УДК 00.004.021; 00.004.4; 00.004.9

ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ ОБРАБОТКИ ИЗОБРАЖЕНИЙ В МАССИВНО-МНОГОПОТОЧНОЙ CUDA-СРЕДЕ

Бибиков С.А., *Самарский государственный аэрокосмический университет, ssau@ssau.ru*
Никоноров А.В., *к.т.н., доцент Самарского государственного аэрокосмического университета, E-mail: adminmcdk.org*
Фурсов В.А., *д.т.н., профессор Самарского государственного аэрокосмического университета, E-mail: fursov@ssau.ru*
Якимов П.П., *магистрант Самарского государственного аэрокосмического университета, E-mail: pavel.yakimov@hotmail.com.*

Постановка задачи

Современная технология офсетной печати позволяет выпускать продукцию очень высокого качества. Возможна печать многокрасочных публикаций, информационной, рекламной продукции, а также репродукционное воспроизведение живописных полотен.

Необходимость проведения предпечатной обработки изображений при производстве цифровых репродукций произведений живописи связана со спецификой фотографирования оригиналов. В процессе фотографирования возникают следующие артефакты: множественные блики, затенения части холста рамкой и так называемые матовые блики. Артефакты обусловлены неровностями нанесения краски на холст и способом установки дополнительного освещения.

В данной работе рассматривается алгоритм цифровой обработки изображений для удаления множественных бликов, которые возникают вследствие того, что отдельные малые участки слоя краски обладают таким углом наклона поверхности, что падающий свет источника отражается и регистрируется камерой в виде пятен. На цифровом изображении такие артефакты выглядят как области небольшого размера, обладающие выделяющейся на общем фоне яркостью. Пример изображения с артефактами представлен на рис 1.

Рис. 1. Пример цифровой копии с точечными бликами.

Если обработку изображения будет проводить специалист по компьютерной графике с использованием стандартных возможностей программы Photoshop либо аналогичной, то для качественной ретуши изображения формата А3 ему понадобится примерно 2-3 часа времени. В настоящей работе описываются разработанные методы, алгоритмы и созданный на их основе программный комплекс, обеспечившие возможность высокой степени автоматизации процессов распознавания, локализации и цветовой коррекции множественных точечных артефактов на цифровых изображениях произведений живописи в процессе их предпечатной подготовки.

Указанные свойства разработанного программного комплекса достигаются за счёт перспективной архитектуры, реализация которой основана на следующих соображениях. В последние годы наряду с совершенствованием и ростом производительности мощных суперкомпьютеров наметилась устойчивая тенденция развития технологий параллельных вычислений на персональных компьютерах. Эта тенденция обусловлена существенным технологическим прорывом, по крайней мере, в трёх направлениях.

Во-первых, это появление настольных CPU с возможностью параллельного выполнения программ. В настоящее время для персональных компьютеров доступны процессоры 7 с параллельным исполнением 8-ми потоков. Вторым технологическим прорывом является представленное в 2007 году компанией Nvidia альтернативное направление развития массивно-параллельных вычислений для настольных компьютеров с использованием GPU. Третий фактор, позволяющий всерьез говорить о настольных суперкомпьютерных системах, – это повышение скорости коммуникаций по системной шине для платформы x86 с развитием стандарта PCI Express. Скорость коммуникаций между узлами всегда была узким местом кластерных систем, снижающим их эффективность. Принятие подавляющим большинством производителей стандарта *Infiniband* позволяет строить весьма недорогие вычислительные системы со скоростью обмена данными между узлами до 5 GB/s.

Совокупность указанных факторов стала мотивом возрастающего интереса к созданию компактных, недорогих и, вместе с тем, высокопроизводительных систем обработки и анализа изображений нового поколения для систем видеоконтроля, видеонаблюдения, подготовки цифровых изображений к печати и др. В работах [1-3] рассматривались примеры создания компактных распределенных систем обработки изображений. В частности, в работе [1] описана реализация алгоритма коррекции цветных цифровых изображений в локальной сети компьютеров посредством специально разработанной GRID-системы, реализованной на Java с native-модулями. В этой работе показана возможность существенного ускорения одного из наиболее затратных по времени процессов подготовки цветных изображений к печати. В работах [4-5] приведены примеры реализации описанного выше алгоритма с помощью технологии CUDA.

