Особенности построения архитектуры масштабируемой графической системы стандарта OpenGL на основе ЦПОС

А.Н. Милов
Научно-производственный центр «Электронные Вычислительно-Информационные Системы»,
Москва, Россия
quattro@elvees.com

Аннотация

Рассматривается вопрос организации интерфейса OpenGL для построения масштабируемых графических архитектур на базе цифровых процессоров обработки сигналов (ЦПОС) и методах организации вычислений в таких архитектурах. Приводятся результаты реализации двухпроцессорной графической системы с внешним интерфейсом OpenGL на базе систем на кристалле серии «Мультикор».

Ключевые слова: *OpenGL, масштабируемая графическая система, цифровые процессоры.*

1. ВВЕДЕНИЕ

Проблема визуализации трёхмерной графики для бортовых систем решена на недостаточном уровне, т.к. используемые в бортовых условиях вычислительные устройства являются недостаточно производительными, а большинство из них изначально не предназначено для вычислений такого уровня.

С другой стороны, в области компьютерной графики в последние годы прослеживается тенденция к переходу от аппаратной реализации вычислений к программируемым и масштабируемым архитектурам.

ЦПОС обладают достаточной производительностью и используются в составе различных бортовых систем, но имеют ряд ограничений. К тому же эти процессоры являются программируемыми – построенная на их базе система является гибкой и легко модифицируемой.

НПЦ «ЭЛВИС» является разработчиком собственной IP-ядерной платформы проектирования систем на кристалле «Мультикор» [1]. Она включает в себя широкий набор программируемых ядер и других аппаратных средств: RISC-ядро (MIPS32 – совместимое), высокопроизводительные 32-битные DSP-ядра, осуществляющие обработку в различных форматах с фиксированной и плавающей точкой, поддерживающие параллелизм уровня инструкций (ILP) посредством VLIW-подобных инструкций и допускающие масштабирование производительности по SIMD-типу [2], контроллеры PCI, контроллер внешней памяти, DMA каналы прямого доступа к памяти и пр.

Была поставлена задача адаптации типичных вычислений компьютерной графики к особенностям архитектуры ЦПОС с универсальной системой команд.

Рассматривается графическая система, поддерживающая следующую основную функциональность OpenGL:

- Задание модельно-видовой и проекционной матриц трансформаций модели;
- Задание области вывода кадрового буфера;
- Описание трехмерных моделей посредством всех поддерживаемых OpenGL примитивов (включая задание нормалей к вершинам, цвета или текстурных координат);
- Задание и масштабирование именованных текстур с размерами $2^n, n \in N, n \le 7$ (до 128x128 текселей);
- Отбраковка нелицевых граней примитивов (culling);
- Создание списков вызова (display lists);
- Задание одного источника света и интенсивности общего освещения (освещение по методу Гуро);
- Работа непосредственно с кадровым буфером.

Можно определить основные направления решения задачи:

- адаптация вычислений к блочному режиму обработки, принятому в устройствах DSP;
- масштабирование производительности системы по SIMD-типу и наращиванием числа процессоров в системе;
- организация общего интерфейса OpenGL системы.

В качестве целевой версии OpenGL была принята версия 1.4. Заметим, что реализация OpenGL 2.0 также возможна (поскольку система является программируемой), однако для этого требуется написание оптимизирующего компилятора графических программ для процессорного ядра, использующего ILP и работающего по конвейерному принципу, что само по себе является актуальной задачей современных научных изысканий.

2. СТАНДАРТ OPENGL

OpenGL (Open Graphics Library) – это открытый и мобильный стандарт для описания двух- и трёхмерных сцен с использованием унифицированных команд [3]. Стандарт предусматривает архитектуру взаимодействия «клиент-сервер» при организации такого взаимодействия. Это означает, что клиент формирует вызовы OpenGL и посылает их на сервер для обработки и формирования окончательного изображения, при этом клиент и сервер OpenGL могут быть даже разными вычислительными устройствами, соединенными например, через локальную сеть.

