УДК 004.78

В.А. КОЛБАСИН, ассистент НТУ «ХПИ»,
Н.С. ШИРЯЕВА, студентка НТУ «ХПИ»

ИСПОЛЬЗОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ РЕСУРСОВ ВИДЕОКАРТ ПРИ СОЗДАНИИ КЛИЕНТ-СЕРВЕРНОЙ СИСТЕМЫ ОБРАБОТКИ РАСТРОВЫХ ИЗОБРАЖЕНИЙ

У статті запропоновано архітектуру сервера обробки растрових зображень, здатного використовувати обчислювальні ресурси головних процесорів системи та відеокарт. Запропоновано підхід до забезпечення прямої та зворотної сумісності модулю обробки зображень та сервера. Розглянуто застосування данної архітектури для створення веб-застосування створення та обробки зображень з розширеним динамічним діапазоном.

В статье предложена архитектура сервера обработки растровых изображений, способного использовать вычислительные ресурсы основных процессоров системы и видеокарт. Предложен подход к обеспечению прямой и обратной совместимости модуля обработки изображений и сервера. Рассмотрено применение данной архитектуры для создания веб-приложения создания и обработки изображений с расширенным динамическим диапазоном.

The architecture of image processing server that can use central processor and video card processors is proposed. The approach to provide forward and backward compatibility between image processing module and other server modules is proposed. Proposed architecture using for development high dynamic range image processing web-application is considered.

Введение. Методы и средства цифровой обработки изображений (ЦОИ) широко используется в медицине, фотографии, предпечатной подготовке и других областях человеческой деятельности. Однако при практическом использовании ресурсоемких операций ЦОИ, примером которых может служить обработка фотографий формата RAW, повышение локальной контрастности медицинских изображений, восстановления изображения с расширенным динамическим диапазоном (HDR), возникает ряд проблем. Чтобы такие операции выполнялись быстро, необходимо увеличить вычислительную мощность компьютера, установленного на рабочем месте оператора, что в случае непостоянного использовании данных операций невыгодно. Но с другой стороны, длительное ожидание результата при использовании маломощного компьютера вызывает дискомфорт оператора и приводит к желанию избежать использования ресурсоемких операций, что в конечном итоге снижает эффективность работы.

В том случае, если потребность в выполнении ресурсоемких операций ЦОИ возникает на нескольких рабочих местах, проблему можно решить установкой в локальной сети предприятия достаточно мощного сервера, который будет выполнять операции обработки изображений. При этом в силу определенных особенностей организации алгоритмов ЦОИ достаточную вычислительную мощность сервера можно обеспечить за счет использования

более дешевых вычислительных средств SIMD-архитектуры [1], таких как платы Tesla и видеокарты, поддерживающие использование их процессоров для неграфических вычислений при помощи технологий CUDA и CTM [2].

Наиболее перспективным представляется организация доступа к серверу приложений при помощи web-интерфейса, так как в этом случае не требуется развертывать на рабочих станциях клиентское программное обеспечение.

Таким образом, данная работа посвящена созданию архитектуры клиент-серверной системы для ресурсоемкой обработки растровых изображений, способной использовать платы ускорения вычислений и видеокарты.

Архитектура системы обработки изображений. Рассматриваемая в данной работе система состоит из двух частей – собственно сервера обработки изображений и web-приложения для доступа к нему. Сервер обеспечивает хранение в памяти и обработку достаточно большого количества изображений с использованием различных вычислительных устройств: основных процессоров сервера и видеокарт. Web-приложение обеспечивает взаимодействие клиента с сервером.

Чтобы выполняться наиболее эффективно, функции ЦОИ должны быть написаны с учетом архитектуры вычислительных ресурсов сервера. Однако в процессе эксплуатации сервера тип и количество его центральных процессоров и плат расширения (видеокарт) может меняться. В этом случае реализация функций ЦОИ должна быть модифицирована, причем, поскольку эти изменения не меняют базовую функциональность сервера, модификации должны осуществляться прозрачно для остальных его частей. Наиболее просто и эффективно этого можно достичь, вынеся реализацию функций ЦОИ в динамически компонуемую библиотеку (далее библиотека). Тогда при изменении конфигурации аппаратного обеспечения сервера для изменения реализации функций ЦОИ достаточно будет заменить лишь файл библиотеки.

