Содержание

Предисловие

Прошло пять лет с тех пор, как версия этой книги для OpenGL ES 2.0 сообщила всем разработчикам, что программируемая графика для мобильных и встраиваемых систем не просто появилась, но и собирается остаться.

Пятью годами ранее более *1 миллиарда* человек каждый день использовали OpenGL ES для взаимодействия со своими вычислительными устройствами для получения информации и развлечения. Практически каждый пиксел на практически каждом экране смартфона был создан, обработан и наложен с применением этого графического API.

Теперь OpenGL ES 3.0 был разработан Khronos Group и поставляется на современных мобильных устройствах, продолжая непрерывное движение продвинутой графики в руки пользователей повсюду – графики, которая была ранее разработана и опробована на мощных компьютерах, оснащенных поддержкой OpenGL.

На самом деле сейчас OpenGL является наиболее широко распространенной группой 3D API, включая как OpenGL для настольных систем и OpenGL ES, так и WebGL, дающий поддержку OpenGL ES для веба. OpenGL ES 3.0 поддержит дальнейшее развитие WebGL, позволяя разработчикам на HTML5 получить доступ ко всем возможностям современных GPU для создания действительно переносимых 3D-приложений.

OpenGL ES 3.0 не только предоставляет разработчикам больше возможностей для целого ряда устройств и платформ, но и позволяет писать более быстрые и энергоэффективные приложения, которые легче писать, переносить и поддерживать, – и эта книга покажет вам, как этого добиться.

Никогда не было более захватывающего и вознаграждающего времени для разработчика 3D-графики, чем сейчас. Я благодарю и поздравляю авторов за то, что они продолжают быть важной частью развивающейся истории OpenGL ES, и за тяжелую работу для создания этой книги, которая поможет разработчикам лучше понять и полностью задействовать всю силу OpenGL ES 3.0.

Нейл Треветт,
президент, Khronos Group,
вице-президент по мобильным системам, NVIDIA

Введение

OpenGL ES 3.0 – это программный интерфейс для рендеринга сложной 3D-графики для мобильных и встроенных устройств. OpenGL ES – это основная графическая библиотека для мобильных и встроенных устройств с программируемой 3D-графикой, включая мобильные телефоны, PDA, консоли, транспортные средства и т. п. Эта книга разбирает весь OpenGL ES 3.0 API и конвейер, в том числе детальные примеры, для предоставления руководства по разработке широкого спектра высокопроизводительных 3D-приложений для мобильных устройств.

Целевая аудитория

Эта книга ориентирована на программистов, которые хотели бы изучить OpenGL ES 3.0. Мы рассчитываем, что читатель хорошо знаком с компьютерной графикой. В книге мы объясняем многие из важных понятий, относящихся к различным частям OpenGL ES 3.0, но мы ожидаем, что читатель хорошо знаком с основными понятиями 3D. Все примеры кода в книге написаны на C. Мы рассчитываем, что читатель знаком с C или C++, и касаемся языка программирования только настолько, насколько это касается OpenGL ES 3.0.

Читатель узнает о настройке и программировании каждого аспекта графического конвейера. Книга детально разбирает написание вершинных и фрагментных шейдеров и реализацию продвинутых эффектов, таких как попиксельное освещение и системы частиц. Также предоставляются различные советы по оптимизации и эффективному использованию API и оборудования. После прочтения этой книги читатель будет готов писать приложения на OpenGL ES 3.0, полностью использующие силу программируемых графических процессоров для мобильных устройств.

Организация книги

Книга рассчитана на последовательный разбор API по мере продвижения.

Глава 1. Введение в OpenGL ES 3.0

Глава 1 является введением в OpenGL ES и предоставляет обзор графического конвейера OpenGL ES 3.0. Мы обсудим философию и ограничения, повлиявшие на разработку OpenGL ES 3.0. Также мы рассмотрим некоторые соглашения и типы, используемые в OpenGL ES 3.0.

Глава 2. Hello Triangle: пример на OpenGL ES 3.0

Глава 2 разбирает простой пример на OpenGL ES 3.0, рисующий треугольник. Нашей целью является показать, как выглядит программа на OpenGL ES 3.0, ознакомить читателя с некоторыми понятиями в API и описать, как построить и запустить программу на OpenGL ES 3.0.