В описываемом в настоящей статье программном комплексе высокое быстродействие и связанная с этим высокая степень автоматизации процессов распознава-

ния, локализации и цветовой коррекции множественных точечных артефактов на цифровых изображениях достигается за счёт построения эффективных рекуррентных процедур обработки данных.

Алгоритм устранения точечных бликов

Цифровое изображение может быть представлено как двумерная функция вида $f(x,y)$, где x и y – координаты элемента изображения на плоскости. Значение f задаётся парой координат (x,y) и называется интенсивностью или уровнем серого (яркости) изображения в этой точке. Значения функции яркости $f(x,y)$, как правило, принимают дискретные значения в диапазоне от 0 до 255 (8-битное кодирование), иногда – от 0 до 65535 (16-битное кодирование).

В настоящей работе цветные изображения представляются в пространстве RGB, поэтому в действительности мы имеем дело с тремя функциями яркости – $f_R(x,y)$, $f_G(x,y)$ и $f_B(x,y)$. При этом функция общей яркости или светлоты $L(x,y)$ определяется согласно [6], как

$$L(x,y) = 0,3 f_R(x,y) + 0,59 f_G(x,y) + 0,11 f_B(x,y), \qquad (1)$$

для которой используется нормированное значение яркости в диапазоне от 0 до 1.

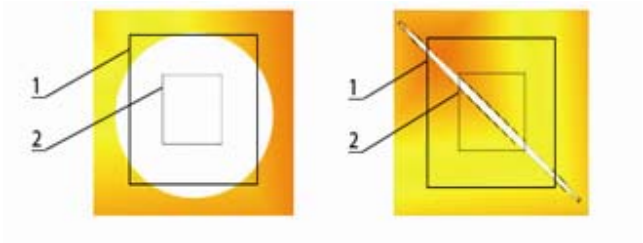
Задача цветовой коррекции множественных точечных бликов естественным образом распадается на следующие два этапа:

- обнаружение точечных артефактов на изображении;
- замещение цвета пикселей в области блика на некоторый цвет, близкий к цвету пикселей окрестности блика (компьютерное ретуширование).

Распознавание и локализация артефактов

Под точечным артефактом понимаются относительно небольшие компактные области с высокой (по относительному или абсолютному значению) яркостью. Поэтому распознавание и локализация артефактов проводится по значениям светлоты, описываемой значениями функции (1). При этом в качестве отличительных характеристик используются высокая относительная и/или абсолютная яркость области в совокупности с компактностью и небольшой плотностью области.

Для определения указанных характеристик используется алгоритм, осуществляющий «просмотр» всего изображения с использованием локальной области, включающей опорное окно и окно поиска. На рис. 2,а приведён пример, когда признаки наличия блика выполняются, а на рис. 2,б,в – случаи, когда условия наличия точечного блика не выполняются. Далее приводятся математически формализованные признаки существования точечного блика.



а) опорное окно б) окно поиска

Рис. 2. Слева пример нарушение размера, справа нарушение компактности

Пусть W_1 – область изображения, покрытая бликом, W_0 – некоторая окрестность области блика. Рассмотрим функцию принадлежности $I_0(x,y)$:

$$\left.\begin{array}{l} I_0(x,y) = 1, \quad (x,y) \in W_0 \\ I_0(x,y) = 0, \quad (x,y) \notin W_0 \end{array}\right\}. \qquad (2)$$

Тогда для блика W_0 должны выполняться следующие требования:

$$\max_{(x,y)\in W_1} f(x,y) \geq f_1, \qquad (3)$$

$$\frac{\underset{(x,y)\in W_1}{M} f(x,y)}{\underset{(x,y)\in W_0}{M} f(x,y)} \geq M_1, \qquad (4)$$

$$\sum I(x,y) \leq S_1, \qquad (5)$$

$$D(W_1) \leq D_1. \qquad (6)$$

Здесь $M(f(x,y))$ – среднее значение функции яркости в области, $D(W_1)$ – диаметр области, величина, характеризующая степень компактности области. Диаметр области определим, как максимальное евклидово расстояние между точками области:

$$D(W_1) = \max_{(x_1,y_1),(x_2,y_2)\in W_1} \left\| (x_1,y_1),(x_2,y_2) \right\|_2. \qquad (7)$$

