РОССИЙСКАЯ ФЕДЕРАЦИЯ

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

(19) **RU** (11) **2 448 369** (13) **C2**

(51) МПК
G06T 1/60 (2006.01)

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ПАТЕНТУ

(21)(22) Заявка: **2010107218/08, 30.07.2008**

(24) Дата начала отсчета срока действия патента:
30.07.2008

Приоритет(ы):
(30) Конвенционный приоритет:
30.07.2007 US 11/830,667

(43) Дата публикации заявки: **10.09.2011** Бюл. № 25

(45) Опубликовано: **20.04.2012** Бюл. № 11

(56) Список документов, цитированных в отчете о поиске: **WO 96/01024 A1, 11.01.1996. RU 2249245 C2, 27.03.2005. US 5881263 A, 09.03.1999. WO 03/005307 A1, 19.01.2003.**

(85) Дата начала рассмотрения заявки PCT на национальной фазе: **01.03.2010**

(86) Заявка PCT:
US 2008/071655 (30.07.2008)

(87) Публикация заявки PCT:
WO 2009/018385 (05.02.2009)

Адрес для переписки:
129090, Москва, ул. Б. Спасская, 25, стр.3, ООО "Юридическая фирма Городисский и Партнеры", пат.пов. Ю.Д.Кузнецову

(72) Автор(ы):
ЦЗЯО Гофан (US),
БОРД Алексей В. (US),
ЮЙ Чунь (US),
ЧЭНЬ Линцзюнь (US),
ДУ Юнь (US)

(73) Патентообладатель(и):
КВЭЛКОММ ИНКОРПОРЕЙТЕД (US)

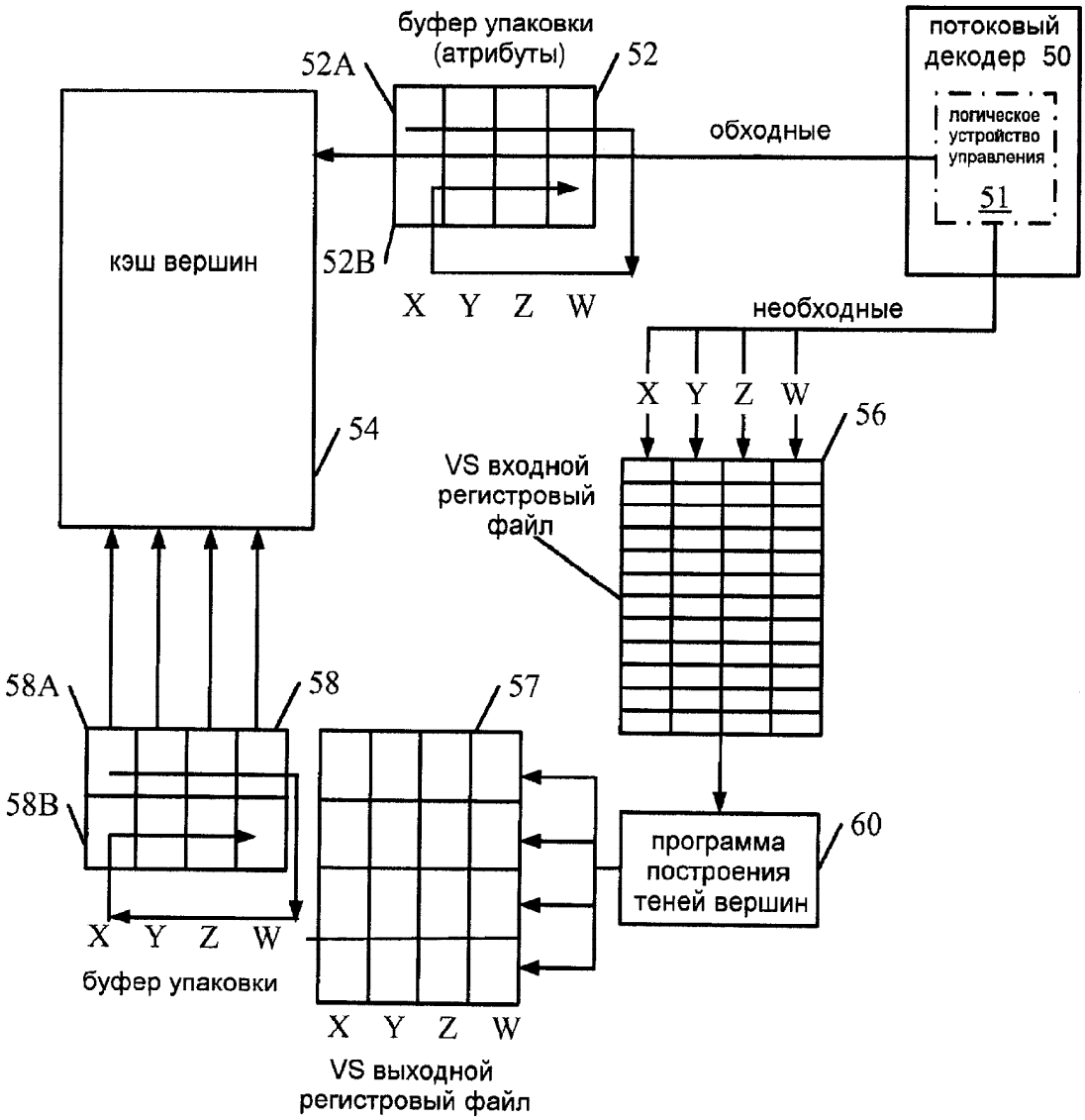
(54) СХЕМА ДЛЯ УПАКОВКИ И СВЯЗЫВАНИЯ ПЕРЕМЕННОЙ В ГРАФИЧЕСКИХ СИСТЕМАХ

(57) Реферат:

Изобретение относится к способам для упаковки и связывания переменной в графических системах. Техническим результатом является расширение функциональных возможностей за счет уменьшения полосы частот трафика, экономии энергии и улучшении производительности. Беспроводное устройство выполняет процесс упаковки компилятором первого уровня и процесс упаковки аппаратным обеспечением второго уровня над переменными. Процесс упаковки компилятором упаковывает две или несколько переменных величин (переменных или атрибутов) программы построения теней, сумма компонентов которых равна M, в совместно используемый M-мерный (MD) векторный регистр. Упаковка аппаратным обеспечением последовательно упаковывает M компонентов переменных величин (переменных или атрибутов) программы построения теней и какие-либо остающиеся переменные величины в кэш вершин или другой носитель данных. 3 н. и 14 з.п. ф-лы, 21 ил., 7 табл.

RU 2448369 C2

RU 2448369 C2



ФИГ. 4

RU 2448369 C2

RU 2448369 C2

RUSSIAN FEDERATION

FEDERAL SERVICE
FOR INTELLECTUAL PROPERTY

(19) **RU** (11) **2 448 369** (13) **C2**

(51) Int. Cl.
G06T 1/60 (2006.01)

(12) ABSTRACT OF INVENTION

(21)(22) Application: **2010107218/08, 30.07.2008**

(24) Effective date for property rights:
30.07.2008

Priority:
(30) Priority:
30.07.2007 US 11/830,667

(43) Application published: **10.09.2011** Bull. 25

(45) Date of publication: **20.04.2012 Bull. 11**

(85) Commencement of national phase: **01.03.2010**

(86) PCT application:
US 2008/071655 (30.07.2008)

(87) PCT publication:
WO 2009/018385 (05.02.2009)

Mail address:
129090, Moskva, ul. B. Spasskaja, 25, str.3, OOO "Juridicheskaja firma Gorodisskij i Partnery", pat.pov. Ju.D.Kuznetsovu

(72) Inventor(s):
TsZJaO Gofan (US),
BORD Aleksej V. (US),
JuJ Chun' (US),
ChEhN' Lintszjun' (US),
DU Jun' (US)

(73) Proprietor(s):
KVEhLKOMM INKORPOREJTED (US)

RU 2448369 C2

RU 2448369 C2

(54) SCHEME FOR VARIABLE PACKING AND BINDING IN GRAPHICS SYSTEMS

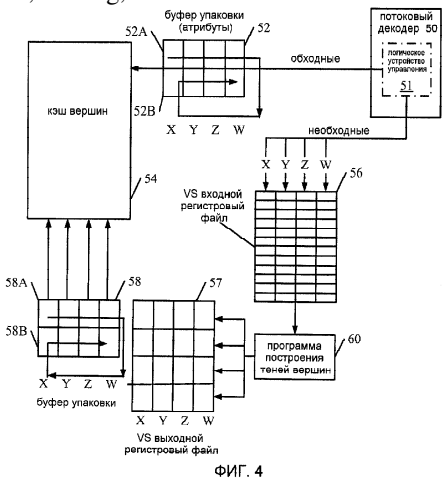
(57) Abstract:

FIELD: information technology.

SUBSTANCE: wireless device performs a first-level compiler packing process and a second-level hardware packing process over variables. The compiler packing process packs two or more shader variables (variables or attributes) whose sum of components equals M into a shared M-dimensional (MD) vector register. The hardware packing consecutively packs M components of the shader variables (variables or attributes) and any remaining variables into a vertex cache or other storage medium.

EFFECT: broader functional capabilities owing to reduced traffic bandwidth, saving energy and higher efficiency.

17 cl, 21 dwg, 7 tbl



ФИГ. 4

УРОВЕНЬ ТЕХНИКИ

I. Область техники

Данное описание относится, в общем, к области обработки графики и, более конкретно, к способам для упаковки и связывания переменной в графических системах.

II. Уровень техники

Общедоступный графический стандарт OpenGL или OpenGL ES имеет определенную функциональность, которая может быть сменена во время операций на вершинах и пикселах с использованием программ построения теней вершин и фрагментов. Программы построения теней вершин и фрагментов были разработаны для визуализации специальных эффектов, которые не были достигнуты стандартной функциональностью OpenGL.

Теперь со ссылкой на фиг. 1 показана общая блок-схема стандартных стадий конвейерной обработки в устройстве обработки графики (GPU) с программами построения теней. Имеется три основных стадии конвейера: программа построения теней вершин, обозначенная в блоке S10, ассемблер примитивов и растеризатор, обозначенные в блоке S12, и программа построения теней фрагментов, обозначенная в блоке S14. Дополнительный блок S16 обеспечен для операций на выборках.

Программа построения теней вершин (VS) S10 является программой или компьютерным программным продуктом, исполняемым для каждой вершины геометрического объекта. Входные данные VS S10 называются атрибутами, обозначенными в блоке A2. VS S10 также принимает в качестве входных данных постоянные (uniform) вершин VU2, которые могут включать в себя некоторое число постоянных вершин 0-95 (т.е. 96 постоянных вершин). Выходные данные из VS S10 и затем ассемблера примитивов и растеризатора S12 обычно называются переменными (varying), обозначенными в блоке V3, и обычно находятся в кэше вершин (хранящем выходные данные VS) или другом носителе данных (хранящем выходные данные растеризатора). Переменными V3 могут быть значения, ассоциированные с пикселами треугольников некоторого геометрического объекта. Этими значениями, ассоциированными с пикселами, являются результаты ассемблера примитивов и растеризатора S12, вычисленные на основе VS результатов, ассоциированных с вершинами треугольников некоторого геометрического объекта. Эти VS результаты, ассоциированные с этими вершинами, и переменные V3, ассоциированные с пикселами, имеют одни и те же имена или идентификаторы, типы и упорядочение. Переменные V3, ассоциированными с пикселами, являются входными данными для программы построения теней фрагментов (FS) S14. FS S14 также принимает в качестве входных данных постоянные фрагментов FU3, которые обычно включают в себя некоторое количество (например, 16) постоянных фрагментов.

Фиг. 2 показывает общую блок-схему стандартных стадий конвейерной обработки с программами построения теней. Для VS S10 внутри устройства обработки графики (GPU), обычно имеется восемь (8) регистров атрибутов RA2 для хранения атрибутов 0-7. Обычно имеется восемь выходных регистров RV3A переменной для хранения переменных 0-7. Регистры RV3A переменной сохраняют выходные данные VS, которые обычно являются кэшем вершин. Обычно имеется восемь выходных регистров RV3B переменной для хранения переменных 0-7. Регистры RV3B переменной сохраняют результаты растеризатора, соответствующие переменным, ассоциированным с пикселами. Регистры RA2 атрибутов и регистры RV3A переменной являются входными регистрами, индексированными атрибутами 0-7, и выходными

регистрами RV3 переменной, индексированными переменными 0-7, соответственно. Эти идентификаторы регистров назначаются компилятором, который компилирует программу построения теней вершин и фрагментов с языка высокого уровня в машинный язык. Эти регистры, используемые в программе построения теней на языке высокого уровня, называются посредством имен вместо идентификаторов/индексов. Эти имена регистров видны только для разработчиков приложений. Приложения получают доступ к регистрам через имена регистров. Идентификаторы регистров видны только для VS S10 или FS S14 в аппаратном обеспечении (HW) GPU. Следовательно, компилятором будет создана таблица символов, такая как таблица входных символов VS, таблица выходных символов и таблица входных символов FS. Однако входные данные VS или таблица входных символов не имеют отношения к выходным данным или таблицей выходных символов в терминах содержания, идентификаторов или имен.

Выходные данные VS или таблица выходных символов должны совпадать с входными данными или таблицей входных символов FS S14 в терминах содержания и имен, хотя входные данные или таблица входных символов FS S14 могут быть некоторым поднабором выходных данных или таблицы выходных символов VS S10.

VS S10 также принимает в качестве входных данных постоянные вершин VU2, хранимые в носителе данных, также как и текстуры, обозначенные как T2, и временные переменные, обозначенные как TV2. Ассемблер примитивов и растеризатор S12 принимают переменные в выходные регистры RV3A переменной, индексированные переменными 0-7 и параметром gl_Position P. Ассемблер примитивов и растеризатор S12 выводят переменные в выходные регистры RV3B переменной, индексированные переменными 0-7 и параметром gl_Position P. FS S14 принимает в качестве входных данных постоянные фрагментов FU3, хранимые в носителе данных, также как и текстуры, обозначенные как T3, и временные переменные, обозначенные как TV3. FS S14 принимает переменные в выходные регистры RV3B переменной, индексированные переменными 0-7 и параметром gl_Position, обозначенным как P. FS S14 также принимает дополнительные параметры gl_Frontfacing, обозначенный как FF, и gl_PointPosition, обозначенный как PP. FS S14 выводит gl_FragColor FC. Атрибуты и переменные также называются переменными программы построения теней.

СУЩНОСТЬ ИЗОБРЕТЕНИЯ

Здесь описаны способы для упаковки и связывания переменной в графических конвейерах. Упаковка переменных программы построения теней является выгодной в мобильном GPU, так что носитель или память используется более эффективно. Упаковка переменных программы построения теней может также уменьшить полосу частот трафика, сэкономить энергию и улучшить производительность.