Глава 3. Введение в EGL

Глава 3 рассматривает EGL – API для создания поверхностей и контекстов для OpenGL ES 3.0. Мы опишем взаимодействие с оконной системой, выбор конфигурации и создание контекстов и поверхностей EGL. Мы расскажем вам о EGL достаточно для того, чтобы вы могли сделать все, что вам понадобится для рендеринга при помощи OpenGL ES 3.0.

Глава 4. Шейдеры и программы

Шейдерные и программные объекты являются одними из самых важных объектов в OpenGL ES 3.0. В главе 4 мы опишем, как создать шейдерный объект, скомпилировать шейдер и проверить на наличие ошибок. Эта глава также объясняет, как создать программный объект, прикрепить к нему шейдерные объекты и выполнить линковку. Мы обсудим, как получать информацию из программного объекта и как задавать значения uniform-переменным. Также вы узнаете о разнице между шейдерами в виде исходного текста и бинарными шейдерами и как использовать оба этих типа.

Глава 5. Шейдерный язык OpenGL ES

Глава 5 объясняет основные понятия языка для написания шейдеров. Сюда входят переменные и типы, конструкторы, структуры, массивы, uniform-переменные, uniform-блоки и переменные для получения и выдачи значений. Эта глава также описывает некоторые тонкости языка для написания шейдеров, такие как задание точности и инвариантность.

Глава 6. Атрибуты вершины, вершинные массивы и объекты-буферы

Начиная с главы 6 (и заканчивая главой 11), мы начнем изучение конвейера, для того чтобы показать, как настроить и запрограммировать каждую часть графического конвейера. Мы начнем с описания того, как геометрические данные поступают на вход конвейера, и обсудим атрибуты вершины, вершинные массивы и объекты-буферы.

Глава 7. Сборка примитивов и растеризация

После обсуждения передачи геометрии в конвейер в предыдущей главе в главе 7 мы рассмотрим, как геометрия собирается в примитивы. Разбираются все типы примитивов, поддерживаемые OpenGL ES 3.0, включая точечные спрайты, треугольники, полосы и вееры из треугольников. Также мы рассмотрим, как выполняются преобразования координат над вершинами, и расскажем, как происходит растеризация в конвейере OpenGL ES.

Глава 8. Вершинные шейдеры

Следующая рассматриваемая часть конвейера – это вершинный шейдер. Глава 8 предоставляет обзор места вершинного шейдера в конвейере и специальных пере-

менных, доступных в вершинных шейдерах OpenGL ES. Приводятся примеры вершинных шейдеров, включая повершинное освещение и скиннинг. Также мы рассмотрим, как фиксированный конвейер OpenGL ES 1.0 (и 1.1) может быть реализован при помощи вершинных шейдеров.

Глава 9. Текстурирование

Глава 9 начинает рассмотрение фрагментных шейдеров с описания возможностей по работе с текстурами в OpenGL ES 3.0. Эта глава детально разбирает создание текстур, загрузку в них данных и использование их для рендеринга. Описываются преобразования текстурных координат, фильтрация, форматы текстур, сжатые текстуры, объекты-сэмплеры, неизменяемые текстуры, буферы и пирамидальное фильтрование. Эта глава рассматривает все типы текстур, поддерживаемые в OpenGL ES 3.0: двухмерные текстуры, кубические текстурные карты, массивы двухмерных текстур и трехмерные текстуры.

Глава 10. Фрагментные шейдеры

Глава 9 рассматривает использование текстур во фрагментных шейдерах; глава 10 рассматривает все остальное, что вам нужно знать для написания фрагментных шейдеров. Мы даем обзор фрагментных шейдеров и всех специальных встроенных переменных, доступных им. Мы также покажем, как реализовать весь фиксированный конвейер из OpenGL ES 1.1 при помощи фрагментных шейдеров. Приводятся примеры мультитекстурирования, тумана, альфа-теста и задаваемых пользователем плоскостей отсечения.

Глава 11. Операции с фрагментами

Глава 11 рассматривает операции, которые могут быть применены либо ко всему фреймбуферу, либо к отдельным фрагментам после выполнения фрагментного шейдера в конвейере рендеринга OpenGL ES 3.0. Эти операции включают в себя тест на попадание в заданный прямоугольник, тест трафарета, тест глубины, мультисэмплинг, смешивание и растрирование. Эта глава завершает рассмотрение конвейера рендеринга.

Глава 12. Объекты фреймбуфера

Глава 12 рассматривает использование объектов фреймбуфера для рендеринга в невидимые поверхности. Данные объекты могут быть использованы для нескольких задач, наиболее распространенной из них является рендеринг в текстуру. Эта глава дает полный обзор части API, связанного с фреймбуферами. Понимание объектов фреймбуфера важно для реализации ряда продвинутых эффектов, таких как отражения, теневые карты и постобработка.