Соотношения (3) и (4) отражают требования к максимальной и относительной яркости блика, а (5) и (6) – требование малой площади и компактности области. Константы f_1, M_1, S_1, D_1 являются определяющими для разрабатываемых алгоритмов и могут жестко задаваться экспертом, либо настраиваться в автоматизированном режиме.

Замещение артефактов

Множественные точечные блики характеризуются почти полной потерей информации о цвете, поэтому задача цветовой коррекции в данном случае сводится к замещению цвета пикселей в области блика на некоторый цвет, близкий к цвету пикселей в окрестности блика. В разработанной системе используется технология замещения пикселей блика, основанная на идентификации параметров модели замещения. Технология реализована в предположении, что цвет задан в пространстве *RGB*. Для определения значений цветовых компонентов в области блика строятся оценки коэффициентов поверхности *M*-го порядка, «опирающейся» на участки поверхности, образованной функциями яркости компонентов в окрестности блика:

$$z_k(x,y) = \sum_{i}^{M}\sum_{j}^{M} C_k^{ij} x^i y^j, \qquad (8)$$

где $z_k(x, y)$ – значение *k*-го цветового компонента в точке (x, y), C_k^{ij} – коэффициенты полинома, подлежащие оцениванию.

Замещение пикселей поверхности (8) может привести к тому, что блик останется заметным в силу гладкости поверхности. Поэтому на область блика дополнительно клонируется текстура из окрестности блика. Затем выполняется размытие по Гауссу в области, чуть большей, чем область блика. Размытие позволяет избежать резких границ между ретушированным участком и его окрестностью.

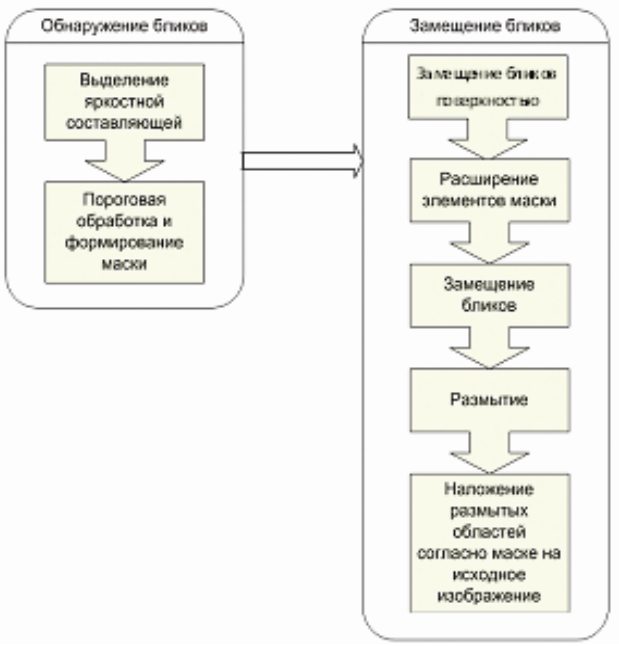


Рис. 3. Общая схема этапов локализации блика

На рис. 3 приведена общая схема этапов локализации блика и замещения пикселей в области блика.

Параллельная реализация алгоритма устранения точечных бликов

В приведённом выше алгоритме обнаружения и локализации бликов наиболее ресурсоёмкой в вычислительном отношении являтся процедура «просмотра» изображения скользящим окном поиска. Известны быстрые рекуррентные (например, параллельно-рекурсивные) алгоритмы обработки изображений скользящим окном, реализованные на традиционных аппаратных средствах [6]. Создание таких же эффективных процедур на основе CUDA-технологий, реализуемых с использованием графических ускорителей NVIDIA, к сожалению, пока еще остается проблемной задачей. Связано это с тем, что вычислительная среда CUDA имеет более сложную, с точки зрения программирования, структуру по сравнению с многопоточной CPU средой или параллельной средой MPI. Основные структурные особенности среды CUDA следующие.