В одной конфигурации, некоторое устройство имеет носитель данных, имеющий множество совместно используемых M-мерных (MD) регистров. Это устройство также включает в себя устройство обработки для реализации некоторого набора операций для упаковки в каждом совместно используемом MD регистре одной или нескольких переменных программы построения теней, сумма компонентов которых равна M.

В другой конфигурации, интегральная схема содержит носитель данных, имеющий множество совместно используемых M-мерных (MD) регистров. Эта интегральная схема также включает в себя устройство обработки для реализации множества операций для упаковки в каждом совместно используемом MD регистре одной или нескольких переменных программы построения теней, сумма компонентов которых равна M.

Еще одна конфигурация включает в себя компьютерный программный продукт. Этот компьютерный программный продукт включает в себя считываемый компьютером носитель, имеющий команды, чтобы заставить компьютер упаковывать одну или нескольких переменных из набора переменных программы построения теней, сумма компонентов которых равна M, в каждом совместно используемом M-мерном (MD) векторном регистре из множества совместно используемых MD векторных регистров.

Еще одна конфигурация включает в себя процессор, содержащий носитель данных, имеющий множество совместно используемых M-мерных (MD) регистров. Этот процессор также включает в себя интегральную схему для реализации набора операций для упаковки в каждом совместно используемом MD регистре одной или нескольких переменных программы построения теней, сумма компонентов которых равна M.

Дополнительные аспекты станут более ясными из подробного описания, особенно взятого вместе с прилагаемыми чертежами.

КРАТКОЕ ОПИСАНИЕ ЧЕРТЕЖЕЙ

Аспекты и конфигурации данного описания станут более ясными из подробного описания, изложенного ниже, взятого в сопряжении с чертежами, в которых подобные ссылочные символы везде идентифицируют соответствующее.

Фиг. 1 показывает общую блок-схему стандартных стадий конвейерной обработки в устройстве обработки графики с программами построения теней.

Фиг. 2 показывает общую блок-схему стандартных стадий конвейерной обработки с программами построения теней.

Фиг. 3 показывает блок-схему беспроводного устройства.

Фиг. 4 показывает общую блок-схему устройства обработки графики (GPU) для программы построения теней вершин и операций упаковки.

Фиг. 5 показывает общую блок-схему устройства обработки графики (GPU) с операциями программы построения теней фрагментов и упаковки.

Фиг. 6 показывает общую блок-схему драйвера.

Фиг. 7 показывает общую блок-схему двухуровневого процесса упаковки переменных программы построения теней.

Фиг. 8A и 8B показывают программу построения теней вершин перед тем и после того, как обходные атрибуты удалены.

Фиг. 9A и 9B показывают другую программу построения теней вершин перед тем и после того, как обходные атрибуты удалены.

Фиг. 10A и 10B показывают еще одну программу построения теней вершин перед тем и после того, как обходные атрибуты удалены.

Фиг. 11A и 11B показывают еще одну программу построения теней вершин перед тем и после того, как обходные атрибуты удалены.

Фиг. 12A и 12B показывают еще одну программу построения теней вершин перед тем и после того, как обходные атрибуты удалены.

Фиг. 13A-13C показывают общую блок-схему процесса упаковки переменных программы построения теней, скомбинированного с обходом атрибутов.

Фиг. 14 показывает общую блок-схему процесса связывания.

Изображения на чертежах упрощены в иллюстративных целях и не показаны в масштабе. Для облегчения понимания, идентичные ссылочные позиции были использованы, где это возможно, для обозначения идентичных элементов, что справедливо для фигур, за исключением того, что могут быть добавлены суффиксы,

когда это уместно, для дифференциации таких элементов.

Прилагаемые чертежи иллюстрируют примерные конфигурации изобретения и, как таковые, не должны рассматриваться как ограничивающие объем изобретения, который может вмещать и другие равно эффективные конфигурации. Предполагается, что особенности или стадии одной конфигурации могут быть выгодно включены в другие конфигурации без дополнительного описания.

В различных конфигурациях ниже, блоки блок-схем выполняются в изображенном порядке, или же эти блоки или их части могут выполняться одновременно, параллельно или в другом порядке.

ПОДРОБНОЕ ОПИСАНИЕ

Слово «примерный» используется здесь в значении «служащий в качестве примера, или иллюстрации». Любая конфигурация или конструкция, описанная здесь как «примерная», не должна с необходимостью толковаться как предпочтительная или преимущественная перед другими конфигурациями или конструкциями.

Способы, описанные здесь, могут использоваться для беспроводной связи, вычисления, персональной электроники и т.д. Примерное использование этих способов для беспроводной связи описано ниже.

Фиг. 3 показывает блок-схему конфигурации беспроводного устройства 10 для использования в системе беспроводной связи. Этим беспроводным устройством может быть сотовый телефон или фотоаппарат, терминал, телефон-трубка, электронный секретарь (PDA) или некоторое другое устройство. Системой беспроводной связи может быть система множественного доступа с кодовым разделением (каналов) (CDMA), глобальная система мобильной связи (GSM) или некоторая другая система.

Беспроводное устройство 10 способно обеспечить двунаправленную связь через тракт приема и тракт передачи. В тракте приема сигналы, переданные базовыми станциями, принимают антенной 12 и обеспечивают для приемника (RCVR) 14. Приемник 14 обрабатывает с заданными условиями и оцифровывает принятый сигнал и обеспечивает выборки для цифрового блока 20 для дальнейшей обработки. В тракте передачи передатчик (TMTR) 16 принимает данные, подлежащие передаче от цифрового блока 20, обрабатывает с заданными условиями эти данные и генерирует модулированный сигнал, который передают через антенну 12 к базовым станциям.

Цифровой блок 20 включает в себя различные блоки обработки, блоки интерфейса и блоки памяти, такие как, например, модемный процессор 22, видеопроцессор 24, контроллер/процессор 26, процессор 28 дисплея, ARM/DSP 32, блок обработки графики (GPU) 34, внутреннюю память 36 и интерфейс внешней шины (EBI) 38. Модемный процессор 22 выполняет обработку для передачи и приема данных (например, кодирование, модуляцию, демодуляцию и декодирование). Видеопроцессор 24 выполняет обработку на видеоконтенте (например, неподвижных изображений, движущиеся видео и движущиеся тексты) для видеоприложений, таких как видеокамера, воспроизведение видео и организация видеоконференций. Контроллер/процессор 26 может направлять работу различных процессоров и блоков интерфейса в пределах цифрового блока 20. Процессор 28 дисплея выполняет обработку для облегчения показа видео, графики и текстов на дисплее 30. ARM/DSP 32 может выполнять различные типы обработки для беспроводного устройства 10. Блок 34 обработки графики выполняет обработку графики графического конвейера.

Способы, описанные здесь, могут использоваться для любого из процессоров в цифровом блоке 20, например блока 34 обработки графики. Внутренняя память 36 сохраняет данные и/или команды для различных блоков в пределах цифрового

блока 20. EBI 38 облегчает перенос данных между цифровым блоком 20 (например, внутренней памятью 36) и основной памятью 40 по шине или линии передачи данных DL.

Цифровой блок 20 может быть реализован с одним или несколькими DSP (цифровыми сигнальными процессорами), микропроцессорами, RISC (процессор с сокращенным набором команд) и т.д. Цифровой блок 20 может быть также изготовлен на одной или нескольких интегральных схемах прикладной ориентации (ASIC) или некотором другом типе интегральных схем (IC).

Способы, описанные здесь, могут быть реализованы в различных блоках аппаратного обеспечения. Например, эти способы могут быть реализованы в ASIC, DSP, RISC, ARM, устройствах обработки цифровых сигналов (DSPD), программируемых логических устройствах (PLD), программируемых пользователем вентильных матрицах (FPGA), процессорах, контроллерах, микроконтроллерах, микропроцессорах и других электронных блоках.

GPU 34 может быть также совместимым с общедоступным графическим стандартом, таким как OpenGL 2.0, OpenGL ES2.0 или D3d9.0.

Фиг. 4 показывает общую блок-схему графического устройства обработки (GPU) 34 для программы построения теней вершин и операций упаковки. GPU 34 включает в себя потоковый декодер 50, который выдает множество атрибутов в VS входной регистровый файл 56. Эти атрибуты принимаются программой построения теней вершин (VS) 60. Выходные данные VS 60 включают в себя переменные, которые хранят в VS выходном регистровом файле 57. Как можно понять, «регистровый» файл является компонентом аппаратного обеспечения, такого как носитель данных для хранения информации. В этом примере «VS входной регистровый файл» сохраняет входной файл, подлежащий посылке к VS 60. Для простоты, в большинстве случаев, при ссылке на VS входной регистровый файл 56 ссылаются на «входной файл» для VS 60 и/или аппаратное обеспечение для сохранения «входного файла». Подобным же образом, для простоты, в большинстве случаев, при ссылке на VS выходной регистровый файл 57, ссылаются на «выходной файл» из VS 60 и/или аппаратное обеспечение для сохранения «выходного файла». Как будет более подробно описано позже, эти переменные интеллектуально упаковываются компилятором 62 (фиг. 6) для упаковки переменной первого уровня. Переменные в VS выходном регистровом файле 57 посылаются к буферу 58 упаковки непрерывно сериями или в цепной последовательности, который упаковывает переменные в упаковке переменной второго уровня. Когда буфер 58 упаковки заполняется, упакованные переменные затем сохраняют в кэше 54 вершин.

Как будет видно из описания ниже, и VS выходной регистровый файл 57, и VS входной регистровый файл 56 включает в себя множество совместно используемых M-мерных (MD) регистров. Каждый из буферов 58 и 52 упаковки включает в себя по меньшей мере один совместно используемый M-мерный (MD) регистр.

В конфигурации фиг. 4 потоковый декодер 50 генерирует два потока, обходной поток и необходной поток. Необходной поток посылается в VS входной регистровый файл 56 и предпочтительно также упаковывается способом, показанным в Таблице 1. Обходные атрибуты упаковываются в буфере 52 упаковки. Обходные атрибуты будут подробно описаны в связи с фиг. 8A, 8B, 9A, 9B, 10A, 10B, 11A, 11B, 12A и 12B.

Фиг. 5 показывает общую блок-схему устройства обработки графики (GPU) с программой построения теней фрагментов и операциями связывания. Упакованные переменные хранят в кэше 54 вершин. Ассемблер примитивов и растеризатор 90

принимает в качестве входных данных переменные в кэше 54 вершин. Ассемблер примитивов и растеризатор 90 выдают упакованные переменные в буфер 92 переменной. Блок 88 связывания имеет набор команд 82 связывания, которые используются модулем 84 перераспределения и загрузки переменной. Компоновщик 80 на фиг. 6 генерирует таблицу 86 связывания, которая загружается в память для команд 82 связывания на фиг. 5 драйвером 61. Пример таблицы 86 связывания показан в Таблицах 4 и 6, изложенных ниже, который связывает упакованные переменные в таблице выходных символов VS (Таблица 2) с таблицей входных символов FS (Таблица 3). Таблица входных символов FS может иметь меньше символов, чем таблица выходных символов VS. После того как процесс связывания выполнен блоком 88 связывания, переменные из модуля 84 перераспределения и загрузки переменной посылаются в FS входной регистровый файл 79 для использования программой 70 построения теней фрагментов (FS).

Фиг. 6 показывает общую блок-схему драйвера. Драйвер 61 включает в себя компилятор 62 и компоновщик 80. Компилятор 62 генерирует таблицу 64 входных символов VS и таблицу 66 выходных символов VS. Примерная таблица входных символов VS показана ниже в Таблице 1. Примерная таблица выходных символов VS показана ниже в Таблице 2. Компилятор 62 может назначить одному и тому же символу идентификатор в таблице 66 выходных символов, отличный от идентификатора в таблице 74 входных символов FS, так как компилятор 62 может компилировать программу 60 построения теней вершин и программу 70 построения теней фрагментов независимо. Таким образом, имеется компоновщик 80 для того, чтобы драйвер 61 осуществлял отображение между идентификаторами регистров в таблице 66 выходных символов VS и идентификаторами регистров в таблице 74 входных символов FS посредством поиска одного и того же символа в обеих таблицах. Компоновщик 80 осуществляет связь с GPU 34 для загрузки некоторой переменной (соответствующего некоторому местоположению в кэше 54 вершин или буфере 92 переменной) в соответствующий входной регистр во входном регистровом файле 79 программы 70 построения теней фрагментов для одного и того же символа переменной.

Драйвер 61 является драйвером программного обеспечения, имеющим набор команд. Компилятор 62 и компоновщик 80 являются частями драйвера 61 программного обеспечения, запущенного на CPU 32 или контроллере/процессоре 26, тогда как GPU 34 является специальным сопроцессором, инструктируемым драйвером 61.

Таблица входных символов VS, показанная в Таблице 1, включает в себя следующие позиции: имя атрибута, тип, первоначально назначенный идентификатор входного регистра атрибута, первоначальная маска, вновь назначенный идентификатор входного регистра атрибута и новая маска. Таблица выходных символов VS, показанная в Таблице 2, включает в себя следующие позиции: имя переменной, тип, первоначально назначенный идентификатор выходного регистра переменной, первоначальная маска, вновь назначенный идентификатор выходного регистра переменной и новая маска. Маска в таблицах представляет собой достоверные компоненты для векторов атрибутов или векторов переменной, соответствующие памяти регистра стандартного MD (M=4) вектора, расположенной в аппаратном обеспечении (HW) GPU 34. Как первоначально назначенные идентификаторы и маска, так и вновь назначенные идентификаторы и маска помещены вместе в таблицах ниже просто для иллюстрации. Фактически,

первоначально назначенные идентификаторы и маска могут быть временным результатом и станут вновь назначенными идентификаторами и маской посредством использования одной и той же ячейки памяти во время операций.

Компилятор 62 генерирует таблицу 74 входных символов FS и выходные данные 76 FS, обозначенные как gl_FragColor FC (фиг. 2). Таблица 74 входных символов FS, показанная в Таблице 3, включает в себя следующие позиции: имя переменной, тип, первоначально назначенный идентификатор входного регистра переменной, первоначальная маска, вновь назначенный идентификатор входного регистра переменной и новая маска.

В Таблицах 1 и 2 ниже последние два столбца заново обновляются согласно процессу упаковки, описанному позже.