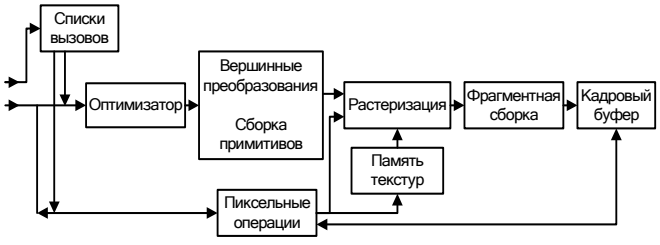

Рис. 1. Операции конвейера OpenGL

Основные операции конвейера OpenGL приведены на рис. 1.

Приведенный конвейер является скорее логическим устройством архитектур OpenGL, чем иллюстрацией их аппаратного устройства, поскольку оно уникально для каждого графического ускорителя ввиду постоянного совершенствования аппаратуры и алгоритмов. Поэтому при построении архитектур OpenGL широко распространено понятие *контекста OpenGL*.

Контекст OpenGL – это упорядоченный набор данных, формируемый и изменяемый вызовами OpenGL, который содержит информацию о текущих настройках сцены, созданных клиентом OpenGL, и однозначно определяющий состояние графической системы между вызовами. Он может быть реализован в виде программно доступной структуры данных как в области памяти клиента OpenGL, так и сервера.

3. КЛАССИФИКАЦИЯ ВЫЗОВОВ OPENGL

Предлагаемая классификация вызовов OpenGL предусматривает эффективное разделение контекста между памятью клиента и памятью сервера и направлена на организацию эффективных потоков данных и вычислений в системе при соблюдении положений стандарта OpenGL.

Вызовы OpenGL классифицируются в соответствии с влиянием, которое они оказывают на конвейер и контекст OpenGL. Можно выделить четыре основные группы вызовов: интерфейсные, контекстные, модельные и конвейерные.

Интерфейсные вызовы не оказывают влияние на конвейер OpenGL и производят только чтение контекста. Примеры таких вызовов: glIsTexture, glIsList, glIsEnabled, glGetFloatv…

Контекстные вызовы формируют состояние контекста OpenGL, который в последствии участвует в добавлении в очередь обработки нового буфера входных данных. Примеры: glMatrixMode, glTexImage2D, glViewport, glLightfv, glClear…

Модельные вызовы не влияют на контекст OpenGL и задают вершины, нормали, цвета или координаты текстуры модели. Примеры: glBegin, glEnd, glVertex, glColor, glNormal…

Конвейерные вызовы оказывают влияние только на конвейер обработки данных: производится останов конвейера, затем выполняются соответствующие операции. Примеры: glDrawPixels, glReadPixels, glCopyPixels, glFinish.

Интерфейсные и контекстные вызовы могут быть реализованы исключительно в рамках ресурсов клиента OpenGL, поскольку их исполнение фактически сводится к чтению и записи по соответствующему смещению контекста.

Для модельных и конвейерных вызовов необходимы некоторые комментарии. Известно, что производительность большинства графических систем с классическими конвейерами растеризации в высокой степени коррелирует с числом и конфигурацией входящих примитивов для растеризации. Так, для задания параметров сцены визуализации в большинстве случаев необходимо не более одного соответствующего контекстного вызова OpenGL, однако для формирования реалистичного изображения необходимы, как правило, десятки и сотни тысяч примитивов, на вид которых влияют параметры контекста. Кроме того, контекст может изменяться в процессе визуализации, однако стандарт запрещает его изменение внутри пары вызовов glBegin/glEnd.

В связи с этим предлагается формировать промежуточные блоки для обработки в моменты вызовов glEnd, которые включают в себя:

- Набор контекстной информации, соответствующей блоку данных, который включает матрицы преобразований координат, параметры освещения, отбраковки примитивов, размера области вывода и пр.
- Массивы координат вершин, текстурных координат, цветов и нормалей к вершинам для текущей модели.

Описанные блоки данных содержат всю необходимую информацию для самостоятельной обработки любым из процессоров в системе с последующим формированием структур растеризации.