Впрочем, в этом случае возникает проблема обеспечения совместимости версий сервера и библиотеки. Причем из-за сложности создания реализации функций ЦОИ может сложиться ситуация, когда более новая версия сервера должна будет работать со старой версией библиотеки и наоборот. То есть, необходимо обеспечить как прямую, так и обратную совместимость. Этого можно достичь, если библиотека будет сообщать серверу о наборе реализуемых ею функций и об их параметрах. При этом каждой функции и каждому параметру функции должны быть присвоены числовые идентификаторы, уникальные на протяжении всего жизненного цикла системы. На основании этих идентификаторов клиенты смогут определять, поддерживает ли сервер нужные им функции с необходимым набором параметров. Также на основании этих данных клиентское приложение сможет внести соответствующие изменения в интерфейс пользователя.

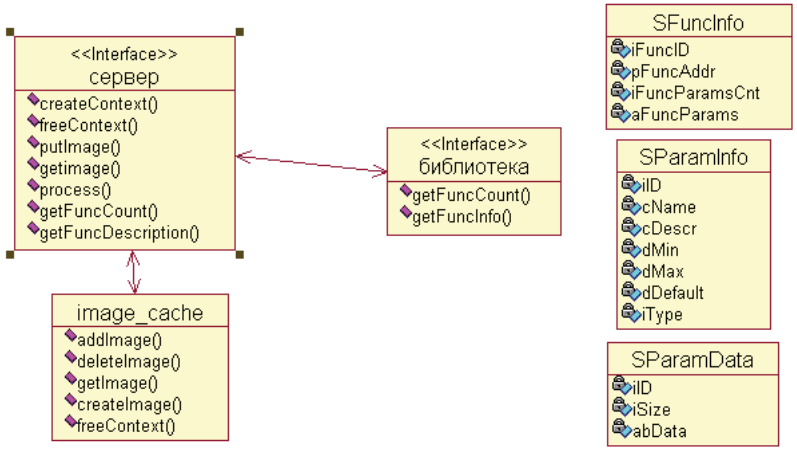
В большинстве практических приложений процесс обработки изображения представляет собой последовательное выполнение набора операций над одним или несколькими изображениями. Чтобы при выполнении каждой

следующей операции не передавать по сети изображение на сервер, последний должен обеспечить хранение исходных и результирующих изображений. Особенно это необходимо при использовании сервера web-приложением, поскольку оно обычно отображает изображения большого размера по частям. Для этой цели на сервере должен быть реализован контейнер с функцией автоматического удаления устаревших изображений (кэш изображений).

Поскольку при использовании видеокарт данные изображений располагаются в памяти видеокарты, о существовании которой известно только библиотеке, то контейнер на первый взгляд должен быть реализован именно в библиотеке функций ЦОИ. Однако такое решение может привести к увеличению вероятности сбоев из-за дублирования функционала контейнера в большом количестве библиотек для разной архитектуры. Поэтому кэш изображений должен быть реализован на сервере, а в библиотеках, использующих локальную память видеокарт, может быть реализован второй уровень кэширования, прозрачный для сервера.

В зависимости от архитектуры аппаратуры сервера, предполагаемой ресурсоемкости запроса и интенсивности потока запросов сервер может выполнять запросы последовательно или параллельно. Поскольку все данные об архитектуре сервера содержаться в библиотеке, то и распределение запросов по вычислительным узлам сервера и их сериализацию, если таковая необходима, должна выполнять библиотека.

С учетом всего вышеизложенных требований, программное обеспечение сервера приобретает архитектуру, представленную на рисунке.



Архитектура сервера обработки изображений

Как и во всех клиент-серверных системах, поддерживающих сеансы [3], клиент перед началом использования функций сервера должен создать

логическое соединение – контекст, путем вызова функции ***createContext***. Функция возвращает идентификатор контекста, который далее используется для разделения доступа к изображениям различных клиентов и для освобождения занятых ресурсов при неожиданном отключении клиента. По завершению работы с сервером клиент (или транспортный уровень сервера при таймауте соединения с клиентом) вызывает функцию ***freeContext***, которая освобождает все связанные с соединением ресурсы.

Хранение изображений на сервере обеспечивает контейнер ***image_cache***, реализующий модель доступа к элементам по ключу. Ключом является идентификатор изображения, назначаемый ему контейнером при размещении или создании изображения. Клиент может размещать изображения на сервере и считывать их с помощью функций ***putImage*** и ***getImage*** соответственно. Для выделения памяти под результирующее изображение используется функция ***createImage***. При размещении изображения на сервере ему автоматически присваивается контекст соединения, в рамках которого оно было создано.