Таблица 1
Таблица входных символов VS

Имя атрибута	Тип	Первоначально назначенный идентификатор входного регистра атрибута	Первона-чальная маска	Вновь назначенный идентификатор входного регистра атрибута	Новая Маска
Позиция0	Плавающий вектор4	0	1111	0	1111
Позиция1	Плавающий вектор3	1	0111	1	0111
Вес	Плавающий	5	0001	2	1000
Нормаль	Короткий плавающий вектор3	2	0111	2	0111
TexCoord0	Плавающий вектор2	3	0011	3	0011
TexCoord1	Плавающий вектор2	4	0011	3	1100

Таблица 2
Таблица выходных символов VS

Имя изменения	Тип	Первоначально назначенный идентификатор выходного регистра переменной	Первона-чальная маска	Вновь назначенный идентификатор выходного регистра переменной	Новая Маска
Позиция	Плавающий вектор4	1	1111	1	1111
Цвет0	Плавающий Вектор4	2	1111	2	1111
Цвет1	Плавающий Вектор3	3	0111	3	0111
TexCoord0	Плавающий вектор2	0	0011	0	0011
TexCoord1	Плавающий вектор2	5	0011	0	1100
TexCoord2	Плавающий Вектор3	4	0111	4	0111

Таблица 3
Таблица входных символов FS

Имя изменения	Тип	Первоначально назначенный идентификатор входного регистра переменной	Первона-чальная маска	Вновь назначенный идентификатор входного регистра переменной	Новая Маска

Цвет0	Плавающий Вектор4	0	1111	2	1111
Цвет1	Плавающий Вектор3	1	0111	0	0111
TexCoord0	Плавающий вектор2	3	0011	1	0011
TexCoord1	Плавающий вектор2	2	0011	1	1100

Таблица 4
Таблица связывания для выходов VS и входов FS

Имя изменения	Первона-чально назначенный идентификатор выходно-го регистра перемен-ной VS	Первона-чально назначен-ный идентификатор входного регистра переменной FS	Перво-началь-ная маска	Вновь назна-ченный идентификатор выход-ного регистра перемен-ной VS	Вновь назначенный идентификатор входного регистра переменной FS	Новая маска
Позиция	1			1		
Цвет0	2	2	1111	2	2	1111
Цвет1	3	3	0111	3	0	0111
TexCoord0	0	1	0011	0	1	0011
TexCoord1	5	1	1100	0	1	1100
TexCoord2	4			4		

Переменные могут быть плавающими (с плавающей точкой), двумерными (2D) векторами, трехмерными (3D) векторами, четырехмерными (4D) векторами, массивом и 2D/3D/4D матрицей и т.д. Спецификация языка построения теней OpenGL ES требует по меньшей мере 32 компонента переменной, подлежащих поддержке в мобильном GPU 34. Каждая переменная имеет различный размер и обычно занимает свое собственное пространство регистра/буфера. В кэше 54 вершин регистр обычно является 4D вектором. Кроме того, регистры, соответствующие VS входному регистровому файлу 56, и регистры, соответствующие VS выходному регистровому файлу 57, обычно являются 4D вектором. Упаковка переменной помещает различные переменные плотно вместе в непрерывном пространстве для каждой вершины или пиксела. Например, упаковка переменной, описанная здесь, помещает два 2D вектора в 4D векторный регистр. В другом примере, упаковка переменной поставит 3D вектор и плавающий (1D) в 4D векторный регистр. Без плотной их упаковки они могут храниться неплотно.

Вышеприведенное описание относится к переменным. Однако, кроме переменных, могут также быть упакованы атрибуты.

Фиг. 7 показывает общую блок-схему двухуровневого процесса 100 упаковки переменных программы построения теней. Процесс 100 начинается в блоке 102, где происходит интеллектуальная упаковка, инструктируемая компилятором 62. В блоке 102 две или более переменных программы построения теней, сумма компонентов которых равна M, назначаются совместно используемому M-мерному (MD) векторному регистру. Для иллюстрации, VS выходной регистровый файл 57 показан со столбцами и строками. Каждая строка имеет четыре (4) блока, обозначенных как X, Y, Z и W. За блоком 102 следует блок 104, где происходит упаковка переменных аппаратного обеспечения в буфер 58 упаковки, который упаковывает переменные программы построения теней в VS выходной регистровый файл 57 последовательно в ряд в блок NxM носителя данных кэша 54 вершин. Переменными программы построения теней фиг. 7 являются переменные.

Как будет видно из описания ниже, обходные атрибуты упаковываются в буфере 52

упаковки подобно процессу блока 104, описанному в связи с фиг. 13A. Необходные атрибуты могут упаковываться с использованием процесса, описанного выше в связи с блоком 102. Таким образом, стадии процесса 100 упаковки могут использоваться для атрибутов. Следовательно, переменные программы построения теней включают в себя переменные или атрибуты.

Первый уровень: упаковка уровня компилятора

Следующее описание интеллектуальной упаковки в блоке 102, которая инструктируется компилятором 62, будет описана в связи с вышеприведенными Таблицами 1 и 2. Интеллектуальная упаковка применяется к переменным программы построения теней (как к переменным, так и к атрибутам). Таблица 1 иллюстрирует упаковку атрибутов, а Таблица 2 иллюстрирует упаковку переменной. Компилятор 62 выполняет упаковку необходных атрибутов или переменной посредством повторного назначения того же самого или общего MD (M-мерного) векторного регистра, имеющего ассоциированный с ним идентификатор регистра, двум или более переменным, сумма компонентов которых равна M (M=4) и обновляет маску, соответственно. MD векторный регистр для атрибутов соответствует памяти для VS входного регистрового файла 56 на фиг. 4. MD векторный регистр для переменных соответствует памяти для VS выходного регистрового файла 57 на фиг. 4. В примерной конфигурации M=4, таким образом, эти векторы обозначены как X, Y, Z и W. Тем не менее, могут использоваться другие конфигурации с большей или меньшей размерностью.

Маска имеет M-битовые местоположения. Таким образом, маска, ассоциированная с каждым переназначенным и/или скомбинированными атрибутами или переменными (переменными программы построения теней) для конкретного MD векторного регистра, используется для обозначения или различения, какая часть совместно используемого MD векторного регистра назначена каждому отдельному атрибуту или переменной (из комбинации) для более позднего повторного обращения и использования.

Например, с конкретной ссылкой на Таблицу 1 выше, texcoord0 и texcoord1 были первоначально назначены различные входные регистры атрибутов, обозначенные номерами идентификаторов 3 и 4 соответственно, в столбце «Первоначально назначенный идентификатор входного регистра атрибута». Кроме того, первоначальными масками для texcoord0 и texcoord1 являются 0011 и 0011 соответственно. Компилятор 62 определяет, что как texcoord0, так и texcoord1 являются 2D векторами, и сумма этих векторов равна 4D (M=4) векторам. Следовательно, компилятор 62 инструктирует упаковку texcoord0 и texcoord1 в один и тот же регистр атрибутов, обозначенный номером идентификатора 3 в столбце «Вновь назначенный идентификатор входного регистра атрибута». Во время упаковки texcoord0 могут быть назначены местоположения наименьших значащих битов 0011 маски, и texcoord1 могут быть назначены местоположения наименьших значащих битов 0011 маски, обозначенной в столбце «Новая маска» в Таблице 1. Маска 0011 обозначает, какая часть MD векторного регистра 3 данных, соответствующих texcoord0, может быть найдена. Подобным же образом, маска 1100 обозначает, какая часть MD векторного регистра 3 данных, соответствующих texcoord1, может быть найдена. Эта номенклатура позволяет двум или более атрибутам разделять общий регистр неперекрывающимся образом. Как можно понять, число бит в маске будет изменяться в зависимости от размерности.

С конкретной ссылкой на переменные texcoord0 и texcoord1 таблицы 66 выходных

символов VS, они упаковываются в один и тот же регистр переменной, имеющий номер идентификатора 0, обозначенный в столбце «Вновь назначенный идентификатор выходного регистра переменной», как наилучшим образом видно в Таблице 2. Новой маской для texcoord0 является 0011, которая является той же самой, что и старая маска. Однако новой маской для texcoord1 является 1100, которая отличается от старой маски. Таким образом, маска имеет M битов, причем каждый бит представляет некоторое местоположение в совместно используемом MD векторном регистре.

В другом примере, атрибуты «Вес» и «Нормаль» таблицы 64 входных символов VS упаковываются в один и тот же регистр атрибутов, имеющий номер идентификатора 2, обозначенный в «Вновь назначенный идентификатор входного атрибута» Таблицы 1. После того как компилятор 62 инструктирует повторное назначение идентификаторов регистров и новых масок, аппаратное обеспечение (HW) GPU 34 автоматически загрузит соответствующие переменные программы построения теней (атрибуты или переменные) в назначенные регистры согласно таблицеподобным командам (с обновленными масками), которые завершают упаковку первого уровня, инструктируемую компилятором 62.

Некоторый массив или матрица может быть логически разбита на 2D/3D/4D вектор или единственное число с плавающей точкой, затем может выполняться упаковка, инструктируемая компилятором 62. Массив может быть представлен группой плавающих чисел (чисел с плавающей точкой), 2D векторов, 3D векторов или 4D векторов. Например, массив из 10 плавающих чисел может быть разбит на два 4D вектора плюс один 2D вектор, или 10 индивидуальных плавающих чисел. Матрица 2x2 может быть разбита на два 2D вектора, матрица 3x3 - на три 3D вектора и матрица 4x4 - на четыре 4D вектора соответственно. Следовательно, компилятор 62 может инструктировать упаковку для следующих случаев: 2D вектор + 2D вектор; 3D вектор + плавающее число; 2D вектор + плавающее число [+ плавающее число]; и плавающее число + плавающее число [+ плавающее число [+ плавающее число]]. Эти примеры даны для 4D векторного регистра. Другие комбинации рассматриваются на основе числа размерностей. Использование входного регистрового файла и выходного регистрового файла может быть минимизировано посредством упаковки первого уровня.

После упаковки, инструктируемой компилятором 62, все переменные программы построения теней (переменные) могут все еще не быть размещены в 4D (MD) векторы, например могут присутствовать некоторые 3D векторы, некоторые 4D векторы и т.д. В примерной конфигурации выполняется некоторый механизм для плотной HW упаковки переменных в памяти переменной или кэше 54 вершин для упаковки переменной второго уровня.

Второй уровень: HW упаковка

В памяти переменной или кэше 54 вершин все переменные для вершины или пиксела хранят в блоке NxM буфера. N является числом переменных; M=4 означает 4D векторы. Блок памяти может трактоваться как некоторое число (NxM) непрерывных (последовательных) компонентов. Для 32 битов/компонентов и M=4 эти компоненты могут быть пронумерованы 0-((Nx4)-1). Например, N=8, 8x4 блок носителя данных может трактоваться как 32 непрерывных (последовательных) компонентов, пронумерованных 0-31.

На фиг. 4 буфер 58 упаковки представлен как 2xM (M=4) массив слотов. Стрелки указывают направление заполнения этих слотов в буфере 58 упаковки. Верхний ряд

буфера 58 упаковки обозначен как временный буфер 58A, тогда как второй ряд обозначен как рабочий буфер 58B. Таблица 5 иллюстрирует результаты HW упаковки.

Таблица 5. Упаковка при переносе из VS выходного регистрового файла в память переменной или кэш вершин

VS выходной регистровый файл				упаковка	Память переменной или кэш вершин			
V0.x	V0.y	V0.z			0: V0.x	1: V0.y	2:V0.z	3: V1.x
V1.x	V1.y	V1.z	V1.w		4: V1.y	5: V1.z	6: V1.w	7: V2.x
V2.x	V2.y				8: V2.y	9: V3.x	10: V3.y	11: V3.z
V3.x	V3.y	V3.z		➔	12: V4.y	13: V5.y	14: V5.z	15: V5.w
	V4.y				16: V6.x	17: V6.y	18: V6.z	19: V6.w
	V5.y	V5.z	V5.w		20: V7.x	21: V7.y	22: V7.z	
V6.x	V6.y	V6.z	V76.w					
V7.x	V7.y	V7.z						

Упаковка второго уровня может быть осуществлена в HW посредством первого заполнения последовательно в ряд временного буфера 58A (первого ряда буфера 58 упаковки). После того как временный буфер 58A окажется полным, содержимое временного буфера 58A может быть перенесено для хранения в кэш 54 вершин. В этой конфигурации буфер 58 упаковки включает в себя первый ряд из M слотов, обозначенный как временный буфер 58A, и второй ряд из M слотов, обозначенный как рабочий буфер.

С использованием примера, изложенного в Таблице 5, HW упаковка начинается со считывания переменной V0, имеющего три компонента, обозначенных как V0.x, V0.y и V0.z, из VS выходного регистрового файла 57 и последовательного заполнения слотов X, Y, Z и W временного буфера 58A (верхнего ряда) переменными V0.x, V0.y и V0.z. Как можно видеть, слот W временного буфера является свободной. Переменные V0.x, V0.y и V0.z еще не посылаются в кэш 54 вершин, пока временный буфер 58A не будет заполнен.

HW упаковка продолжается посредством считывания переменной V1, имеющего четыре компонента, обозначенных как V1.x, V1.y, V1.z и V1.w, из VS выходного регистрового файла 57 и заполнения остающегося слота (слотов) во временном буфере 58A. В этом случае слот W временного буфера 58A (верхнего ряда) заполняется переменной V1.x. Оставшиеся компоненты переменной V1.y, V1.z и V1.w заполняются последовательно в слоты X, Y и Z второго ряда или рабочего буфера 58B. Когда временный буфер 58A полностью заполнен, содержимое временного буфера 58A может быть записано в (первый) ряд кэша 54 вершин для опустошения временного буфера 58A.

Если временный буфер 58A пуст, то содержимое оставшихся компонентов переменной V1.y, V1.z и V1.w, заполненные последовательно в слоты X, Y и Z рабочего буфера 58B, переносятся во временный буфер 58A. Опять же, временный буфер 58A не является пустым. Таким образом, HW упаковка продолжается посредством считывания переменной V2, имеющего два компонента, обозначенных как V2.x и V2.y, из VS выходного регистрового файла 57 и заполнения остающегося слота (слотов) во временном буфере 58A. В этом случае слот W временного буфера 58A (верхнего ряда) заполняется переменной V2.x. Остающийся компонент V2.y переменной заполняется в слот X второго ряда или рабочего буфера 58B. Когда временный буфер 58A полностью заполнен, содержимое временного буфера 58A

может быть записано во (второй) ряд кэша 54 вершин для опустошения временного буфера 58A.

Этот процесс продолжается для переменных в VS выходном регистровом файле 57. В этом примере, так как последняя переменная заполняет только слоты X, Y и Z временного буфера 58A, это содержимое записывается в память переменной или кэш 54 вершин с маской=xyz или (111).

Временный буфер 58A и рабочий буфер 58B из буферов 58 упаковки готовы к работе. Когда временный буфер 58A является заполненным и готовым к выписыванию в память переменной или кэш 54 вершин, другой буфер (рабочий буфер 58B) может быть одновременно заполнен. В каждый момент времени может использоваться как шина считывания, так и шина записи для четырех (M) компонентов. Если данные одного считывания или записи меньше чем 4 компонента, то маска считывания или записи используется для указания того, какие компоненты являются достоверными для считывания или записи.