Глава 13. Объекты синхронизации и барьеры

Глава 13 предоставляет обзор объектов синхронизации и барьеров, которые являются эффективными примитивами для синхронизации с приложением и GPU

в OpenGL ES 3.0. Мы рассмотрим, как использовать объекты синхронизации и барьеры, и завершим рассмотрение примером.

Глава 14. Продвинутое программирование с OpenGL ES 3.0

Глава 14 является ключевой в том смысле, что она связывает вместе многие темы, рассматриваемые на протяжении всей книги. Мы выбрали набор продвинутых методов и приведем примеры, демонстрирующие, как реализовать эти методы. Глава включает в себя такие методы, как попиксельное освещение с использованием карт нормалей, отражение окружающей среды, системы части, постобработка изображений, процедурные текстуры, теневые карты, рендеринг ландшафта и проективное текстурирование.

Глава 15. Получение состояния

В OpenGL ES 3.0 поддерживается большое число запросов состояния. Практически для всего, что можно установить, есть соответствующий способ получения этого значения. Глава 15 предоставляет справочник по различным запросам состояния, доступным в OpenGL ES 3.0.

Глава 16. Платформы OpenGL ES

В последней главе мы перейдем от деталей API к разговорам о том, как собрать примеры для OpenGL ES на iOS 7, Android 4.3 NDK, Android 4.3 SDK, Windows и Linux. Эта глава служит как справочник по тому, как собрать примеры по OpenGL ES 3.0 на вашей платформе.

Приложение А. GL_HALF_FLOAT_OES

Приложение А рассматривает детали формата для 16-битовых чисел с плавающей точкой и процесс перевода чисел между IEEEE-форматами для чисел с плавающей точкой и 16-битовым форматом с плавающей точкой.

Приложение Б. Встроенные функции

Приложение Б предоставляет информацию о всех встроенных функциях в шейдерном языке OpenGL ES.

Приложение В. Описание библиотеки, использованной в данной книге

В приложении В приводится обзор библиотеки, которая была разработана авторами книги и объясняет, что делает каждая функция.

Справка OpenGL ES 3.0

Вставлена в середину книги и перепечатывается с разрешения Khronos Group. Включает в себя полный список всех функций OpenGL ES 3.0 вместе со всеми типами, операторами, описателями, встроенными величинами и функциями в шейдерном языке OpenGL ES 3.0.

Примеры кода и шейдеров

В этой книге содержится много примеров кода и шейдеров. Вы можете скачать все примеры с сайта книги opengles-book.com, предоставляющего ссылку на проекты на github с кодом из книги. Все примеры были собраны и проверены на iOS 7, Android 4.3 NDK, Android 4.3 SDK, Windows (в режиме эмуляции OpenGL ES 3.0) и Ubuntu Linux. Некоторые из примеров были реализованы в PVRShaman, инструменте для разработки шейдеров от PowerVR, доступном для Linux, Windows и Mac OS X. Сайт книги предоставляет для скачивания все требуемые инструменты.

Ошибки

Если вы найдете что-то в книге, что, по вашему мнению, является ошибкой, пожалуйста, пошлите нам сообщение по адресу **errors@opengles-book.com**. Список найденных ошибок можно найти на сайте книги opengles-book.com.

Благодарности

Я хочу поблагодарить Эффи Мунши и Дэйва Шрейнера за их огромный вклад в первое издание этой книги. Я также крайне признателен Буди Пурномо, присоединившемуся ко мне для обновления этиой книги для версии OpenGL ES 3.0. Еще я хотел бы поблагодарить многих коллег, с которыми я работал много лет вместе, кто помог мне в изучении компьютерной графики, OpenGL и OpenGL ES. Таких людей слишком много, чтобы указать их всех, но особенно я бы хотел поблагодарить Шона Лифа, Билла Лиси-Кэйна, Мориса Риббла, Бенжа Липчака, Роджера Дешано, Дэвида Гусселина, Торстена Шурмана, Джона Исидоро, Криса Оата, Джейсона Митчела, Дана Гесселя и Эвана Харта.

Я также хотел бы выразить особую благодарность моей жене Софии за ее поддержку во время моей работы над книгой. Еще я хотел бы поблагодарить моего сына Этана, который родился во время работы над этой книгой. Твоя улыбка и смех приносят мне радость каждый день.