Собственно, ограничение, которым мы воспользовались, и было введено в стандарт OpenGL для того, чтобы позволить каждой реализации OpenGL эффективно производить буферизацию вершинных и параметрических данных.

Таким образом, предлагаемый подход к организации клиент-серверной архитектуры позволяет формировать из непрерывной последовательности вызовов OpenGL отдельные независимые блоки данных и осуществлять их конвейерную обработку. Вместе с тем, такое разбиение решает еще одну серьезную проблему при концентрации основных вычислений в рамках ресурсов DSP-ядер – относительно малый размер внутренней памяти данных, который обязывает производить блочную обработку потока данных.

Тогда схематичная диаграмма взаимодействия внутренних компонентов OpenGL примет вид (рис. 2). Диспетчер конвейера обработки осуществляет постановку нового блока данных в очередь обработки процессора, который занимается формированием текущего кадра.

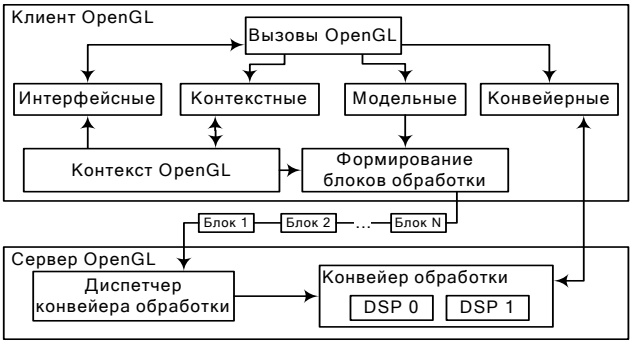

Рис. 2. Взаимодействие внутренних компонентов OpenGL

Конвейерные вызовы OpenGL производят следующую последовательность действий:

- останов добавления в очередь новых блоков данных;
- ожидание окончания обработки последнего переданного кадра сопоставленным ему процессором;
- операции чтения/записи буфера на плате;
- возобновление очереди конвейера.

В связи с тем, что для выполнения конвейерных вызовов требуется останов конвейера, общая производительность системы снижается при использовании таких вызовов.

Впрочем, последнее утверждение справедливо для всех современных графических ускорителей.

Предложенный способ блочной обработки данных эффективен как для систем, масштабируемых по SIMD-типу (можно варьировать размеры входных буферов вершин для обработки), так и для систем, производительность которых масштабируется увеличением числа процессоров в системе.

Рассмотрим более подробно последовательность вычислений в конвейере обработки данных на примере двухпроцессорной системы.

4. ОРГАНИЗАЦИЯ КОНВЕЙЕРА ОБРАБОТКИ ДАННЫХ В ДВУХПРОЦЕССОРНОЙ СИСТЕМЕ

Предложенная классификация вызовов OpenGL позволила существенно сократить набор функций внешнего интерфейса, поддерживаемого непосредственно каждым из процессоров «Мультикор» в системе, а значит упростить реализацию и избежать множественного исполнения общих операций для всех процессоров. Список этих функций и их описание приведены в таблице 1.

Таблица 1. Функциональность сервера OpenGL

Название функции	Описание
ClearBuffer	Очистка соответствующего буфера заданным значением. Применяется для кадрового буфера, буфера глубины и пр.
Render	Производится обработка входного блока данных: вершинные преобразования, сборка примитивов, отбраковка нелицевых граней, отсечение невидимых элементов, расчет освещения. Формируются структуры растеризации (аналитически заданные фрагменты растра) для стадии Rasterize.
Rasterize	Растеризация фрагментов в конечный кадровый буфер в соответствии со сформированными фрагментными данными на стадии Render.
LoadTexture	Загрузка текстуры в локальную память во внутреннем формате, совместимом с форматом кадрового буфера.
LoadFragment	Загрузка в кадровый буфер растрового фрагмента данных во внутреннем формате (glDrawPixels, glBitmap).
SwapBuffers	Передача сформированного растрового изображения на контроллер видео и смена активного процессора.