Во-первых, CUDA-среда имеет двухуровневую многомерную топологию исполнения потоков, которая может описывать десятки тысяч потоков. По аналогии с термином массивно-параллельная среда такую среду можно назвать массивно-многопоточной.

Во-вторых, потоки могут обмениваться данными только посредством сравнительного небольшого объема общей памяти. Такой обмен возможен в пределах небольшой области локальной связности – блока, включающего (для выпускаемых в настоящее время GPU) не более 512 потоков. Кроме того, когерентность данных в общей памяти достигается только путем приостановки всех потоков, т.е. упорядоченный обмен данными невозможен. Указанные ограничения создают серьёзные проблемы при реализации в среде CUDA рекурсивных алгоритмов (сжатия данных, фильтрации и др.) [7].

Наконец, ещё одной важной особенностью является то, что в CUDA-среде можно работать с несколькими видами памяти, имеющими разную область видимости и скорость доступа. Крайне важным при доступе к глобальной памяти является согласование (coalescing) адреса памяти с номером активного потока. Если условие согласования выполняется, то GPU может объединить несколько запросов в одну транзакцию [14]. Несоблюдение этого условия может привести к падению производительности почти в 8 раз [8, 9].

Многие базовые алгоритмы обработки данных требуют существенной переработки с учётом указанных особенностей. Например, в работе [10] рассмотрен адаптированный для CUDA-среды алгоритм префиксного суммирования. В настоящей статье, являющейся развитием работ [1-5], из всего комплекса рассмотренных в них задач выделены наиболее характерные и сложные для реализации на GPU-процессорах: обработка изображений скользящим окном и организация передачи видеоданных в реальном времени. В частности, рассмотрены основные особенности построения указанных алгоритмов и приводится краткое описание созданного в массивно-многопоточной CUDA-среде программного комплекса.

Продемонстрируем особенности реализации рекуррентных вычислительных алгоритмов в CUDA-среде на примере простейшего локального оконного фильтра, выполняющего расчёт среднего значения. Простота этого алгоритма позволяет наглядно продемонстрировать все основные особенности среды CUDA. Вместе с тем, иллюстрируемый на этом простом примере подход может быть легко применен к более сложным локальным оконным фильтрам, применяемым в различных технологиях обработки изображений.

Пусть алгоритм обработки скользящим окном состоит в вычислении для каждой точки изображения суммы значений отсчётов в некоторой окрестности (окне):

$$s_{x,y} = \sum_{i,j=1}^{m} L_{x+i,y+j}, \quad 0 < x, y \le N, \qquad (9)$$

где N – размер изображения, m – размер окна (для простоты полагаем, что изображение и окно квадратные). При этом потребуется $O_1 = N^2 m^2$ (10) арифметических операций.

При рекуррентной реализации этих вычислений в CUDA-среде существенное значение играет согласование адреса памяти с номером активного потока. Будем считать обращение потока к глобальной памяти согласованным, если оно происходит к словам памяти размером 32 бита с адресами

$$I(t,k) = 32 * n * k + t, \quad k = 0,1,2..., \quad t = 0,1,2,..., \qquad (11)$$

где $I(k,t)$ – адрес ячейки глобальной памяти, к которой производится обращение, n – натуральное число, характеризующее особенности конкретной задачи, связанные с количеством потоков в блоке; t – номер потока внутри блока, k – номер шага некоторой итерации алгоритма, одинаковый для всех потоков внутри блока. Приведённые в спецификации CUDA [8] требования согласованности несколько более сложные и к тому же различаются для различных реализаций стандарта CUDA. Тем не менее, легко проверить, что выполнение требования (11) для всех потоков внутри блока позволяют получить согласованный доступ к памяти в смысле спецификации [8].