– Ден Гинсбург

Я хотел бы выразить глубокую благодарность Дэну Гинсбургу за предоставленную мне возможность работы над этой книгой. Благодарю моего менеджера Кэлана МкИналли и коллег из AMD за поддержку. Также я хотел бы поблагодарить моих преподавателей Джонатана Кохена, Субодха Кумара, Чинг-Куанг Шена и Джона Лоусера за приведение меня в мир компьютерной графики и OpenGL.

Я хотел бы поблагодарить моих родителей и сестру за их любовь. Особые благодарности моей жене Лиане Хади, чья любовь и поддержка позволили мне закончить этот проект. Благодарю моих дочерей Мишель Ло и Скарлетт Ло. Они свет в моей жизни.

– Буди Пурномо

Мы все хотим поблагодарить Нейла Треветта за написание предисловия и получение одобрения от Khronos Board Of Promoters за разрешение использовать текст из описания шейдерного языка OpenGL ES в приложении Б и справки по OpenGL ES 3.0. Особая благодарность тем, кто ознакомился с книгой и дал свои замечания: Морису Риббл, Питеру Лорманну и Эммануэлю Агу. Мы также хотим поблагодарить всех тех, кто дал свои замечания на первую редакцию книги: Брайана Коллинза, Криса Гримма, Джереми Сэндмела, Том Олсона и Адама Смита.

Мы хотим выразить огромную благодарность нашему редактору Лоре Левин из Addison-Wesley, оказавшей огромную помощь в каждом вопросе, связанном с книгой. Также в Addison-Wesley было много других, оказавших огромную помощь и которых мы хотели бы поблагодарить, включая Дебру Вильямс Коли, Оливию Баседжио, Шери Кэйн и Курта Джонсона.

Мы хотели бы поблагодарить читателей первого издания книги, оказавших огромную помощь, сообщая ошибки и улучшая код примеров. Мы хотели бы особенно поблагодарить нашего читателя Джаведа Раббани Шаха, кто портировал

примеры OpenGL ES 3.0 на Android 4.3 SDK на Java. Также он помог нам с Android NDK и рядом проблем, зависящих от конкретных устройств. Мы хотим поблагодарить Джарко Ватьюса-Антиллу за портирование на Linux X 11 и Эдуардо Пеллегри-Ллопарта и Дэррила Гофа за портирование кода из первого издания на BlackBerry Native SDK.

Большая благодарность OpenGL ARB, рабочей группе по OpenGL ES и каждому, кто внес свой вклад в разработку OpenGL ES.

Об авторах

Дэн Гинсбург

Ден – основатель Upsample Software, компании, предлагающей консультирование по 3D-графике и вычислениям на GPU. Дэн участвовал в написании ряда других книг, включая OpenCL Programming Guide и OpenGL Shading Language, 3rd Edition. Ранее он работал над написанием драйверов OpenGL, десктопных и мобильных приложений, средств для разработчиков GPU, трехмерной медицинской визуализацией и играми. Он бакалавр по информатике Ворчестерского политехнического института и магистр университета Бентли.

Будирижанто Пурномо

Буди – старший архитектор программных систем в AMD, где он возглавляет разработки по отладке и профилированию для GPU для различных платформ. Он сотрудничает со многими разработчиками программных и аппаратных средств в AMD для разработки архитектур для отладки и профилирования приложений на GPU. Он опубликовал много статей по компьютерной графике на международных конференциях. Он бакалавр и магистр Мичиганского технологического университета, защитил диссертацию в университете Джона Хопкинса.

Аафтаб Мунши

Аффи разрабатывал GPU больше десятка лет. В ATI (теперь AMD) он был ведущим архитектором по мобильным устройствам. Он участвовал в разработке спецификаций по OpenGL ES 1.1, OpenGL ES 2.0 и OpenCL. Теперь он работает в Apple.

Дэйв Шрейнер

Дэйв работал с OpenGL более двадцати лет и недавно работает и с OpenGL ES. Он разрабатывал первый тренинг по OpenGL во время работы в SGI и был автором OpenGL Programming Guide. Он проводил вводные и продвинутые курсы по OpenGL на различных международных конференциях, включая SIGGRAPH.

Сейчас он работает в ARM Inc. Он бакалавр по математике университета Дэлавэра.

