН Институт системного анализа Российской академии наук (ИСА РАН), г. Москва.