После завершения HW упаковки второго уровня, идентификатор регистра, соответствующий «Вновь назначенному идентификатору выходного регистра переменной VS» в Талице 4, соответствующий упакованным переменным в таблице выходных символов VS (Таблица 2), будет изменен для соответствия памяти переменной или кэшу 54 вершин, обозначенному в столбце «Вновь назначенный идентификатор выходного регистра переменной VS» в Таблице 6. Для простоты и гибкости, отношение выходного идентификатора к местоположению в кэше 54 вершин назначается на основе единичного компонента вместо векторного регистра. Для этого примера предполагается, что texcoord0, чей идентификатор = 0, и texcoord1, чей идентификатор = 2, упаковываются в первый ряд памяти переменной или кэша 54 вершин, цвет0, чей идентификатор = 4, - во второй ряд, и цвет1, чей идентификатор = 8, - в третий ряд. Позиция и texcoord2 не используются в FS 70, таким образом, память/упаковка не распределяется для них в FS входном регистровом файле 79. Таким образом, «Вновь назначенный идентификатор входного регистра переменной FS» не обеспечен в Таблице 4 или Таблице 6.

HW упаковка второго уровня осуществляется посредством HW, но таблица 86 связывания, такая, как показанная в Таблице 6, обновляется компоновщиком 80 драйвера 61. Драйвер 61 способен вычислить новый идентификатор регистра/ идентификатор компонента в памяти переменной или кэше 54 вершин для каждого компонента переменной на основе одного и того же механизма упаковки и таблиц 64 и 66 входных и выходных символов VS и таблицы 74 входных символов FS и т.д. на фиг. 4. Таблица 4 иллюстрирует, что представляла бы собой таблица связывания без HW упаковки второго уровня.

Таблица 6
Таблица связывания для VS выходов и FS входов после HW упаковки второго уровня

Имя переменной	Вновь назначенный выходной идентификатор (переменная) VS	Вновь назначенный входной идентификатор (переменная) FS	Новая маска
Позиция			
Цвет0	4	2	1111
Цвет1	8	3	0111
Texcoord0	0	1	0011
Texcoord1	2	1	1100
Texcoord2			

Обходные атрибуты

Программируемая программа построения теней вершин, такая как VS 60, является ключевым блоком вычисления в современном GPU как в игровых устройствах ПК, так и в мобильных устройствах. VS 60 является вычислительно потребляющим мощность, а также обычно узким местом работы. Однако некоторые приложения могут не использовать функциональность программы построения теней. Другим рассмотрением является то, что некоторые входные данные для VS 60 могут быть непосредственно перемещены во выходные данные без какой-либо необходимости в вычислениях.

Простейшим решением для этих функций является пропускание всех входных данных в программу построения теней, когда программа построения теней исполняет команды перемещения. Однако такое решение будет потреблять много вычислительной мощности и вызовет уменьшение в производительности программы построения теней. Снижение производительности является результатом 1) ненужной полосы частот трафика для вводов/выводов данных; и 2) ненужных команд перемещений, выполняемых в программе построения теней.

Таким образом, GPU 34 построен и организован с входным трактом обхода от декодера 50 входного потока. Входной тракт обхода может идти непосредственно к кэшу 54 вершин. Драйвер 61 или компилятор 62 может определить, какие входные данные могут быть направлены непосредственно в обход к кэшу 54 вершин, и какие входные данные должны быть загружены в программу 60 построения теней. Компилятор 62 удалит все ненужные команды перемещения из программы построения теней для обходных входных данных.

Малое логическое устройство 51 управления аппаратного обеспечения, фиктивно показанное на фиг. 4, находится в декодере 50 входного потока. Таким образом, когда принятые входные данные определены как «обход», после декодирования входного формата, эти входные данные будут посланы по обходному тракту и сохранены в кэше 54 вершин. Только те принятые входные данные, которые не обозначены как «обход», будут упакованы в VS входном регистровом файле 56 и посланы к программе 60 построения теней вершин.

В примерном варианте осуществления обходные атрибуты упаковываются в буфере 52 упаковки перед сохранением в кэше 54 вершин. Компилятор 62 будет модифицировать маску и/или идентификатор регистра способом, описанным выше в связи с буфером 58A упаковки, индекс кэша будет пропущен к кэшу 54 вершин вместе с обходными входными данными. Выходные данные из программы 60 построения теней будут иметь один и тот же идентификатор/индекс для одной и той же вершины, таким образом, кэш 54 вершин может легко синхронизировать обходные входные данные с выходными данными программы построения теней вершин.

Фиг. 8A и 8B показывают программу построения теней вершин перед и после удаления обходных атрибутов. Некоторые программы построения теней вершин имеют MOV команды в строках, обозначенных посредством L3 и L4. Эти MOV команды вызывают перемещение от входных регистров, ассоциированных с атрибутами, к выходным регистрам, ассоциированным с переменными. Такие атрибуты могут быть пущены в обход из программы 60 построения теней вершин. Например, на фиг. 8A, параметры v0, v1, v2 являются входными атрибутами, а oPos, oFog, oT0 и oD0 являются выходными переменными. В этом примере входной атрибут v1 в строках L1 и L3 и входной атрибут v2 в строках L2 и L4 не вовлекают каких-либо вычислений в программе 60 построения теней вершин и просто перемещаются в oT0 и oD0. Следовательно, атрибуты v1 и v2 могут быть пущены в

обход непосредственно к памяти переменной или кэшу 54 вершин перед исполнением программы (набора команд) 60 построения теней вершин. После того как атрибуты v1 и v2 пущены в обход, они не будут посланы в программу 60 построения теней вершин, обозначенную посредством удаления строк L1, L2, L3 и L4 на фиг. 8B. Кроме того, выходные переменные oT0 и oD0 не выводятся из программы 60 построения теней вершин, обозначенной отсутствием строк L3 и L4 на фиг. 8B. Следовательно, функция обхода экономит полосу частот трафика и мощность вычисления программы построения теней вершин.

Для обхода атрибутов, HW упаковка второго уровня настраивается, как описано ниже. Обходные атрибуты подвергаются HW упаковке второго уровня только в буфере 52 упаковки. Буфер 52 упаковки принимает обходные атрибуты от потокового декодера 50. Потоковый декодер 50 отвечает за извлечение (атрибутов) потока вершин из основной (внешней) памяти 40 и преобразования формата из различных форматов атрибутов в IEEE формат с плавающей точкой. Драйвер 61 будет связываться с потоковым декодером 50, чьи атрибуты будут пущены в обход, и чьи атрибуты будут посланы в VS входной регистровый файл 56 для программы 60 построения теней вершин. Эти обходные атрибуты будут упакованы тем же способом, что описан выше, с использованием вышеупомянутых временного буфера 58A и рабочего буфера 58B. Необходные атрибуты будут посланы и упакованы в VS входном регистровом файле 56 программы 60 построения теней вершин.

Переменные как из обходных атрибутов, так и из VS выходного регистрового файла 57 будут заполнены во всю память переменной или кэш 54 вершин, как целый отпечаток переменной. Для упрощения, переменные из обходных атрибутов упаковываются и сохраняются в первых нескольких рядах в памяти переменной или кэше 54 вершин, а выходные данные VS, упакованные в буфере 58 упаковки, сохраняются после этого в памяти переменной или кэше 54 вершин. Например, опять со ссылкой на фиг. 8A, вывод переменной (обходной атрибут) oD0 (v2) упаковывается во временный буфер 52A буфера 52 упаковки и сохраняется в первом ряду в памяти переменной или кэше 54 вершин. Вывод переменной (обходной атрибут) oT0 (v1) упаковывается или сохраняется в двух нижних значащих компонентах во втором ряду в памяти переменной или кэше 54 вершин. Выводы VS oPos и oFog будут упакованы или сохранены после этого, начиная с двух наиболее значащих компонентов второго ряда. В этом случае oPos.xy будет упакован в слоте zw временного буфера и затем записан во второй ряд памяти переменной или кэша 54 вершин с маской записи = zw. Таким образом, он упаковывается последовательно после oT0 в тот же ряд, но в другие места компонентов. oPos.zw и oFog будут упакованы в рабочий буфер 52B в слоте компонентов xyz и записаны в третий ряд памяти переменной или кэша вершин с маской записи = xyz. Таблица 86 связывания будет обновлена соответственно.

Фиг. 9A и 9B показывают другую программу построения теней вершин перед и после удаления обходных атрибутов. Команды перемещения в строках, обозначенных стрелками L5, L6, L7, L8 и L9 могут быть обойдены. Например, в строке, обозначенной стрелкой L5, вывод переменной oT0 (v1) может быть обойден. Другие выводы переменной oT1 (v1), oT2 (v3), oD0 (v4) и oD1 (v5) могут быть также обойдены. На фиг. 9B, строки, обозначенные стрелками L5, L6, L7, L8 и L9 фиг. 9A, удалены.

Фиг. 10A и 10B показывают еще одну программу построения теней вершин перед и после удаления обходных атрибутов. Команды перемещения в строках, обозначенных стрелками L10 и L11, могут быть обойдены. На фиг. 10B, строки, обозначенные стрелками L10 и L11 на фиг. 10A, удалены.

Фиг. 11A и 11B показывают еще одну программу построения теней вершин перед и после удаления обходных атрибутов. Команды перемещения в строках, обозначенных стрелками L12, L13, L14, L15 и L16, могут быть обойдены. На фиг. 11B, строки, обозначенные стрелками L12, L13, L14, L15 и L16 фиг. 11A, удалены.

Фиг. 12A и 12B показывают еще одну программу построения теней вершин перед и после удаления обходных атрибутов. Команды перемещения в строках, обозначенных стрелками L17 и L18, могут быть обойдены. На фиг. 12B, строки, обозначенные стрелками L17 и L18 фиг. 12A, удалены. Примеры, показанные на фиг. 8A, 8B, 9A, 9B, 10A, 10B, 11A, 11B, 12A и 12B, приведены с иллюстративными целями, и другие команды перемещения или атрибуты, не требующие вычислений, могут быть обозначены как «обходные атрибуты».

Как легко видеть, преимущества процесса обходных атрибутов включают в себя: 1) уменьшение размера кода программы построения теней и числа команд исполнения; 2) уменьшение полосы частот трафика входов/выходов; 3) уменьшение размера регистрового файла для большего количества вершин для покрытия времени ожидания ALU (арифметического и логического блока) и времени ожидания загрузки текстуры; 4) лучшую производительность благодаря меньшему числу команд и большему числу вершин для покрытия времени ожидания; 5) экономию мощности благодаря меньшему числу выполняемых команд и меньшему трафику; 6) общего для программы построения теней обхода/блокирования; 7) опцию для драйвера 61 для настройки работы посредством перемещения части программы построения теней вершин к CPU или DSP для балансирования нагрузки между CPU/DSP 32 и GPU 34; и 8) опцию для драйвера 61 для работы вокруг неожиданных результатов.

Было определено, что большинство из программ построения теней вершин (VS) из реальных игр и эталонных тестов имеют некоторые входные данные, перемещаемые непосредственно в выходные данные. Таблица 7 иллюстрирует различные программы построения теней и сравнение сэкономленного входного трафика и сэкономленного выходного трафика на основе функции обхода, описанной здесь. Таблица 7 также обеспечивает отношение сэкономленных команд.

Таблица 7. Экономия полосы частот трафика и вычислений

Программы построения теней	Сэкономленный входной трафик (DW)	Сэкономленный выходной трафик (DW)	Сэкономленные команды (скаляр)
VSF8 (3DMark06) (Фиг. 12A и 12B)	4/17 = 23.5%	4/27 = 14.8%	4/97 = 4.1%
VSF12 (FarCry) (Фиг. 8A и 8B)	6/10 = 60%	6/11 = 54.5%	4/28 = 14.3%
VSF14 (FarCry) (Фиг. 9A и 9B)	13/21 = 61.9%	15/20 = 75%	15/70 = 21.4%
VSF17 (FarCry) (Фиг. 10A и 10B)	0%	3/7= 42.8%	3/19 = 16.8%
VSF25 (FarCry) (Фиг. 11A и 11B)	9/13 = 69.2%	15/21 = 71.4%	15/61 = 24.6%

Фиг. 13A-13C показывают общую блок-схему процесса 200 упаковки переменных программы построения теней, скомбинированного с обходом атрибутов. Процесс 200 упаковки переменных программы построения теней будет описан в связи с блок-схемой фиг. 4. Процесс 200 упаковки переменных программы построения теней начинается с блока 201, где формат входных атрибутов декодируется, например, посредством потокового декодера 50. За блоком 201 следует блок 202, где определяется, являются ли атрибуты из потокового декодера 50 «обходными атрибутами». Если определением является «Да», то за блоком 202 следует блок 204, где достоверные компоненты (обходного) атрибута собираются во временный буфер 52A. За блоком 204 следует блок 206, где определяется, заполнен ли временный буфер 52A. В качестве примера, предел из M (M=4) компонентов обходного атрибута может быть заполнен во временный буфер 52A. Временный буфер 52A заполняется также посредством заполнения рабочего буфера 52B.

Однако, если определением в блоке 206 является «Нет», то этот процесс ответвляется к блоку 211. Блок 211 является блоком определения, оценивающим, был ли достигнут последний входной атрибут. Подробности блока 211 будут описаны позже.

Когда временный буфер 52a заполнен, за блоком 206 следует блок 208, где компоненты обходных атрибутов, сохраненные или заполненные во временный буфер 52A, посылаются и сохраняются в кэше 54 вершин. Как описано выше, обходные атрибуты в рабочем буфере 52B затем переносятся во временный буфер 52A, пока последний не заполнится или не заполнится повторно. За блоком 208 следует блок 211, подлежащий описанию позже.

Возвращаясь опять к блоку 202, если атрибуты являются необходными атрибутами, что означает, что определением в блоке 202 является «Нет», то, согласно заданным командам упаковки, необходные атрибуты будут упакованы в VS входной

регистровый файл 56 в блоке 210. За блоком 210 следует блок 211, где определяется, был ли достигнут последний входной атрибут. Если определением является «Нет», то блок 211 делает петлю обратно к блоку 201, где декодируются другие входные атрибуты. В противном случае, если определением является «Да», то за блоком 211 следует блок 212, где оставшиеся обходные атрибуты во временном буфере 52A посылаются в кэш 54 вершин.