Глава 1

Введение в OpenGL ES 3.0

OpenGL для встроенных систем (OpenGL ES) – это API для продвинутой 3D-графики, рассчитанный на мобильные и встроенные устройства. OpenGL ES – это основной графический API для современных смартфонов, применяется даже на настольных системах. Список платформ, поддерживающих OpenGL ES, включает в себя iOS, Android, BlackBerry, bada, Linux и Windows. OpenGL ES также лежит в основе WebGL – стандарта для трехмерной графики для веба.

Начиная с релиза iPhone 3GS в июне 2009-го и Android 2.0 в марте 2010-го, OpenGL ES 2.0 поддерживался на устройствах с iOS и Android. Первое издание этой книги детально рассматривало OpenGL ES 2.0. Нынешнее издание нацелено на OpenGL ES 3.0, следующую версию OpenGL ES. Практически неизбежно, что каждая развивающаяся мобильная платформа будет поддерживать OpenGL ES 3.0. Сейчас OpenGL ES 3.0 уже поддерживается на устройствах с Android 4.3 и старше и на iPhone 5S с iOS7. OpenGL ES 3.0 обратно совместим с OpenGL ES 2.0, под этим подразумевается, что приложения, написанные под OpenGL ES 2.0, будут работать и под OpenGL ES 3.0.

OpenGL ES – это один из многих API, созданных Khronos Group. Основанная в январе 2000 года, Khronos Group – это поддерживаемый своими членами консорциум, ориентированный на создание открытых стандартов и API, не требующих лицензирования. Khronos Group также занимается развитием OpenGL – кроссплатформенного API для трехмерной графики, ориентированного на настольные системы с Linux, различными вариантами UNIX, Mac OS X и Microsoft Windows. Это широко распространенный 3D API, имеющий большое влияние в мире.

В связи с широким распространением OpenGL как API для трехмерной графики его положили в основу при разработке открытого стандарта для 3D-графики для мобильных и встроенных устройств и затем изменили для того, чтобы соответствовать потребностям и ограничениям мобильных и встроенных устройств. В ранних версиях OpenGL ES (1.0, 1.1, 2.0) эти ограничения включали в себя ограниченные возможности по обработке данных, невысокую ширину шины для передачи данных и чувствительность к потреблению энергии. При определении спецификаций OpenGL ES рабочая группа использовала следующие критерии:

- OpenGL API очень большой и сложный, а целью рабочей группы по OpenGL ES является создание API, подходящего для устройств с ограниченными возможностями. Для достижения этой цели рабочая группа убрала всю избыточность из OpenGL API. В случае, когда одна и та же операция может быть выполнена более, чем одним способом, оставлялся наиболее полезный способ, остальные просто удалялись. Хорошим примером этого является за-

дание геометрии, когда в OpenGL приложение может использовать непосредственный режим, дисплейные списки и вершинные массивы. В OpenGL ES существуют только вершинные массивы; непосредственный режим и дисплейные списки были удалены.

- Удаление избыточности является важной целью, но также важным было сохранение совместимости с OpenGL. По возможности, OpenGL ES был разработан так, что приложения, написанные на подмножестве OpenGL для встроенных систем, также будут работать на OpenGL ES. Это было важной целью, поскольку позволило бы разработчикам использовать сразу оба API и разрабатывать приложения и инструменты, применяющие общую функциональность.
- Новые возможности были добавлены для учета специфических ограничений мобильных и встроенных устройств. Например, для уменьшения потребления энергии и увеличения быстродействия шейдеров в язык для написания шейдеров были добавлены спецификаторы точности.
- Разработчики OpenGL ES хотели гарантировать минимальный набор возможностей для обеспечения качества получаемых изображений. В ранних мобильных устройствах размеры экрана были довольно небольшими, поэтому было важно, чтобы качество каждого выводимого пиксела было настолько хорошим, насколько это возможно.
- Рабочая группа по OpenGL ES хотела гарантировать, что любая реализация OpenGL ES будет удовлетворять определенным соглашениям по качеству изображения, корректности и устойчивости. Для этого были разработаны соответствующие тесты, которые должны быть выполнены для того, чтобы реализация OpenGL ES была признана совместимой.

На данный момент времени Khronos выпустил четыре спецификации OpenGL ES: OpenGL ES 1.0 и OpenGL ES 1.1 (в книге мы будем обозначать их как OpenGL ES 1.x), OpenGL ES 2.0 и OpenGL ES 3.0. Спецификации OpenGL ES 1.0 и 1.1 реализуют фиксированный конвейер рендеринга и основаны, соответственно, на спецификациях OpenGL 1.3 и OpenGL 1.5.