Сервер предоставляет клиенту информацию о реализуемых библиотекой функциях ЦОИ посредством функций ***getFuncCount*** и ***getFuncInfo***, которые вызывают одноименные функции библиотеки ЦОИ. Первая функция возвращает количество реализуемых библиотекой функций ЦОИ, а вторая – информацию о функции с заданным порядковым номером в виде структуры ***SFuncInfo***, в которой содержится идентификатор функции, адрес вызова (только при обмене между сервером и библиотекой) и список параметров функции.

Параметры функции описываются структурой ***SParamInfo***, содержащей идентификатор параметра ***iID***, его название ***cName*** и описание ***cDescr***, граничные значения ***dMin*** и ***dMax***, значение по умолчанию ***dDefault*** и тип параметра ***iType***. Значения полей идентификатора и типа параметра используются при формировании списка параметров для вызова функции ЦОИ, остальные поля используются только для поддержки интерфейса пользователя. На данный момент параметр может быть либо числом, либо изображением. Значение числовых параметров задается числом с плавающей точкой двойной точности. Для изображений в качестве значения параметра выступает числовой идентификатор изображения в контейнере.

В процессе осуществления вызова функции ЦОИ клиент передает серверу идентификатор вызываемой функции и список ее параметров в виде массива структур ***SParamData***. Сервер проверяет допустимость параметров функции, получает из библиотеки адрес вызова функции по ее идентификатору и выполняет вызов.

Все функции ЦОИ имеют следующий прототип:

int process(image_cache *c, int iParamCnt, SParamData *pParams).

Возвращаемое значение отражает результат выполнения функции. Ненулевое возвращаемое значение свидетельствует о том, что при

выполнении функции произошла ошибка и является кодом этой ошибки. Нулевое – о том, что функция выполнена успешно.

Применение. Предложенная выше архитектура была применена при создании web-приложения для создания и обработки изображений с расширенным динамическим диапазоном (HDRI). Соответствующие алгоритмы достаточно ресурсоемки и неплохо реализуются на вычислительных устройствах SIMD-архитектуры. Процесс работы с HDRI включает в себя их создание из нескольких фотоизображений, сделанных с различным значением экспозиции, обработку и тональную компрессию – процесс преобразования HDRI изображения в изображение с обычным динамическим диапазоном.

Для создания web-приложения был использован язык PHP. Сервер был реализован в виде динамически компонуемой библиотеки расширения PHP машины [4]. Сервер инициализируется при запуске PHP-машины и предоставляет PHP-программе набор функций для работы с контекстом соединения и чтения/записи изображений. Функция вызова операции ЦОИ сервером не экспортируется, а вместо этого экспортируются конкретные функции создания HDR изображения и его тональной компрессии. Также сервер предоставляет приложению возможность получить информацию о параметрах данных функций.

Алгоритмы создания HDR изображения [5] и тональной компрессии [6] были реализованы в двух версиях библиотек функций ЦОИ, использующих вычислительные ресурсы основных процессоров и процессора видеокарты. По трем фотографиям размером 3000*2000 точек было сформировано HDR изображение и выполнена его тональная компрессия с использованием обоих библиотек. При использовании ресурсов процессора время обработки составило 115 секунд, а при использовании ресурсов видеокарты – 7 секунд, то есть скорость обработки изображений при переходе на использование видеокарт увеличилась в 16,5 раз. При таком ускорении обработки затраты времени на передачу изображения по сети будут скомпенсированы. А данная архитектура может найти свое применение при построении систем ресурсоемкой обработки изображений.

Список литературы: ***1.*** *Немнюгин С., Стесик О.* Параллельное программирование для многопроцессорных вычислительных систем. – СПб.:БХВ-Петербург. – 400 с. – 2002. ***2.*** *Mark W.* Future graphics architecture // ACM Queue. – 2008. – Vol. 6, № 2. – p. 54-64. ***3.*** *Танненбаум Э., ванн Стеен М.* Распределенные системы: принципы и парадигмы. – СПБ.: Питер, 2003. – 880 с.. ***4.*** Шлосснейгл Дж. Профессиональное программирование на PHP. – М.: Вильямс, 2006. – 624 с. ***5.*** Debevec P., Jitendra M. Recovering High Dynamic Range Radiance Maps From Photographs // Proceedings of SIGGRAPH 1997. – pp. 130-135. ***6.*** Biswas K.K., Pattanaik S. A Simple Spatial Tone Mapping Operator for High Dynamic Range Images // Proceedings of 13th Color Imaging Conference, 2005. – P. 291-296.

Поступила в редколлегию 26.02.09