За блоком 212 следует блок 213, где определяется, доступны ли какие-либо необходные атрибуты. Если определением является «Нет», то процесс 200 завершается. Однако, если определением в блоке 213 является «Да», то за блоком 213 следует блок 214 фиг. 13B. В блоке 214 необходные атрибуты затем посылаются к VS 60. После того как компоненты необходных атрибутов были посланы к VS 60, VS 60 выполняет операции построения теней вершин в блоке 216. После завершения VS 60 достоверные компоненты выходной переменной автоматически упаковываются в VS выходной регистровый файл 57 во время выполнения команд программы построения теней, что завершает упаковку компилятора первого уровня в блоке 218. Упаковка в блоке 218 соответствует блоку 102 фиг. 7.

За блоком 218 следует блок 222 на фиг. 13C. Выходные переменные из VS выходного регистрового файла 57 собирают во временный буфер 58A, как описано выше в связи с Таблицей 5. Временный буфер 58A заполняется в комбинации с рабочим буфером 58B. За блоком 222 следует блок 224 для определения того, заполнен ли временный буфер 58A. Если «Нет», то процесс возвращается к блоку 222. Если определением является «Да», то за блоком 224 следует блок 226, где содержимое временного буфера 58A посылается в кэш 54 вершин. За блоком 226 следует блок 228, где определяется, достигнут ли конец файла в VS выходном регистровом файле 57. Если определением является «Нет», то процесс возвращается к блоку 222. Если определением является «Да», то за блоком 228 следует блок 230, где оставшиеся компоненты переменной во временном буфере 58A посылаются в кэш 54 вершин.

После упаковки полоса частот трафика уменьшается. Память высоко используется и также улучшается производительность.

Могут использоваться альтернативные механизмы упаковки. Например, переменные в VS выходном регистровом файле 57 не упаковываются с использованием HW упаковки второго уровня. Вместо этого файл 57 копируется в кэш 54 вершин, как есть. В связи с Таблицей 5 левая сторона Таблицы 5 копируется в кэш 54 вершин. Это сохраняет один и тот же макет и форму. Тот же самый механизм упаковки осуществляется после ассемблера примитивов и растеризатора 90, где результаты растеризатора посылаются в буфер 92 изменений. Ассемблер примитивов и растеризатор 90 пропустит вычисления для дефектных (маска = 0) компонентов на основе масок в Таблице 2 для экономии вычислений.

Компоновщик и связывание

Переменная из VS 60 будет входными данными для FS 70. Таким образом, символ переменной для VS 60, генерируется компилятором 62 и вводится в таблицу 66 выходных символов VS. Соответствующий ввод FS 70 ограничен выводом соответствующего символа переменной в таблице 66 выходных символов VS, определенным его символом переменной или именем переменной. Таким образом, если символ переменной, обозначенный именем переменной в таблице 66 выходных символов VS, совпадает с одной позицией, обозначенной именем переменной FS, в таблице 74 входных символов FS (Таблица 3), то вывод (переменная) из VS 60 ограничен входом FS 70. Компоновщик 80 определяет, какой выход VS ограничен

каким FS входом, так как порядок и упаковка в VS 60 обычно отличается от FS 70. Компоновщик 80 является частью драйвера 61, который генерирует команды 82 связывания или таблицу 86 связывания для модуля 84 перераспределения и загрузки переменной на фиг. 5.

Подобное решение связывания применяется для связывания между потоковым декодером 50 вершин и VS входом, обозначенного таблицей 64 входных символов VS. Кроме того, решение связывания применяется для связывания FS выхода, обозначенного таблицей 76 выходных символов FS и входом блока операций на выборках. Компоновщик 80 может использоваться для любых двух соседних программируемых стадий обработки.

Фиг. 14 показывает общую блок-схему процесса 300 связывания. Процесс 300 связывания начинается с блока 302, где компоновщик 80 ищет и сравнивает один и тот же символ как из таблицы 66 выходных символов VS, так и из таблицы 76 входных символов FS в блоке 302. В блоке 304 переменная, ассоциированная с совпадающим символом, считывается из буфера 92 переменной. Блок 206 посылает эту переменную в FS входной регистровый файл 79. Таким образом, связывание завершено. Примерная результирующая таблица связывания показана в Таблице 6. Процесс 300 связывания повторяется для каждой переменной, требуемого для FS 70.

Из-за упаковки в компиляторе 62 таблица 66 выходных символов VS отличается от таблицы 76 входных символов FS. Таким образом, предпочтительная команда связывания определяется на основе компонента переменной.

Как можно видеть, упаковка второго уровня делает плотную упаковку переменной возможной и легкой. Упаковка компилятора первого уровня делает входы и выходы меньше, что уменьшает размер отпечатка регистрового файла. HW упаковка является простой и эффективной. Процесс упаковки дополнительно уменьшает полосу частот трафика входов/выходов. Процесс 100 или 200 упаковки высоко использует кэш-память и экономит мощность благодаря меньшему трафику.

GPU конвейер, показанный на фиг. 4 и 5, использует общее связывание для любых двух соседних программируемых стадий обработки. Это позволяет осуществить иерархическое связывание, отображение для больших и сложных структур переменной, таких как массив и матрица, на векторы и числа с плавающей точкой. Процесс упаковки позволяет компилятору 62 переупорядочить или перераспределить регистры свободно для оптимизации. Этот процесс позволяет драйверу/компоновщику легко удалить некоторые из VS выходов, если они не используются в FS посредством модификации некоторых из команд связывания.

В одной или нескольких примерных конфигурациях, функции, описанные здесь, могут быть реализованы в аппаратном обеспечении, программном обеспечении, программно-аппаратных средствах или любой их комбинации. Если эти функции реализованы в программном обеспечении, то они могут быть сохранены или переданы как одна или несколько команд или код на считываемом компьютером носителе. Считываемые компьютером носители включают в себя как компьютерные носители данных, так и среды передачи данных, включающие в себя любой носитель, который облегчает перенос компьютерной программы из одного места в другое. Носителем данных может быть любой доступный носитель, доступ к которому может быть получен посредством компьютера. В качестве примера, а не ограничения, такие считываемые компьютером носители могут содержать RAM, ROM, EEPROM, CD-ROM или другую память на оптическом диске, память на магнитном диске или другие магнитные ЗУ, или любой другой носитель, который может использоваться для

переноса или сохранения желаемого программного кода в форме команд или структур данных, доступ к которым может быть получен посредством компьютера. Также любое подключение правильно называется считываемым компьютером носителем. Например, если программное обеспечение передается от Web-сайта, сервера или другого удаленного источника с использованием коаксиального кабеля, волоконно-оптического кабеля, витой пары, цифровой линии абонента (DSL) или беспроводных технологий, таких как инфракрасное излучение, радио- и микроволны, то коаксиальный кабель, волоконно-оптический кабель, витая пара, DSL или беспроводные технологии, такие как инфракрасное излучение, радио- и микроволны включены в определение носителя. Диск (disk и disc), как используется здесь, включает в себя компакт-диск (CD), лазерный диск, оптический диск, цифровой универсальный диск (DVD), флоппи-диск и диск «blu-ray», где диски (disks) обычно воспроизводят данные магнитным способом, тогда как диски (discs) воспроизводят данные оптически лазерами. Комбинации вышеупомянутого должны быть также включены в объем считываемого компьютером носителя.

Предыдущее описание раскрытых конфигураций обеспечено для того, чтобы позволить любому специалисту в данной области техники осуществить или использовать данное описание. Различные модификации этих конфигураций будут понятны для специалистов в данной области техники, и общие принципы, определенные здесь, могут быть применены к другим конфигурациям, не выходя за рамки сущности и объема данного описания. Таким образом, данное описание не предназначено для ограничения конфигурациями, показанными здесь, а должно соответствовать наиболее широкому объему, совместимому с принципами и элементами новизны, описанными здесь.

Формула изобретения

1. Устройство обработки информации, содержащее:

носитель данных, имеющий множество совместно используемых M-мерных (MD) регистров; и

блок обработки для реализации набора операций для упаковки в каждый совместно используемый MD регистр одной или нескольких переменных величин программы построения теней, общее число компонентов которых равно M;

отличающееся тем, что дополнительно содержит

кэш вершин, имеющий множество MD регистров кэша; и

в котором носитель данных включает в себя буфер упаковки с MD регистром для упаковки M компонентов из множества совместно используемых MD регистров последовательно в нем и переноса содержимого MD регистра буфера упаковки, когда он заполнен, в соответствующий MD регистр кэша в кэше вершин.

2. Устройство по п.1, в котором набор операций упаковывает одно из: двух переменных величин программы построения теней 2D вектора; переменной величины программы построения теней 3D вектора и переменной величины с плавающей точкой программы построения теней; переменной величины программы построения теней 2D вектора и двух различных переменных величин с плавающей точкой программы построения теней; и четырех переменных величин с плавающей точкой программы построения теней.

3. Устройство по п.1, в котором блок обработки реализует второй набор операций для связывания выходных переменных в кэше вершин с набором входных переменных для программы построения теней фрагментов.

4. Устройство по п.3, в котором второй набор операций включает в себя операции для согласования имен символов переменной, соответствующих выходным переменным в программе построения теней вершин, с соответствующими именами символов переменной в наборе входных переменных для программы построения теней фрагментов.

5. Устройство по п.1, дополнительно содержащее второй носитель данных, имеющий второе множество совместно используемых MD регистров; в котором переменные величины программы построения теней дополнительно содержат необходные входные атрибуты для ввода в программу построения теней вершин и обходные входные атрибуты; и в котором набор операций включает в себя операции для упаковки необходных входных атрибутов во втором множестве совместно используемых MD регистров и заполнения каких-либо остающихся необходных входных атрибутов во второй носитель данных для создания входного файла программы построения теней вершин.

6. Устройство по п.5, в котором кэш вершин дополнительно содержит второе множество MD регистров кэша; и в котором носитель данных включает в себя второй буфер упаковки с MD регистром для упаковки M компонентов обходных входных атрибутов последовательно в него и переноса содержимого MD регистра второго буфера упаковки, когда он заполнен, в каждый MD регистр кэша второго множества MD регистров кэша в кэше вершин.

7. Устройство по п.1, дополнительно содержащее второй носитель данных, имеющий второе множество совместно используемых MD регистров; в котором переменная величина программы построения теней содержит набор входных атрибутов для ввода в программу построения теней вершин; и в котором набор операций включает в себя операции для упаковки набора входных атрибутов во втором множестве совместно используемых MD регистров, пока компоненты остающихся неупакованных входных атрибутов не превысят M, если они упакованы, и заполнения остающихся входных атрибутов во второй носитель данных для создания входного файла программы построения теней вершин.

8. Устройство по п.1, в котором блок обработки является частью одного устройства из: сотовый телефон, беспроводное устройство, устройство беспроводной связи, игровая консоль, электронный секретарь (PDA), портативный компьютер и аудио/видеоустройство.

9. Устройство по п.1, в котором носитель данных или блок обработки содержатся в некоторой интегральной схеме.

10. Устройство по п.1, в котором носитель данных или блок обработки содержатся в некотором процессоре.

11. Устройство обработки информации, содержащее:

средство хранения, имеющее множество совместно используемых M-мерных (MD) регистров, для хранения набора переменных величин программы построения теней; и

средство упаковки для упаковки в каждый совместно используемый MD регистр одной или нескольких переменных величин программы построения теней из набора переменных величин программы построения теней, общее число компонентов которых равно M;

отличающееся тем, что дополнительно содержит

средство кэширования вершин, имеющее множество MD регистров кэша;

и

в котором средство хранения включает в себя буфер упаковки с MD регистром для

упаковки M компонентов из множества совместно используемых MD регистров последовательно в нем и переноса содержимого MD регистра буфера упаковки, когда он заполнен, в соответствующий MD регистр кэша в средстве кэширования вершин.

12. Устройство по п.11, в котором средство упаковки содержит средство для упаковки по меньшей мере одного из: двух переменных величин программы построения теней 2D вектора; переменной величины программы построения теней 3D вектора и переменной величины с плавающей точкой программы построения теней; переменной величины программы построения теней 2D вектора и двух различных переменных величин с плавающей точкой программы построения теней; и четырех переменных величин с плавающей точкой программы построения теней.

13. Устройство по п.11, в котором средство упаковки является частью одного устройства из: сотовый телефон, беспроводное устройство, устройство беспроводной связи, игровая консоль, электронный секретарь (PDA), портативный компьютер и аудио/видеоустройство.

14. Устройство по п.11, в котором носитель данных является считываемым компьютером носителем, имеющим команды, чтобы заставить компьютер выполнять набор операций для упаковки в каждый совместно используемый MD регистр.

15. Способ обработки информации, содержащий:

первый этап упаковки, содержащий упаковку одной или нескольких переменных величин программы построения теней набора переменных величин программы построения теней, общее число компонентов которых равно M, в каждый совместно используемый M-мерный (MD) регистр из первого множества совместно используемых MD регистров; и

повторение упаковки в первое множество совместно используемых MD регистров, пока любые оставшиеся переменные величины программы построения теней не будут неупаковываемыми;

отличающийся тем, что способ дополнительно содержит

второй этап упаковки, содержащий последовательную упаковку M компонентов переменных величин программы построения теней из первого множества совместно используемых MD регистров в первую часть второго множества совместно используемых MD регистров; и

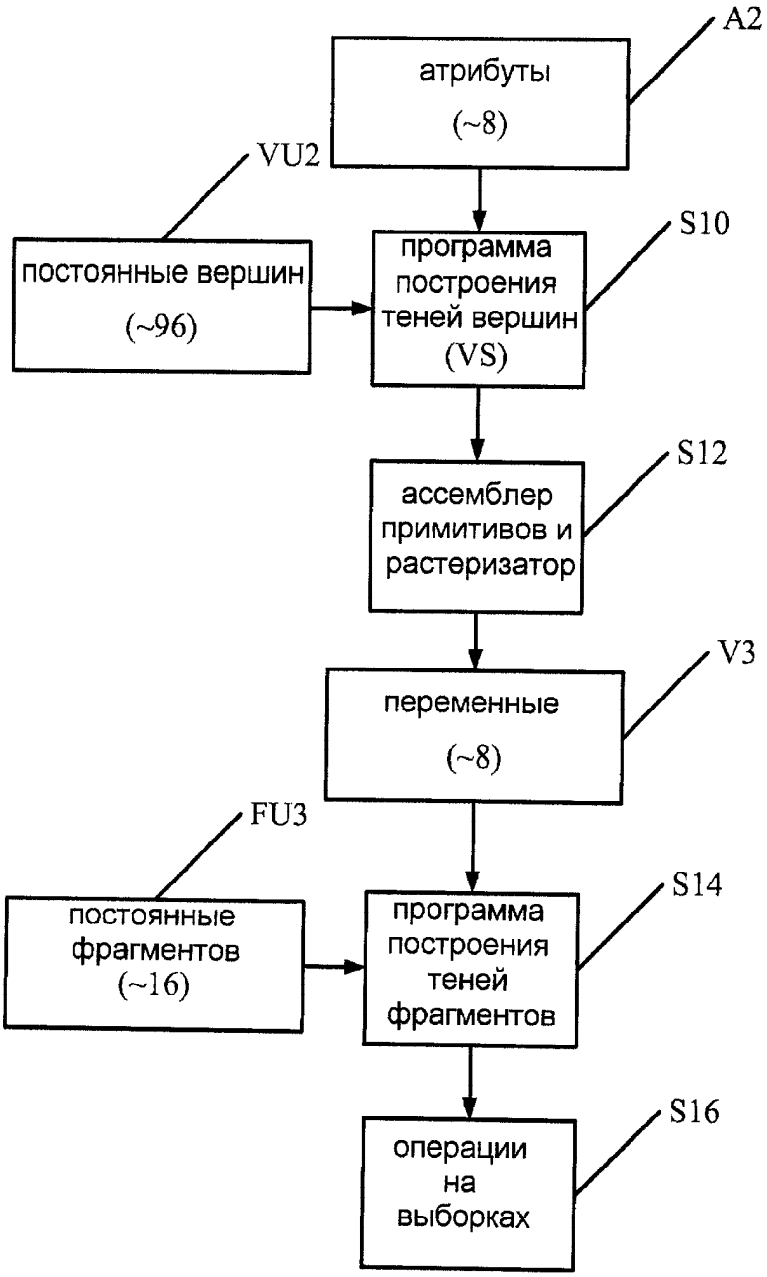
перенос содержимого первой части второго множества совместно используемых MD регистров, когда она заполнена, в соответствующий MD кэш-регистр кэша вершин.