Спецификации OpenGL ES 2.0 вводят программируемый конвейер и основаны на спецификациях OpenGL 2.0. Это значит, что соответствующие спецификации для OpenGL были взяты за основу для определения того, что войдет в соответствующую версию OpenGL ES.

OpenGL ES 3.0 – это следующий шаг в эволюции мобильной графики, он основан на спецификациях OpenGL 3.3. В то время когда OpenGL ES 2.0 был успешен в привнесении на мобильные устройства возможностей, похожих на DirectX 9 и Microsoft Xbox 360, на настольных системах графика продолжала развиваться. В OpenGL ES 2.0 не хватало многих важных возможностей, необходимых для реализации таких эффектов, как теневые карты, рендеринг объемных фигур (volume rendering), выполняемая на GPU анимация систем частиц, instancing, сжатие текстур и гамма-коррекция. OpenGL ES 3.0 добавляет эти возможности для мобильных устройств, продолжая при этом философию адаптации к ограничениям мобильных устройств.

Конечно, некоторые из этих ограничений, учтенные при разработке предыдущих версий OpenGL ES, были уже не актуальны. Например, на мобильных устройствах сейчас экраны с большим размером (иногда их разрешение даже выше, чем на настольных системах). Кроме того, на многих мобильных устройствах сейчас есть многоядерные центральные процессоры и большой объем памяти. Поэтому при разработке OpenGL ES 3.0 целью Khronos стал, скорее, своевременный вывод на рынок новых возможностей, а не ограниченные возможности устройств.

Следующие разделы посвящены конвейеру OpenGL ES.

OpenGL ES 3.0

Как уже отмечалось, эта книга посвящена OpenGL ES 3.0 API. Нашей целью является детально разобрать спецификации OpenGL ES 3.0, дать примеры использования OpenGL ES 3.0 и обсудить различные способы оптимизации. После прочтения этой книги у вас будет отличное понимание OpenGL ES 3.0 API, вы сможете легко писать сложные приложения, использующие OpenGL ES 3.0, и вам не придется изучать многочисленные спецификации, для того чтобы понять, как та или иная возможность работает.

OpenGL ES 3.0 реализует программируемый графический конвейер и состоит из двух частей – описания OpenGL ES 3.0 API и описания языка шейдеров для OpenGL ES 3.0. На рис. 1.1 приведен графический конвейер OpenGL ES 3.0. Закрашенные прямоугольники на этом рисунке обозначают программируемые части конвейера в OpenGL ES 3.0. Далее приводится обзор каждой из частей конвейера.

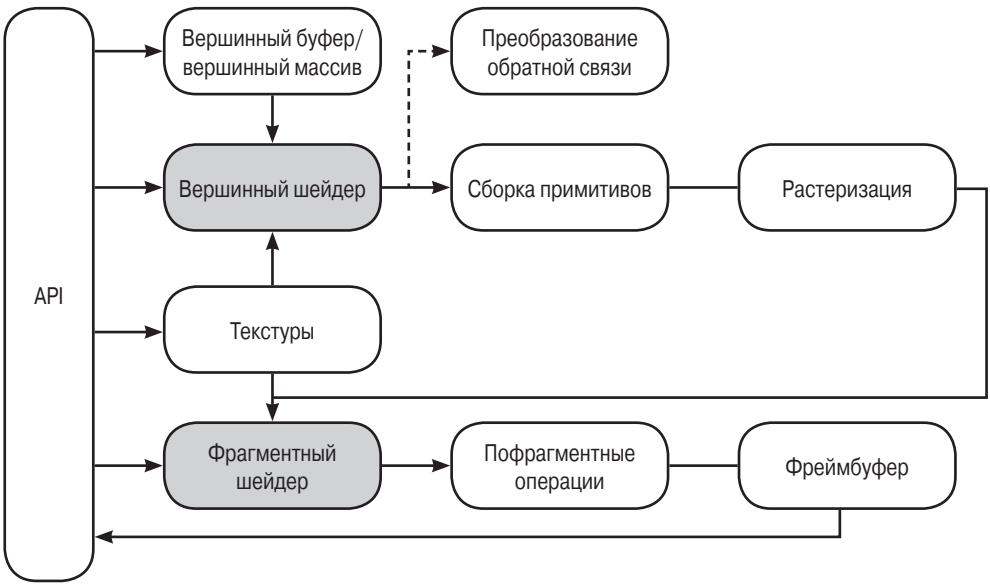

Рис. 1.1 ❖ Графический конвейер OpenGL ES 3.0

Вершинный шейдер

Этот раздел дает обзор вершинных шейдеров. Детально вершинные и фрагментные шейдеры будут рассмотрены в дальнейших главах книги. Вершинный шейдер реализует программируемый подход для работы с вершинами.