Рекуррентный алгоритм, удовлетворяющий условию согласованности (11), может быть построен, например, как двухпроходный. При этом сначала выполняется вычисление частичных сумм по столбцам:

$$s^1_{x,y} = s^1_{x,y-1} - L_{x,y-m} + L_{x,y}, \; s_{x,1} = \sum_{j=1}^{m} L_{x,y+j}, \qquad (12)$$

а окончательная сумма получается суммированием по строкам:

$$s_{x,y} = s_{x-1,y} - s^1_{x-1,y} + s^1_{x,y},\ s_{1,y} = \sum_{j=1}^{m} s^1_{x+j,y} \qquad (13)$$

Вычисления по такой схеме требуют

$$O_1 = (2N - m)^2 \qquad (14)$$

арифметических операций.

В CUDA-среде исходные данные располагаются в глобальной памяти таким образом, что строки изображения располагаются последовательно. Вычисление частичных сумм (12) удовлетворяет условию (11), если длина строк изображения кратна 32 пикселям. Если это не так, строки могут быть дополнены до требуемого размера, поэтому далее полагаем, что

$$N \bmod 32 = 0\,. \qquad (15)$$

При последующем суммировании по строкам (13) требование (11) не выполняется, что приводит к существенному снижению скорости вычислений. Это подтверждается данными эксперимента. Рассмотрим модификацию алгоритма построчного суммирования, обеспечивающую согласованное обращение к глобальной памяти за счёт использования при проведении суммирования локальной и общей памяти потоков.

Текущая спецификация CUDA обеспечивает всего 16384 байт общей памяти, а каждый блок содержит не более 512 потоков. Небольшая часть этой памяти – около 20-30 байт расходуется на хранение переданных ядру аргументов. В таком объёме невозможно записать изображение целиком, поэтому предлагается итерационная схема блочной обработки изображения.

В каждом эпизоде обрабатывается блок из 32 строк изображения, при этом удобно включать в блок 32 потока. С учётом того, что данные хранятся в формате с плавающей точкой и одинарной точностью, для такой обработки потребуется 32×32×4=4096 байт общей памяти на блок. Заметим, что в случае необходимости предлагаемая блочная реализация позволяет с использованием имеющегося объёма общей памяти выполнять вычисление нескольких показателей одновременно.

В предлагаемой схеме потоки группируются в блоки по 32 потока, общее число блоков, с учётом (15), равно N/32. Каждый поток текущего блока считывает в общую память данные из столбца изображения в глобальной памяти, т.е. каждый поток последовательно читает данные из соседних столбцов изображения. Таким образом, за одну итерацию все блоки обрабатывают 32 строки изображения, формируя в общей памяти фрагмент изображения размерностью 32*N. Каждую итерацию чтения можно описать следующим выражением:

$$s[k*N+32*b+t] := d[K*N*32+k*N+32*b+t],\ \ k = \overline{0,31}, \qquad (16)$$

где s – указатель на область общей памяти, d – указатель на данные изображения в глобальной памяти, k – номер шага итерации, K – номер итерации, b и t индексы блока и потока соответственно.

После синхронизации потоков, каждый поток начинает суммирование данных согласно (13) «вдоль» строки изображения в общей памяти, при обращении к которой согласованности не требуется. Промежуточные данные сохраняются в локальной памяти потока. После выполнения вычислений потоки синхронизируются, а данные из общей памяти записываются в глобальную память по столбцам, аналогично операции чтения (16). При такой схеме поток читает и пишет в глобальную память данные «транспонированные» по отношению к тем, над которыми поток выполняет вычисления. Схема организации взаимодействия потоков с локальной и глобальной памятью показана на рис. 4. Описанный рекуррентный алгоритм суммирования практически без существенных изменений может использоваться для реализации широкого круга параллельно-рекурсивных методов [11] обработки изображений.

Ниже приводятся результаты сравнения эффективности предложенных реализаций. Эксперименты проводились на тестовом изображении размером 1024×1024 для окон различных размеров. В первом случае декомпозиция проводилась на 2048 блоков по 512 потоков, часть которых запускалась не одновременно. Во втором и третьем случае запускалось одинаковое количество потоков – 1024, при этом они группировались в 32 блока по 32 потока в каждом. Тестирование проводилось для CPU PIV 3Ghz и GPU Nvidia GF 9500. Результаты представлены в таблице 1.