16. Способ по п.15, в котором набор переменных величин программы построения теней содержит набор необходных входных атрибутов для программы построения теней вершин; и в котором упаковка включает в себя упаковку набора входных атрибутов в множество совместно используемых MD регистров.

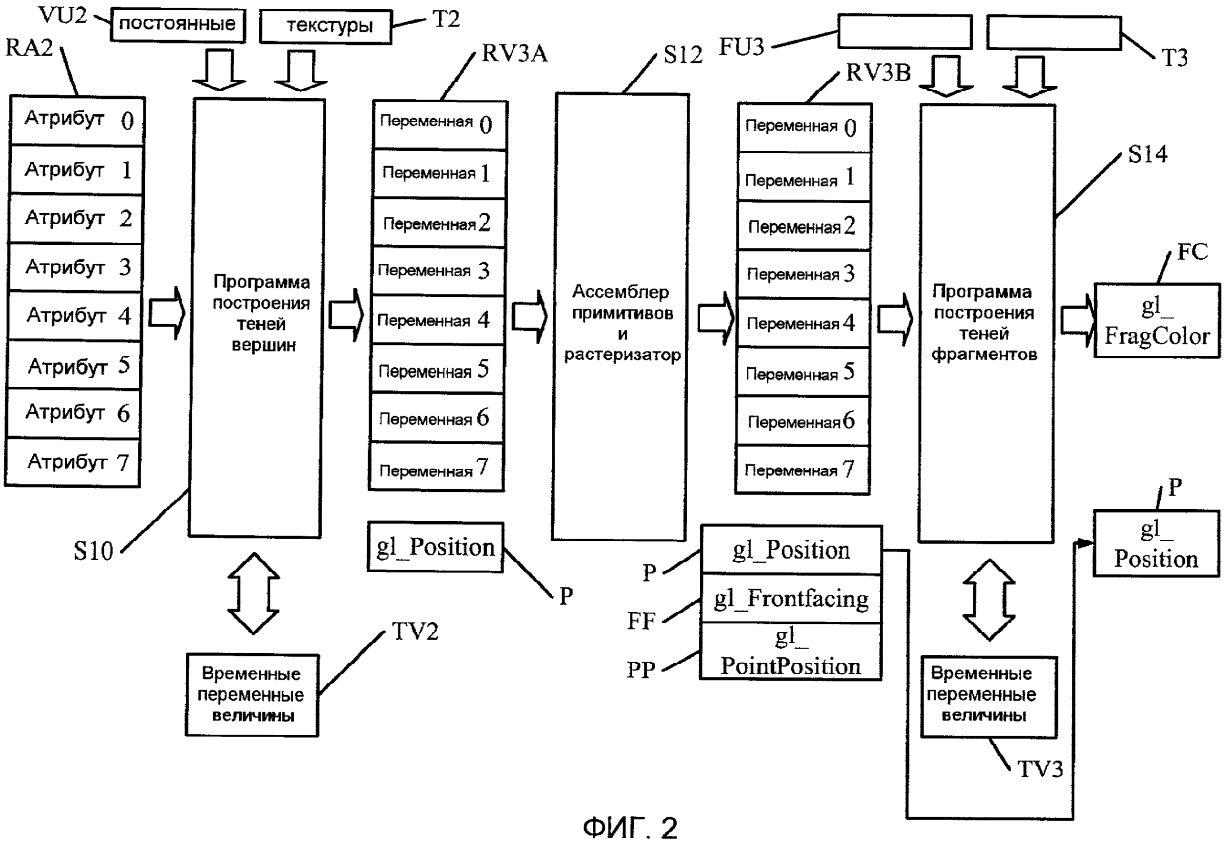
17. Способ по п.16, дополнительно содержащий:

упаковку M компонентов обходных входных атрибутов последовательно в MD временный регистр буфера упаковки и

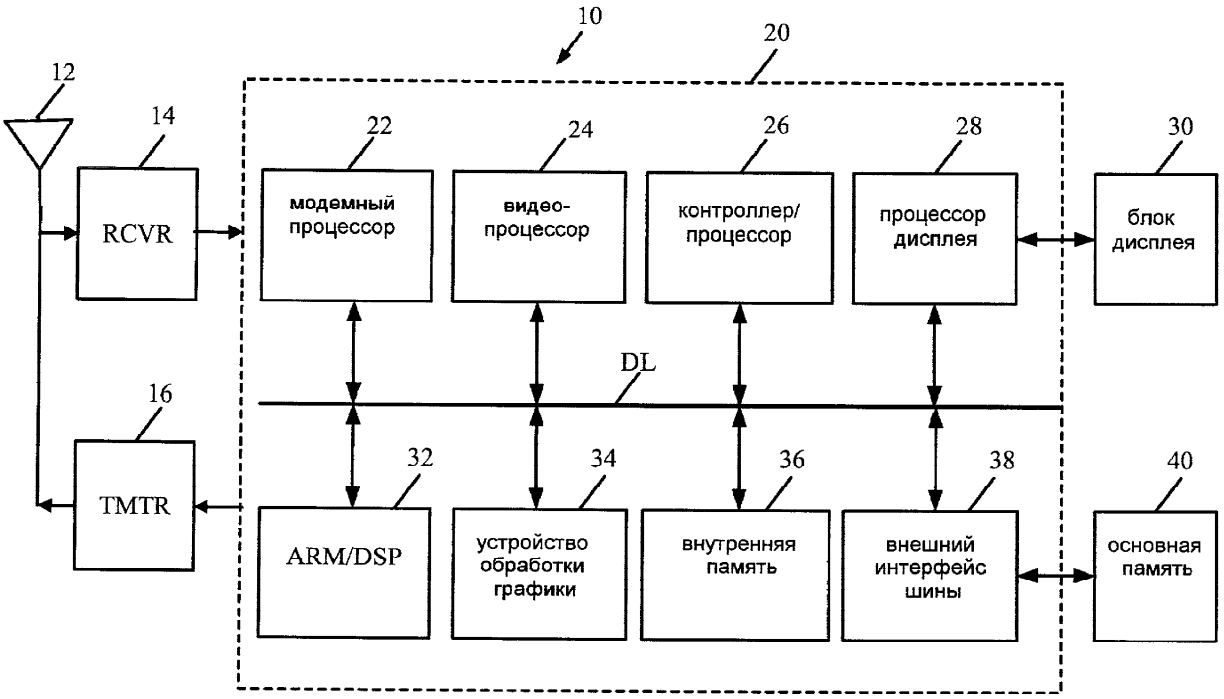
перенос содержимого MD временного регистра, когда он заполнен, в соответствующий MD регистр кэша в кэше вершин.


A2
атрибуты
(~8)
VU2
постоянные вершин
(~96)
S10
программа построения теней вершин (VS)
S12
ассемблер примитивов и растеризатор
V3
переменные
(~8)
FU3
постоянные фрагментов
(~16)
S14
программа построения теней фрагментов
S16
операции на выборках

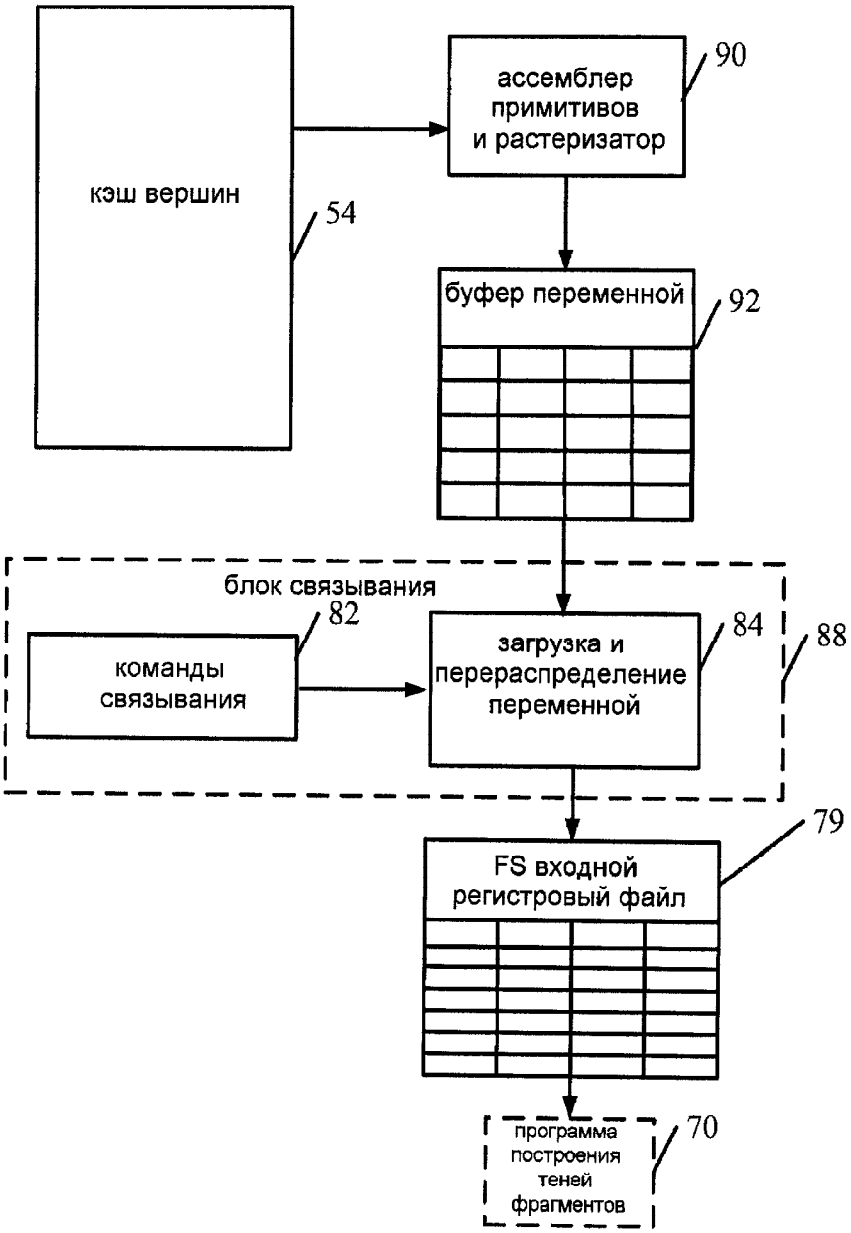
ФИГ. 1


VU2
постоянные
текстуры
T2
RA2
RV3A
S12
FU3
T3
RV3B
S14
FC
Атрибут 0
Атрибут 1
Атрибут 2
Атрибут 3
Атрибут 4
Атрибут 5
Атрибут 6
Атрибут 7
Программа построения теней вершин
Переменная 0
Переменная 1
Переменная 2
Переменная 3
Переменная 4
Переменная 5
Переменная 6
Переменная 7
Ассемблер примитивов и растеризатор
Программа построения теней фрагментов
gl_ FragColor
S10
gl_Position
P
TV2
Временные переменные величины
FF
PP
gl_Frontfacing
gl_ PointPosition
TV3

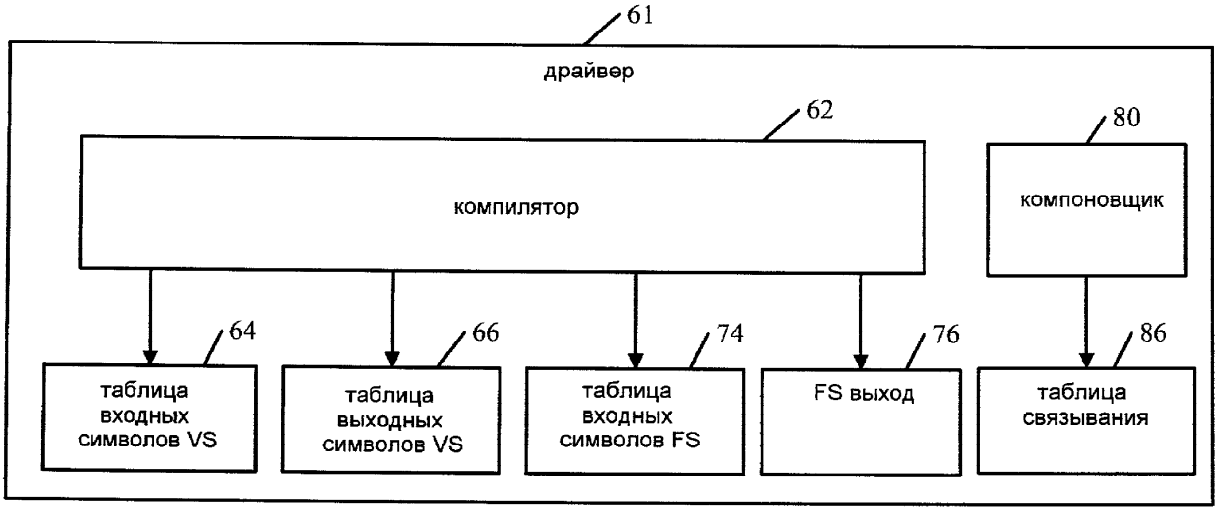
ФИГ. 2


10
20
12
14
22
24
26
28
30
RCVR
модемный процессор
видео-процессор
контроллер/ процессор
процессор дисплея
блок дисплея
DL
16
32
34
36
38
40
TMTR
ARM/DSP
устройство обработки графики
внутренняя память
внешний интерфейс шины
основная память