Входными данными для вершинного шейдера являются:

- исходный код шейдера в виде теста или в бинарном виде, задающий действия, которые будут выполняться над вершиной;
- входные значения вершины (атрибуты) – данные, передаваемые для каждой вершины при помощи вершинных массивов;
- Uniform-переменные – значения, используемые вершинными и фрагментными шейдерами;
- сэмплеры – особый тип uniform-переменных, служащий для представления текстур, используемых вершинным шейдером.

Выходные значения вершинного шейдера назывались в OpenGL ES 2.0 varying-переменными, но в OpenGL ES 3.0 были переименованы в выходные переменные. На стадии растеризации примитивов выходные значения вершинного шейдера вычисляются для каждого полученного фрагмента и передаются как входные значения во фрагментный шейдер. Механизм, используемый для получения значений для каждого фрагмента из выходных значений вершинного шейдера для вершин, называется интерполяцией. Также OpenGL ES 3.0 добавляет новую возможность, называемую преобразованием обратной связи (transform feedback), которая позволяет записать выходные значения вершинного шейдера в буфер (в дополнение или вместо обработки фрагментным шейдером). Например, как показано в примере по преобразованию обратной связи в главе 14, система частиц может быть реализована при помощи вершинного шейдера, в котором частицы выводятся в буферный объект при помощи преобразования обратной связи. Входные и выходные значения вершинного шейдера показаны на рис. 1.2.

Вершинные шейдеры могут быть использованы для традиционных операций над вершинами, так как преобразование координат при помощи матриц, вычисление при помощи уравнения освещенности цвета в вершине, генерирование или преобразование текстурных координат. Кроме этого, поскольку вершинный шейдер задается в приложении, вершинные шейдеры могут быть использованы для реализации нестандартных преобразований, освещения или повершинных эффектов, недоступных в традиционных фиксированных конвейерах.

Пример 1.1 показывает вершинный шейдер, написанный с использованием языка шейдеров OpenGL ES. Мы детально объясним вершинные шейдеры позже. Мы приводим этот шейдер здесь для того, чтобы дать вам представление о том, как выглядит вершинный шейдер. Вершинный шейдер из примера 1.1 берет положение и связанный с ним цвет как входные атрибуты, преобразует координаты при помощи матрицы 4×4 и выводит преобразованные координаты и цвет.

Строка 1 задает версию шейдерного языка – информацию, которая должна быть в первой строке шейдера (`#version 300 es` задает использование шейдерного языка для OpenGL ES 3.0). Строка 2 описывает uniform-переменную `u_mvpMatrix`,