Таблица 1.
Время выполнения различных алгоритмов для разных размеров окна (*m*).

	m = 5	m = 9	m = 15
1. CPU, не рекуррентный	283 мс	922 мс	2 531 мс
2. CPU, рекуррентный	142 мс	297 мс	733 мс
3. GPU, не рекуррентный	78 мс	262 мс	473 мс
4. GPU, рекуррентный	23 мс	25 мс	27 мс
5. GPU, рекуррентный с оптимизацией обращений к памяти	12 мс	12 мс	12 мс

Как видно из таблицы 1, результаты для рекуррентной CUDA-реализации практически не зависят от *m*, а зависят только от стратегии использования памяти.

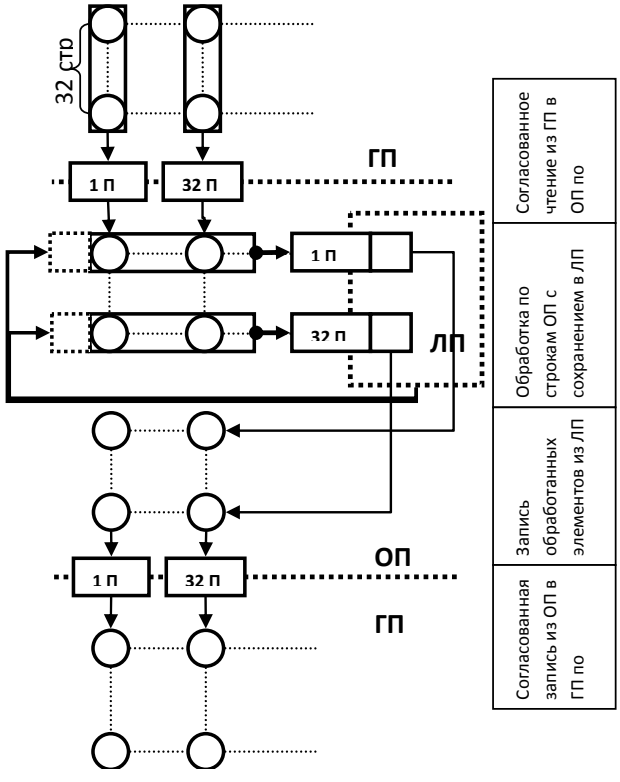


Рис. 4. Схема организации взаимодействия потоков с памятью

Рис. 5. Архитектура программного комплекса

Архитектура системы обработки изображений

Первый уровень – это интерфейс пользователя. Он может быть реализован в виде модуля расширения существующей системы, либо в виде независимого интерфейса работы с файлами растровых изображений в формате tiff или jpeg. Средний уровень – реализация графического интерфейса (GUI) обработки изображений представляет собой унифицированный графический интерфейс для любых режимов работы.

Графический интерфейс GUI связан с алгоритмической частью посредством динамической линковки. Тем не менее, в данном случае для создания GUI используется кросплатформенная C++ библиотека. В настоящее время известно несколько таких библиотек. Среди них это QT, GTK и библиотека wxWidgets [12]. На наш взгляд, wxWidgets обладает большей концептуальной чистотой и гибкостью в использовании, что предопределило её использование в настоящей работе.

Третий уровень, не зависящий от других уровней, – это динамические библиотеки, в которых реализованы вычислительные алгоритмы. В настоящей работе описан алгоритм удаления точечных бликов. Однако архитектура позволяет подключать и другие алгоритмы обработки изображений. Если используемый алгоритм вычислительно сложен и доступны аппаратные средства, поддерживающие технологию CUDA, алгоритмы могут вызывать процедуры реализованные для GPU. Такой подход позволяет использовать для вычислений не только GPU Nvidia, но и GPU видеопроцессоры AMD (ATI). Заметим, что вычислительные процедуры могут также выполняться в несколько потоков на CPU с использованием OpenMP или OpenCL [13]. Кроме того, система легко масштабируется с использованием обычных схем декомпозиции изображений.