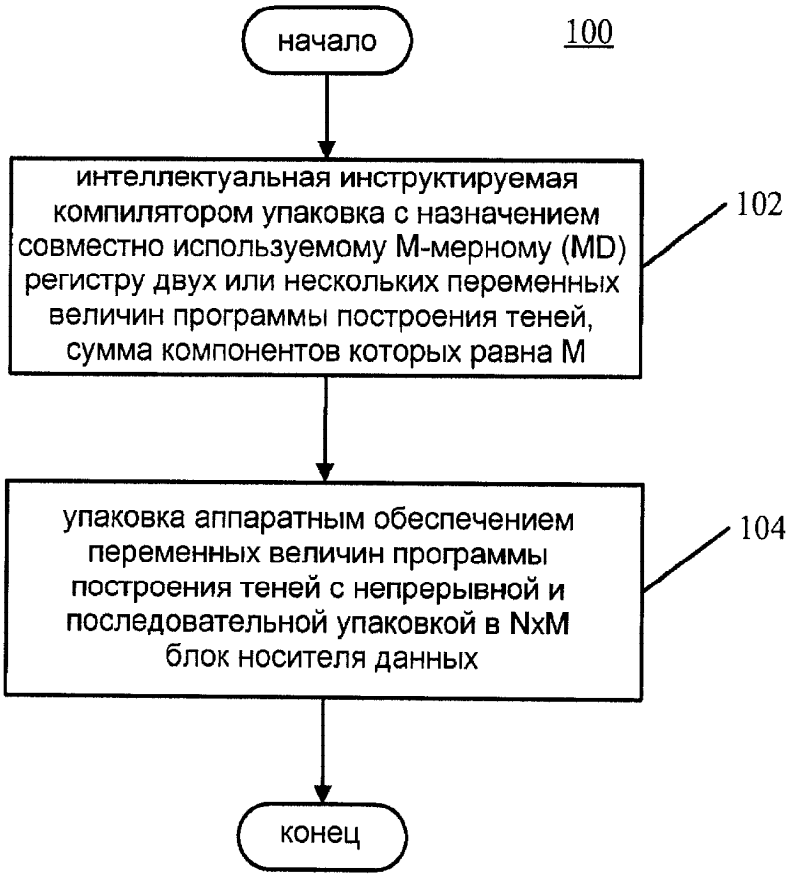
ФИГ. 3


кэш вершин
54
ассемблер примитивов и растеризатор
90
буфер переменной
92
блок связывания
82
команды связывания
загрузка и перераспределение переменной
84
88
FS входной регистровый файл
79
программа построения теней фрагментов
70

ФИГ. 5


61
драйвер
62
компилятор
80
компоновщик
64
таблица входных символов VS
66
таблица выходных символов VS
74
таблица входных символов FS
76
FS выход
86
таблица связывания

ФИГ. 6



ФИГ. 7

```
VERTEXSHADER_FUNCTI
ON_CREATE
“VSF12”
{
vs.1.1
dcl_position0 v0
dcl_texcoord0 v1   ◄L1—
dcl_color0 v2      ◄L2—
dp4 oPos.x, c0, v0
dp4 oPos.y, c1, v0
dp4 r0.z, c2, v0
mad r1.w, c29.x, -r0.z, c29.y
dp4 r0.w, c3, v0
max r1.w, r1.w,c28.x
add r0.w, r0.w, c4.x
min oFog, r1.w, c28.w
mov oPos.zw, r0
mov oT0.xy, v1     ◄L3—
mov oD0, v2        ◄L4—
}
```

ФИГ. 8A

```
VERTEXSHADER_FUNCTI
ON_CREATE
“VSF12”
{
vs.1.1
dcl_position0 v0


dp4 oPos.x, c0, v0
dp4 oPos.y, c1, v0
dp4 r0.z, c2, v0
mad r1.w, c29.x, -r0.z, c29.y
dp4 r0.w, c3, v0
max r1.w, r1.w,c28.x
add r0.w, r0.w, c4.x
min oFog, r1.w, c28.w
mov oPos.zw, r0


}
```

ФИГ. 8В

```
VERTEXSHADER_FUNCTION
_CREATE
“VSF14”
{
vs.1.1
def c4, 1,0, 0.0, 0.0, 0.0
dcl_position0 v0
dcl_texcoord0 v1
dcl_texcoord1 v2
dcl_texcoord2 v3
dcl_color0 v4
dcl_color1 v5
mad r0. v2.xyzx, c4.xxxy,
c4,yyyx
mul r1.xyz, v2.w, c0
dp4 r1.w, r0, c0
dp4 oPos.x, r1, v0
dp4 r2.w, r0, c2
mul r2.xyz, v2.w, c2
mul r1.xyz, v2.w, c1
dp4 r2.z, r2, v0
dp4  r1.w, r0, c1
mad r2.w, c29.x, -r2.z, c29.y
dp4 oPos.y, r1, v0
max r1.w, r2.w, c28.x
dp4 r0.w, r0, c3
min r1.x, r1.w, c28.w
mov oFog, r1.x
mul r0.xyz, v2.w, c3
mov oD1, r1.x
dp4 r2.w, r0, v0
mov oPos.zw, r2
mov oT0.xy, v1      ◄L5—
mov oT1.xy, v1      ◄L6—
mov oT2, v3         ◄L7—
mov oD0, v4         ◄L8—
mov oD1.xyz, v5     ◄L9—
}
```

ФИГ. 9A

```
VERTEXSHADER_FUNCTION
_CREATE
“VSF14”
{
vs.1.1
def c4, 1,0, 0.0, 0.0, 0.0
dcl_position0 v0
dcl_texcoord0 v1
dcl_texcoord1 v2
dcl_texcoord2 v3
dcl_color0 v4
dcl_color1 v5
mad r0. v2.xyzx, c4.xxxy,
c4,yyyx
mul r1.xyz, v2.w, c0
dp4 r1.w, r0, c0
dp4 oPos.x, r1, v0
dp4 r2.w, r0, c2
mul r2.xyz, v2.w, c2
mul r1.xyz, v2.w, c1
dp4 r2.z, r2, v0
dp4  r1.w, r0, c1
mad r2.w, c29.x, -r2.z, c29.y
dp4 oPos.y, r1, v0
max r1.w, r2.w, c28.x
dp4 r0.w, r0, c3
min r1.x, r1.w, c28.w
mov oFog, r1.x
mul r0.xyz, v2.w, c3
mov oD1, r1.x
dp4 r2.w, r0, v0
mov oPos.zw, r2




}
```

ФИГ. 9В

```
VERTEXSHADER_FUNCTION
_CREATE
"VSF17"
{
vs.1.1
dcl_position0 v0
dp4 oPos.x, c0, v0
dp4 oPos.y, c1, v0
dp4 oPos.z, c2, v0
dp4 oPos.w, c3, v0
mov oFog, v0.w      ◄L10─
mov oT0.xy.v0.w     ◄L11─


}
```

ФИГ. 10A

```
VERTEXSHADER_FUNCTION
_CREATE
“VSF17”
{
vs.1.1
dcl_position0 v0
dp4 oPos.x, c0, v0
dp4 oPos.y, c1, v0
dp4 oPos.z, c2, v0
dp4 oPos.w, c3, v0




}
```

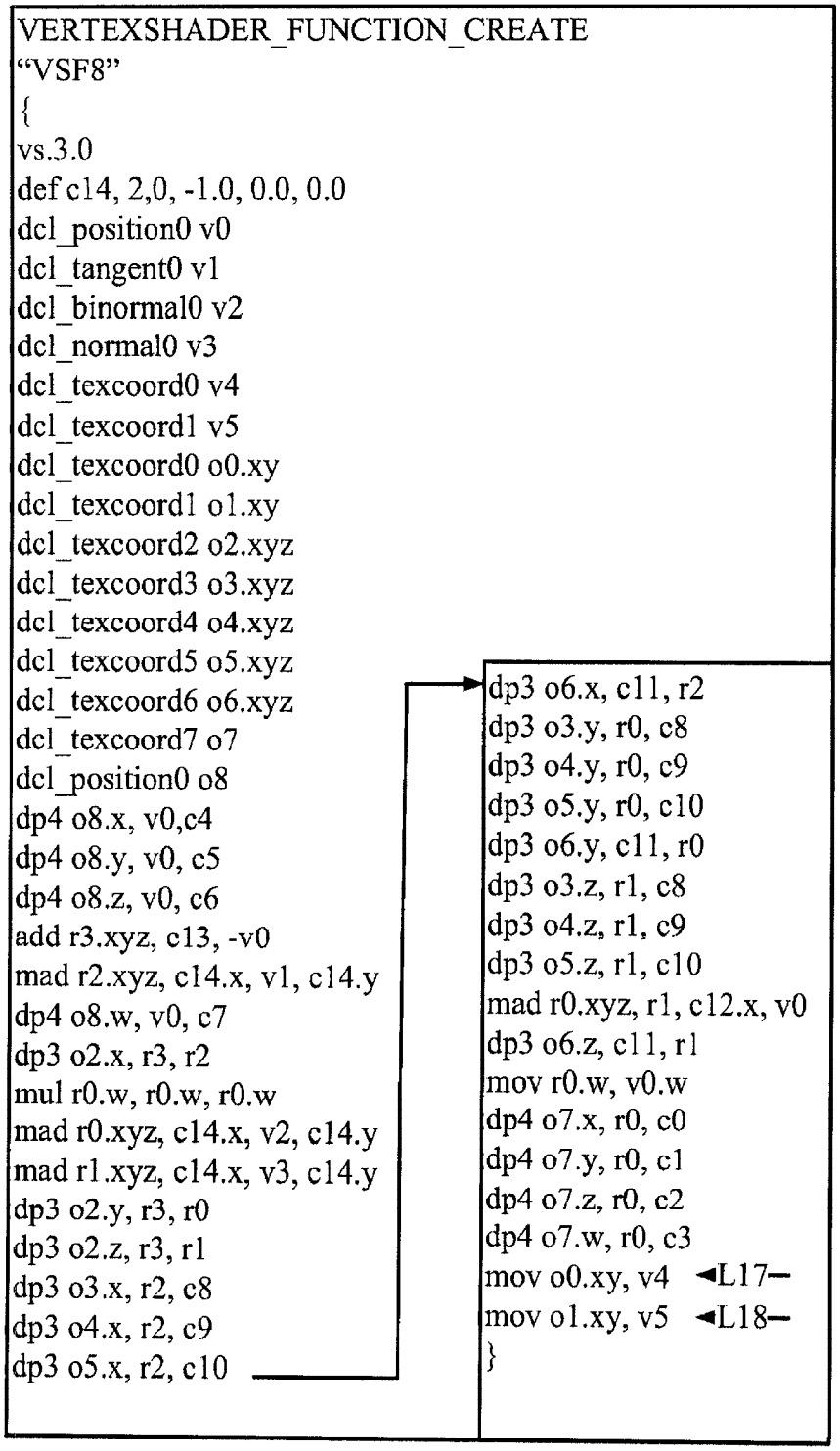
ФИГ. 10B

```
VERTEXSHADER_FUNCTION
_CREATE
“VSF25”
{
vs.1.1
def c4, 0,0, 0.0, 0.0, 0.0
dcl_position0 v0
dcl_texcoord0 v1
dcl_color0 v2
dcl_color1 v3
max r0.w, v0.z, c4.w
mad r0.w, r0.w, c7.z, c7.w
mul r0.w, r0.w, r0.w
mad r0.w, r0.w, r0.w, -c7.w
mad r0.xy, c7, r0.w, v0
mov r0.z, v0.z
dp3 r2.x, r0, r0
dp3 r1.x, v0, v0
rsq r1.w. r2.x
rsq r0.w. r1.x
mul r0.xyz, r0, r1.w
rcp r0.w, r0.w
mul r0.xyz, r0, r0.w
mov r0.w, v0.w
dp4 oPos.x, c0, r0
mad r1.w, c29.x, -r1.z, c29.y
dp4 oPos.y, c1, r0
max r2.w, r1.w, c28.x
dp4 r1.w, c3, r0
min r0.x, r2.w, c28.w
mov oPos.zw, r1
mov oFog, r0.x
mov oD1.w, r0.x
mov oT0.xy, v1        ◄L12—
mov oT1.xy, v1        ◄L13—
mov oT2, c4.x         ◄L14—
mov oD0, v2           ◄L15—
mov oD1.xyz, v3       ◄L16—
}
```

ФИГ. 11A

```
VERTEXSHADER_FUNCTION
_CREATE
“VSF25”
{
vs.1.1
def c4, 0,0, 0.0, 0.0, 0.0
dcl_position0 v0
dcl_texcoord0 v1
dcl_color0 v2
dcl_color1 v3
max r0.w, v0.z, c4.w
mad r0.w, r0.w, c7.z, c7.w
mul r0.w, r0.w, r0.w
mad r0.w, r0.w, r0.w, -c7.w
mad r0.xy, c7, r0.w, v0
mov r0.z, v0.z
dp3 r2.x, r0, r0
dp3 r1.x, v0, v0
rsq r1.w. r2.x
rsq r0.w. r1.x
mul r0.xyz, r0, r1.w
rcp r0.w, r0.w
mul r0.xyz, r0, r0.w
mov r0.w, v0.w
dp4 oPos.x, c0, r0
mad r1.w, c29.x, -r1.z, c29.y
dp4 oPos.y, c1, r0
max r2.w, r1.w, c28.x
dp4 r1.w, c3, r0
min r0.x, r2.w, c28.w
mov oPos.zw, r1
mov oFog, r0.x
mov oD1.w, r0.x




}
```