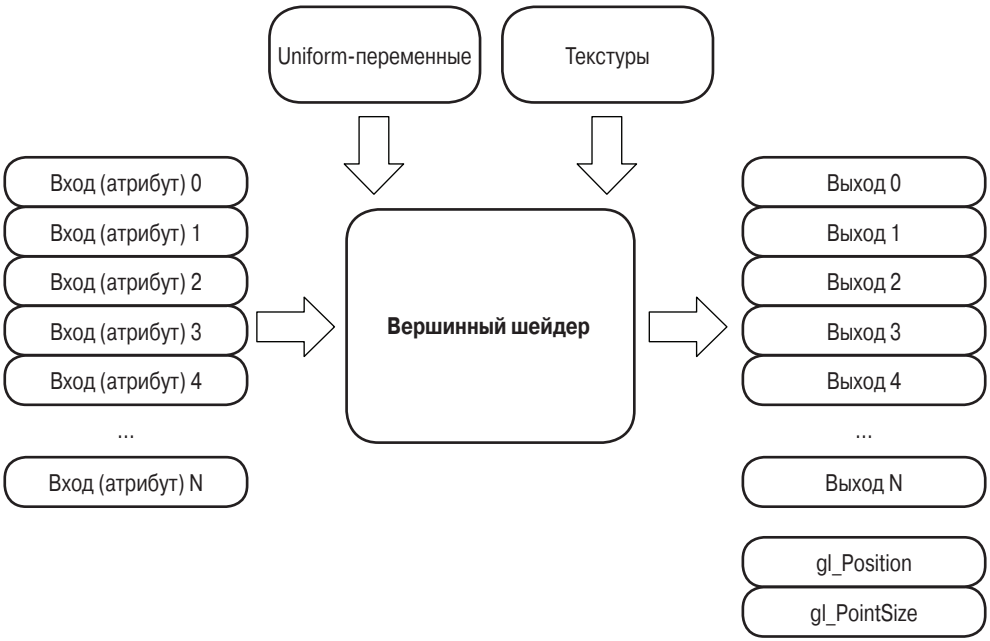

Рис. 1.2 ❖ Вершинный шейдер OpenGL ES

Пример 1.1 ❖ Пример вершинного шейдера

```
1.  #version 300 es
2.  uniform mat4 u_mvpMatrix; // matrix to convert a_position
3.                            // from model space to normalized
4.                            // device space
5.
6.  // attributes input to the vertex shader
7.  in vec4 a_position;       // position value
8.  in vec4 a_color;          // input vertex color
9.
10. // output of the vertex shader - input to fragment
11. // shader
12. out vec4 v_color;         // output vertex color
13. void main()
14. {
15.     v_color = a_color;
16.     gl_Position = u_mvpMatrix * a_position;
17. }
```

которая хранит в себе произведение модельно-видовой матрицы и матрицы проектирования. Строки 7 и 8 описывают входные значения для вершинного шейдера, называемые вершинными атрибутами. `a_position` – это атрибут, соответствую-

щий координатам вершины, а a_color – это атрибут, содержащий цвет вершины. В строке 12 мы описываем выходное значение v_color, в которое мы запишем цвет в вершине. Встроенная переменная gl_Position в объявлении не нуждается, и вершинный шейдер должен записать преобразованные координаты в эту переменную. У вершинного и фрагментного шейдеров есть всего одна точка входа – функция main. Строки 13–17 задают функцию main. В строке 15 мы читаем атрибут вершины из a_color и записываем его в выходную переменную v_color. В строке 16 преобразованные координаты вершины записываются в переменную gl_Position.

Сборка примитивов

После вершинного шейдера следующей стадией конвейера OpenGL ES 3.0 является сборка примитивов. Примитив – это геометрический объект, такой как треугольник, отрезок или точечный спрайт. Каждая вершина примитива посылается отдельной копии вершинного шейдера. Во время сборки примитива эти вершины группируются вместе для образования примитива.

Для каждого примитива необходимо определить, лежит ли он внутри области видимости (области трехмерного пространства, которая видна на экране). Если примитив не полностью находится внутри области видимости, то необходимо выполнить отсечение примитива по границе области видимости. Если примитив находится полностью вне области видимости, то он отбрасывается. После отсечения координаты вершин переводятся в координаты на экране. Также может быть произведено отбрасывание граней в зависимости от того, какой стороной они повернуты. После отсечения и отбрасывания примитив готов для передачи на следующую стадию конвейера – стадию растеризации.

Растеризация

Следующая стадия, показанная на рис. 1.3, – это стадия растеризации, на которой соответствующий примитив (точечный спрайт, отрезок или треугольник) выводится. Растеризация – это процесс, который переводит примитивы в набор двухмерных фрагментов, обрабатывающихся фрагментным шейдером. Эти двухмерные фрагменты представляют пикселы, которые могут быть нарисованы на экране.

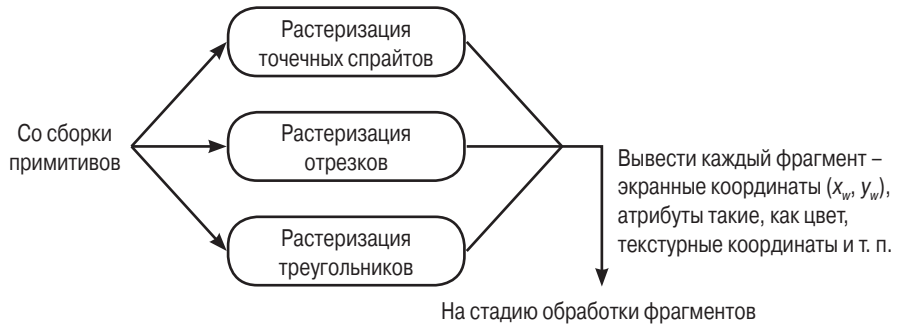

Рис. 1.3 ❖ Стадия растеризации OpenGL ES 3.0