Рис.6. Графический интерфейс программного комплекса

В процессе работы системы модуль GUI вызывается динамически основным модулем расширения в качестве модуля расширения. В случае автономной работы программного комплекса GUI является стартовой точкой приложения и работает как самостоятельный исполняемый файл.

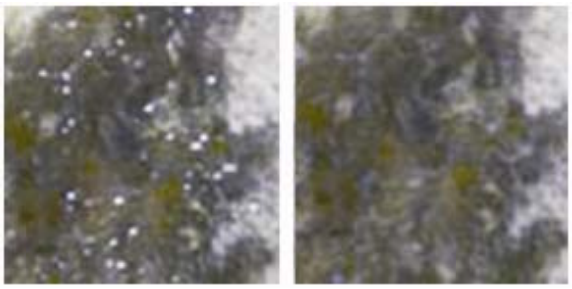
Наиболее гибкой платформой, позволяющей вести разработку модулей расширения существующих систем, а также разработку под платформу CUDA, является C. Кроме того, C позволяет создавать многопоточные и распределённые программы с использованием технологий OpenMP и MPI. Поэтому, несмотря на существование других более комфортных для разработки систем платформ, все описанные алгоритмы реализованы на C. Интерфейсная часть программы выполнена на C++.

Реализация программного комплекса и результаты экспериментов

Описанная выше схема взаимодействия потоков с памятью при реализации рекуррентного алгоритма использовалась при создании последних версий программного комплекса локализации и коррекции бликов на цифровых изображениях репродукций произведений живописи. Первые версии этого программного комплекса описаны в работах [2-5]. Программный комплекс имеет широкий набор функций для работы с цифровыми изображениями. Приложение работает в режиме stand alone, т.е. не требует установки дополнительного программного обеспечения. Благодаря использованию кроссплатформенных библиотек wxWidgets комплекс работает как на платформе Windows, так и на Unix-подобных операционных системах.

На рис. 6 представлен вид GUI, разработанного приложения в режиме автоматизированного поиска точечных артефактов (точки на экране указывают найденные блики).

В последней версии параметры поиска могут задаваться как вручную, так и автоматически путём перехода в режим адаптивного определения параметров. Кроме того, имеется возможность выполнять масштабирование изображения, выбор области поиска, а также модифицировать маску с отмеченными артефактами, сформированную алгоритмом, вручную. На рис. 7 приведены изображения, иллюстрирующие результаты устранения бликов с использованием разработанного программного комплекса.



а) *б)*

Рис. 7. Пример работы программной системы по устранению бликов на изображении «лес»: а) исходное, б) обработанное

В работах [4] и [5] авторами была показана возможность существенного ускорения алгоритмов обработки изображений в программной системе локализации и устранения бликов за счёт использования универсальных графических процессоров NVIDIA и технологии CUDA. В частности, на графических ускорителях NVIDIA GeForce 8400 и 9500 с 8 мультипроцессорами, было продемонстрировано более чем десятикратное ускорение CUDA реализации по сравнению с реализацией на CPU. В настоящей работе мы демонстрируем возможность дальнейшего повышения ускорения за счёт использования в программной системе оптимизированных рекуррентных алгоритмов.

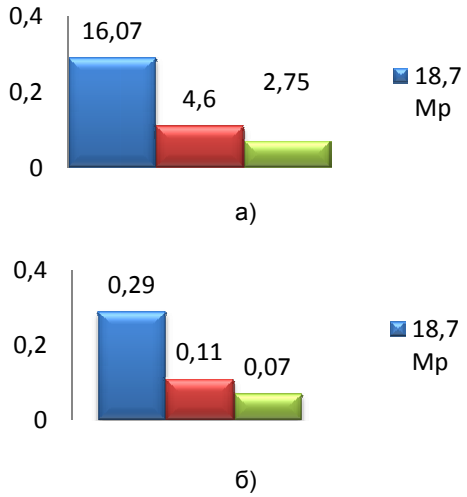


Рис. 8. Время расчёта оптимизированного алгоритма (с). а) алгоритм, опубликованный в работе 4;б) результаты, оптимизированного рекуррентного алгоритма