ФИГ. 11В

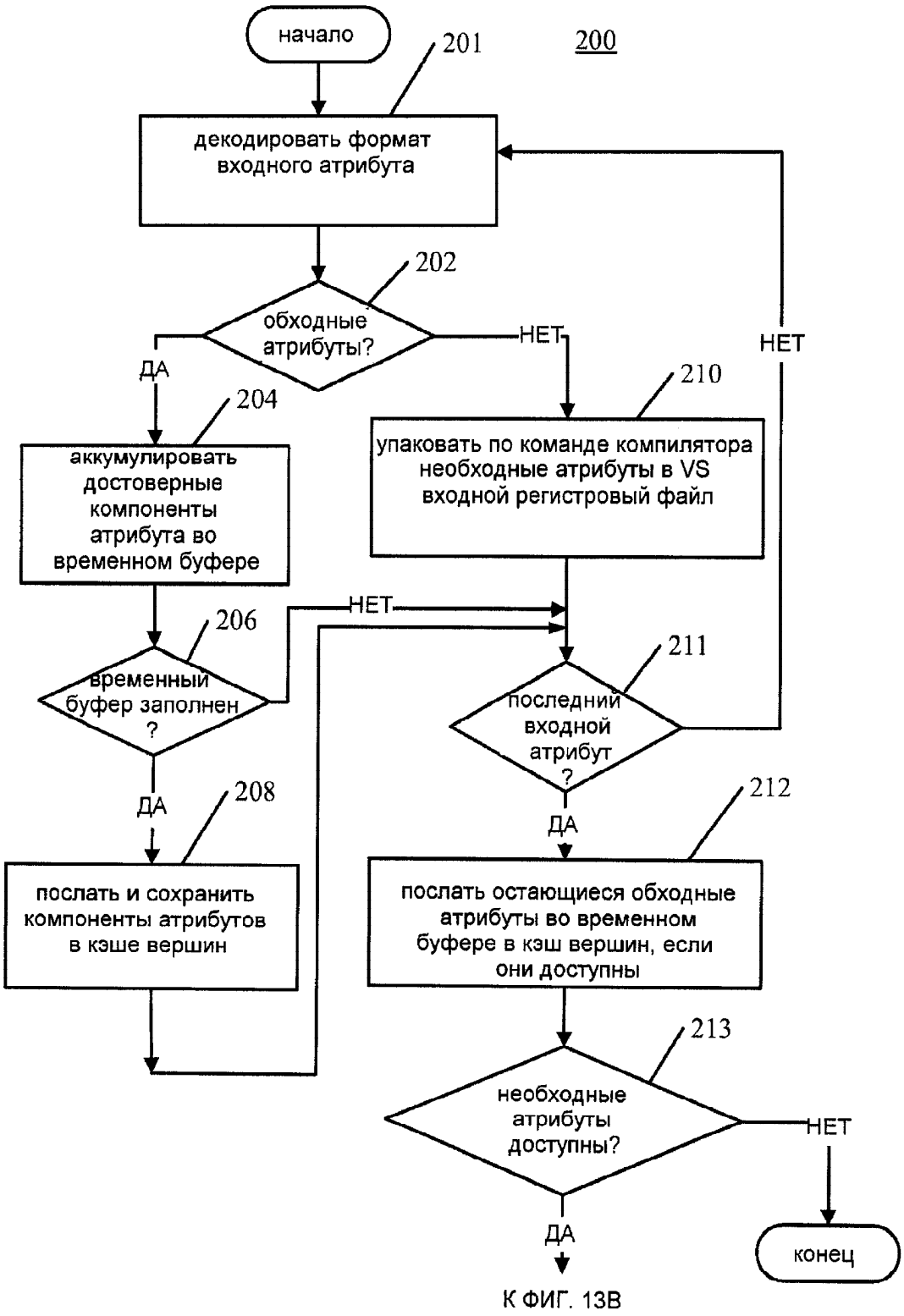

VERTEXSHADER_FUNCTION_CREATE
"VSF8"
{
vs.3.0
def c14, 2,0, -1.0, 0.0, 0.0
dcl_position0 v0
dcl_tangent0 v1
dcl_binormal0 v2
dcl_normal0 v3
dcl_texcoord0 v4
dcl_texcoord1 v5
dcl_texcoord0 o0.xy
dcl_texcoord1 o1.xy
dcl_texcoord2 o2.xyz
dcl_texcoord3 o3.xyz
dcl_texcoord4 o4.xyz
dcl_texcoord5 o5.xyz
dcl_texcoord6 o6.xyz
dcl_texcoord7 o7
dcl_position0 o8
dp4 o8.x, v0,c4
dp4 o8.y, v0, c5
dp4 o8.z, v0, c6
add r3.xyz, c13, -v0
mad r2.xyz, c14.x, v1, c14.y
dp4 o8.w, v0, c7
dp3 o2.x, r3, r2
mul r0.w, r0.w, r0.w
mad r0.xyz, c14.x, v2, c14.y
mad r1.xyz, c14.x, v3, c14.y
dp3 o2.y, r3, r0
dp3 o2.z, r3, r1
dp3 o3.x, r2, c8
dp3 o4.x, r2, c9
dp3 o5.x, r2, c10
dp3 o6.x, c11, r2
dp3 o3.y, r0, c8
dp3 o4.y, r0, c9
dp3 o5.y, r0, c10
dp3 o6.y, c11, r0
dp3 o3.z, r1, c8
dp3 o4.z, r1, c9
dp3 o5.z, r1, c10
mad r0.xyz, r1, c12.x, v0
dp3 o6.z, c11, r1
mov r0.w, v0.w
dp4 o7.x, r0, c0
dp4 o7.y, r0, c1
dp4 o7.z, r0, c2
dp4 o7.w, r0, c3
mov o0.xy, v4 ◄L17—
mov o1.xy, v5 ◄L18—
}

ФИГ. 12A

```
VERTEXSHADER_FUNCTION_CREATE
"VSF8"
{
vs.3.0
def c14, 2,0, -1.0, 0.0, 0.0
dcl_position0 v0
dcl_tangent0 v1
dcl_binormal0 v2
dcl_normal0 v3
dcl_texcoord0 v4
dcl_texcoord1 v5
dcl_texcoord0 o0.xy
dcl_texcoord1 o1.xy
dcl_texcoord2 o2.xyz
dcl_texcoord3 o3.xyz
dcl_texcoord4 o4.xyz
dcl_texcoord5 o5.xyz
dcl_texcoord6 o6.xyz
dcl_texcoord7 o7
dcl_position0 o8
dp4 o8.x, v0,c4
dp4 o8.y, v0, c5
dp4 o8.z, v0, c6
add r3.xyz, c13, -v0
mad r2.xyz, c14.x, v1, c14.y
dp4 o8.w, v0, c7
dp3 o2.x, r3, r2
mul r0.w, r0.w, r0.w
mad r0.xyz, c14.x, v2, c14.y
mad r1.xyz, c14.x, v3, c14.y
dp3 o2.y, r3, r0
dp3 o2.z, r3, r1
dp3 o3.x, r2, c8
dp3 o4.x, r2, c9
dp3 o5.x, r2, c10
```

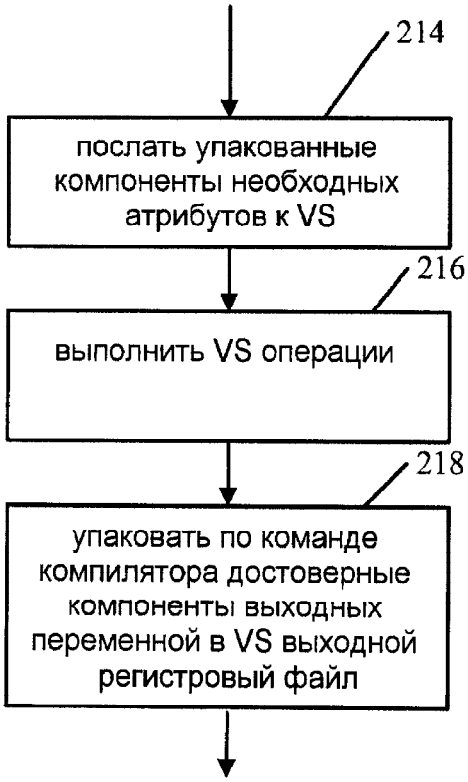
```
dp3 o6.x, c11, r2
dp3 o3.y, r0, c8
dp3 o4.y, r0, c9
dp3 o5.y, r0, c10
dp3 o6.y, c11, r0
dp3 o3.z, r1, c8
dp3 o4.z, r1, c9
dp3 o5.z, r1, c10
mad r0.xyz, r1, c12.x, v0
dp3 o6.z, c11, r1
mov r0.w, v0.w
dp4 o7.x, r0, c0
dp4 o7.y, r0, c1
dp4 o7.z, r0, c2
dp4 o7.w, r0, c3


}
```

ФИГ. 12В


начало
201
200
декодировать формат входного атрибута
202
обходные атрибуты?
НЕТ
ДА
НЕТ
210
204
упаковать по команде компилятора необходимые атрибуты в VS входной регистровый файл
аккумулировать достоверные компоненты атрибута во временном буфере
НЕТ
206
211
временный буфер заполнен ?
последний входной атрибут ?
ДА
208
ДА
212
послать и сохранить компоненты атрибутов в кэше вершин
послать остающиеся обходные атрибуты во временном буфере в кэш вершин, если они доступны
213
необходимые атрибуты доступны?
НЕТ
ДА
конец
К ФИГ. 13В

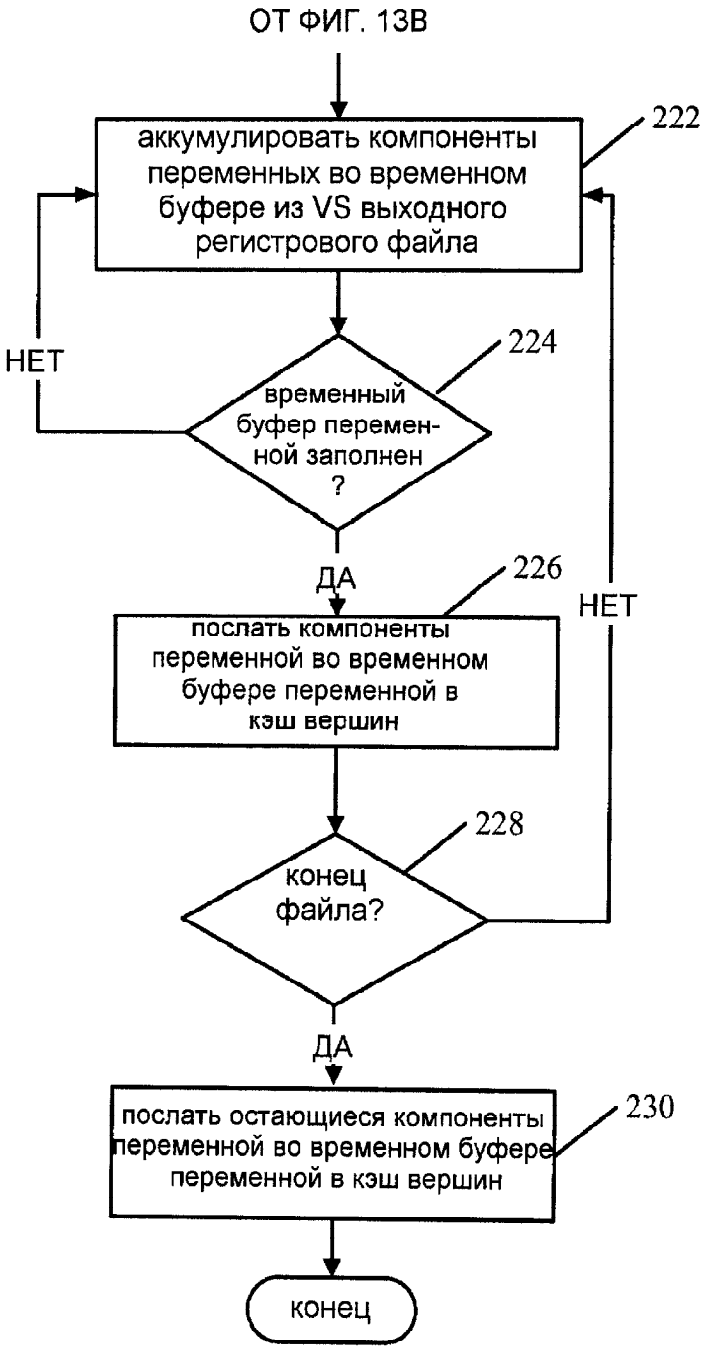
ФИГ. 13А

ОТ ФИГ. 13А

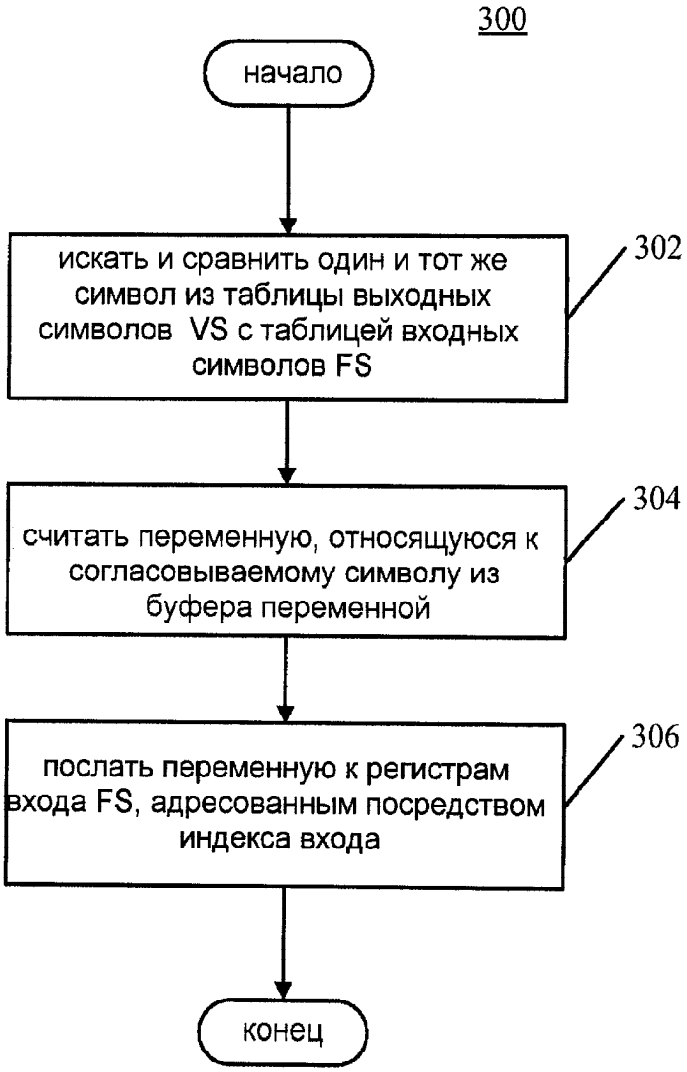


К ФИГ. 13С

ФИГ. 13В


ОТ ФИГ. 13B
222
аккумулировать компоненты переменных во временном буфере из VS выходного регистрового файла
224
временный буфер перемен-ной заполнен ?
НЕТ
ДА
226
послать компоненты переменной во временном буфере переменной в кэш вершин
НЕТ
228
конец файла?
ДА
230
послать остающиеся компоненты переменной во временном буфере переменной в кэш вершин
конец

ФИГ. 13C


300
начало
искать и сравнить один и тот же символ из таблицы выходных символов VS с таблицей входных символов FS
302
считать переменную, относящуюся к согласовываемому символу из буфера переменной
304
послать переменную к регистрам входа FS, адресованным посредством индекса входа
306
конец

ФИГ. 14