На транспортном уровне системы каждой приведенной функции соответствует одноименное *сообщение*, включающее в себя заголовок и тело. Заголовок кодирует функцию сообщения и атрибут применения. В теле функции содержатся параметры обработки сообщения, соответствующие его типу. Например, тело сообщения ClearBuffer содержит индекс буфера для очистки (кадровый, Z-буфер и пр.) и цвет очистки.

При анализе различных вариантов организации многопроцессорной системы принимались во внимание несколько аспектов:

1. обработка входных вызовов OpenGL согласно стандарту должна происходить в порядке очереди;
2. необходимо осуществить распараллеливание исходной задачи между процессорами максимально эффективно, чтобы получить наилучший результат производительности;
3. промежуточный обмен данными между процессорами нежелателен ввиду малых темпов передачи данных.

Принципиальная маркированная сеть Петри [4], описывающая работу системы, приведена на рис. 3.

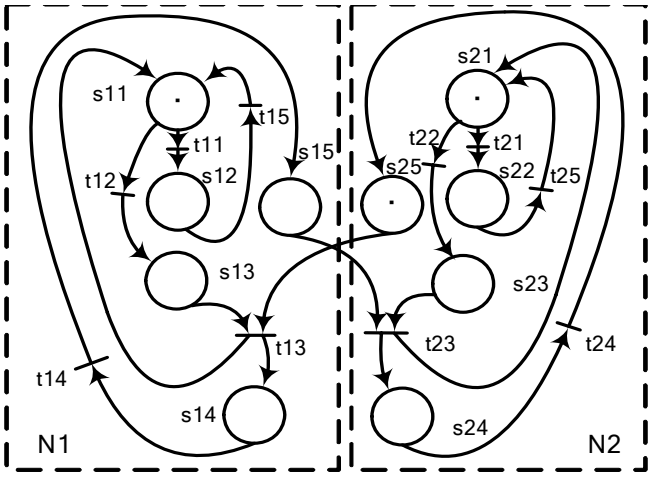

Рис. 3. Сеть Петри работы двухпроцессорной системы.

Условные обозначения: s_{11}, s_{21} - состояние готовности процессора к обработке входящего сообщения; s_{12}, s_{22} - состояние работы функций Render, LoadTexture, LoadFragment или ClearBuffer; s_{13}, s_{23} - работа функции Rasterize; s_{14}, s_{24} - работа функции SwapBuffers; s_{15}, s_{25} - состояния доступности ресурса кадрового буфера после выгрузки соответственно первым и вторым процессором; t_{13}, t_{23} - переходы получения права работы с кадровым буфером. Подсети N1 и N2 соответствуют двум процессорам в системе.

В представленной системе работают два процессора, имеющие один общий ресурс – выходной кадровый буфер. При помощи маркера-семафора право работы с ним переходит от одного процессора к другому. Начальная маркировка соответствует состояниям готовности обоих процессоров (s_{11}, s_{21}) и принадлежности маркера-семафора первому процессору (s_{25}). Промежуточный обмен данными между процессорами отсутствует, т.к. они обрабатывают разные кадры. Осуществляется параллельная конвейерная обработка соседних кадров в очереди. В каждый момент времени сформированные входные буферы передаются для обработки только одному из процессоров. Назовем его *активным*. Тогда действие функции SwapBuffers в такой системе сводится к смене активного процессора. При ее

вызове, однако, не прекращается обработка данных в процессоре, который был активным на момент вызова.

5. РЕЗУЛЬТАТЫ И ВЫВОДЫ

Представленный подход был реализован в рамках создания двухпроцессорной графической системы с единым внешним интерфейсом OpenGL на базе микросхем 1892ВМ3Т (рис. 4). Основное назначение системы – встраиваемые (бортовые) системы визуализации.

Рис. 4. Двухпроцессорная графическая система.

На рис. 5 приведен прирост производительности двухпроцессорной системы относительно однопроцессорной. В качестве тестовых было выбрано несколько сцен с варьируемым числом треугольников. Режим работы графической библиотеки – VGA(640x480), 32bpp.