Для экспериментальной проверки скорости использовались три изображения размером 3.2, 5.8 и 18.7 мегапикселей. Для расчётов использовалась видеокарта Nvidia GeForce 9500. На рис. 8,а приведено время вычислений для алгоритма из работы [4], на рис. 8,б приведено время вычислений для алгоритма поиска бликов на изображении, использующего оптимизированную рекуррентную процедуру обработки скользящим окном. Из приведённых диаграмм видно, что в среднем, по трём экспериментам, достигнуто ускорение более чем в 45 раз.

Заключение

Подчеркнём, что полученное в эксперименте ускорение получено исключительно за счёт оптимизации CUDA алгоритма. Достигнутые показатели скорости обработки с использованием оптимизированной версии алгоритма поиска бликов открывают возможности для их более широкого применения. В частности, наряду с использованием этого алгоритма в задачах предпечатной обработки изображений, как это было сделано в [4], по-видимому, удастся использовать эту технологию также для обработки видео данных в реальном времени.

Работа выполнена при финансовой поддержке РФФИ (проект № 09-07-00269-а) и программы Президиума РАН

Литература

1. А.В. Никоноров, В.А. Фурсов. Предоставление сервиса управления цветовоспроизведением цветных изображений в сети интернет. Труды XIV Всероссийской научно-методической конференции «Телематика 2007», С.-Петербург, 18-21 июня, 2007, С. 412-413.
2. Никоноров А.В., Фурсов В.А. Распределенная вычислительная среда коррекции цветных изображений. Тр. XV Всероссийской научно-методической конференции «Телематика 2008», С.-Пб, 23-26 июня, 2008, С. 88-89.
3. В.А. Фурсов, А.В. Никоноров. Распределённый алгоритм коррекции точечных артефактов на цветных изображениях // Труды четвёртой международной конференции «Параллельные вычисления и задачи управления» (PACO’ 2008), Москва, 27-29 октября, 2008, 1087.
4. С.А. Бибиков, А.В. Никоноров, В.А. Фурсов, П.Ю. Якимов. Исследование эффективности технологии CUDA в задаче распределённой предпечатной подготовки цифровых изображений. Труды Всероссийской суперкомпьютерной конференции «Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность», 21-26 сентября 2009 г., г. Новороссийск. – М.: Изд-во МГУ, 2009. – С. 204-207.
5. С.А. Бибиков, А.В. Никоноров, В.А. Фурсов, П.Ю. Якимов. CUDA-технология цветовой коррекции теневых искажений на цифровых фотокопиях произведений живописи. Сб. трудов международной научной конференции «Параллельные вычислительные технологии’ 2010» Уфимский государственный авиационный технический университет, г. Уфа, 29 марта – 2 апреля 2010 г., С. 656.
6. Сойфер, В.А. и др., Методы компьютерной обработки изображений. – М.: Физматлит, 2003. – С. 784.
7. Nickolls J., Buck I., Skadron K., Scalable Parallel Programming with CUDA, // ACM Queue, VOL 6, No. 2 (March / April 2008) 40-53 pp. http://www.mags.acm.org/queue/20080304/?u1=texterity.
8. NVIDIA CUDA Programming Guide, 2010, 130 p.
9. NVIDIA CUDA Best Practices Guide, 2009, 68 p.
10. Harris M., Parallel Prefix Sum (Scan) with CUDA, 2008, 21 p.
11. Мурызин С.А., Сергеев В.В., Фролова Л.Г. Исследование эффективности двумерных параллельно-рекурсивных КИХ-фильтров // Компьютерная оптика. – М.: МЦНТИ, 1992. – Вып. – 12. – С. 65-71.
12. Smart, J. Cross-platform GUI Programming with wxWidgets / J. Smart – Prentice Hall, 2000. – 714 p.
13. Munshi, A. Kronos OpenCL Working Group, The OpenCL Specification. – 2009. – 314 p.