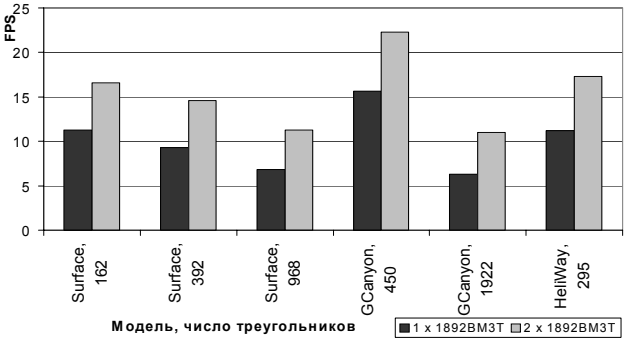

Рис. 5. Прирост производительности от масштабирования.

Как видно из рис. 5, производительность двухпроцессорной системы возрастает с увеличением вычислительной нагрузки на процессоры (а именно, с увеличением числа треугольников в сцене). И это вполне логично, т.к. при малом числе треугольников основная нагрузка лежит на транспорте кадрового растра на выход системы.

Аппаратные характеристики графической системы приведены в таблице 2.

Таблица 2. Характеристики графической системы.

Характеристика	Значение
Глубина цвета	16, 32bpp
Разрешения	QQVGA(120x160)-XGA+(1400x1050)
Z-буфер	32 битный
Параметры процессоров	80 МГц, 0.25 мкм, 300 MFLOPs, SISD (RISC+DSP)

В настоящее время рассматривается перспектива перехода на процессор 1892ВМ5Я (ЦПОС-02), обладающий следующими характеристиками: 120МГц, 0.25 мкм, 1440 MFLOPs, ICACHE, PCI, MIMD (RISC + 2 x DSP). Прогнозируемый прирост производительности составит порядка 300-400%.

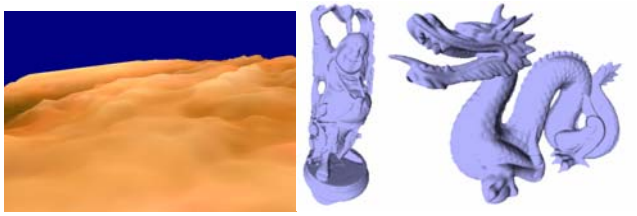

Рис. 6. Некоторые результаты работы графической системы (модели Happy Buddah и Dragon заимствованы из [5]).

Дополнительную информацию о параметрах упомянутой графической системы можно найти в [6].

6. БИБЛИОГРАФИЯ

[1] «*Микросхема интегральная 1892ВМ3Т. Руководство пользователя*», ГУП НПЦ «ЭЛВИС», Москва, 2005.

[2] «*DSP-ядро ELcore-x4. Система инструкций*», ГУП НПЦ «ЭЛВИС», Москва, 2006.

[3] M. Segal, K. Akeley, «*The OpenGL® Graphics System: A Specification (Version 1.4)*», Silicon Graphics Inc., 2002.

[4] Котов В.Е. «*Сети Петри*», М. Наука, 1984.

[5] The Stanford 3D Scanning Repository, http://graphics.stanford.edu/data/3Dscanrep/

[6] Качоровский Д.А., Милов А.Н., *«Бортовой модуль графического ускорителя стандарта OpenGL на базе ЦПОС серии «Мультикор»*, Микроэлектроника и информатика – 2007. 14-я Всероссийская межвузовская научно-техническая конференция студентов и аспирантов: Тезисы докладов. - М.: МИЭТ, 2007.

Об авторе

Алексей Николаевич Милов – аспирант ППИ «Научный центр», разработчик графических систем НПЦ «ЭЛВИС». E-mail: quattro@elvees.com

On building of scalable OpenGL architecture based on DSP

Abstract

The paper is devoted to some questions of implementing of scalable OpenGL architecture based on digital signal processors. A new method of graphical data stream processing (including shared OpenGL interface) is discussed. Some implementation results are also quoted.

Keywords: *OpenGL, scalable graphical architecture, digital signal processors*.

About the author

Alexey Milov is a Ph.D. student at PE «Science Centre». His contact email is quattro@elvees.com.