

Архитектура настраиваемого конвейера рендеринга с общей памятью

Михаил Самохвалов (mike{a dog}ldesignstudio.ru)
Институт Системного Анализа РАН
Москва 2005г.

В данной статье рассматривается архитектура полностью настраиваемого графического конвейера с общей памятью, описываются его возможности и перспективы его реализации в софтверных рендерах и в системах реального времени (графические ускорители).

Введение

Процесс визуализации (рендеринг) состоит из большого числа операций, которые производятся над трехмерными данными, начиная от установки детализации объектов и заканчивая растеризацией треугольников. Все эти операции в сумме составляют графический конвейер (pipeline).

Для простоты принято разделять конвейер на стадии. Примером таких стадий могут служить растеризация примитивов, текстурирование, трансформация итд. Эти стадии выполняются в строго определенном порядке, тк являются сильно зависимыми друг от друга.

На данный момент практически все реализации конвейеров основываются на более-менее общей схеме «тесселяция-трансформация-освещение-растризация-фильтрация». Эта схема также реализована в специализированных графических процессорах. Она позволяет выполнять весь набор операций, необходимый для построения изображений. Но также она имеет и недостатки.

При разработке системы нефотореалистичного рендеринга выяснился довольно неприятный факт – для

правильной отрисовки перестало хватать возможностей стандартного конвейера. Было необходимо использовать более гибкую структуру, а именно – конвейер в виде графа. Обычно в системах рендеринга для сложных схем (например, сложные модели освещения или шейдера) используются несколько проходов для получения результата. Те сцена отрисовывается несколько раз с различными настройками, а потом полученные изображения используются для получения результирующей картинки. Но этого оказывается недостаточно, так как снижает скорость, а главное, зачастую и качество изображений. В моем случае сложность состояла в том, что нужно было считать видимость линий контура по z-buffer'у. Для этого было необходимо выстроить граф из стадий. Данные при этом должны были проходить обработку в строго определенной последовательности и с использованием результатов, полученных на нескольких предыдущих шагах. Многопроходностью этого достичь невозможно. Но об этом – ниже.

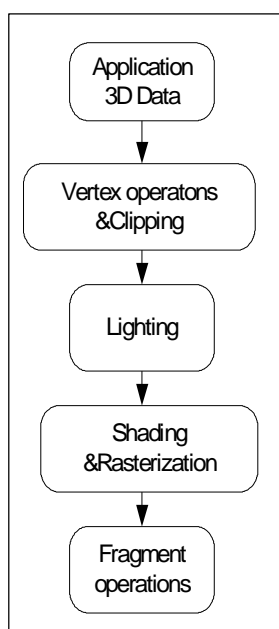
Конвейер рендеринга

Все современные системы визуализации делают очень простую на первый взгляд

вещь: мы имеем описание трехмерных предметов или геометрии, расположение источников света и, может быть, некую дополнительную информацию и получаем картинку на выходе. Однако отрисовывать 3х мерную геометрию можно разными путями. В настоящее время существует несколько вариантов конвейера. Мы рассмотрим наиболее известные.

Краткий обзор существующих решений

OpenGL



Конвейер OpenGL в упрощенном виде представлен на рисунке. Можно выделить несколько основных его стадий. Сначала данные приложения (как правило, треугольники) поступают на стадию векторной обработки. Эта стадия нужна для того, чтобы произвести преобразования (например, перевести

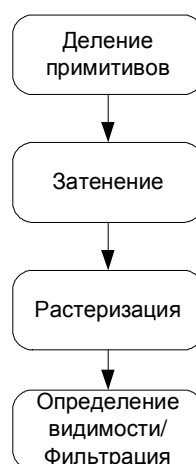
в экранную систему координат или отмасштабировать объект) над вертексами и произвести отсечение. Потом производится расчет повертексного освещения. Сейчас на современных процессорах эти 2 стадии объединены в одну и являются программируемыми (используются Vertex program). Процессоры, которые обрабатывают данные на этих двух стадиях, являются векторными и позволяют, например, за одну операцию перемножать вектора.

Следующая стадия – растеризация и текстурирование. На ней производится наложение текстур на треугольники и их вывод в буфер.

Далее идет стадия для операций с фрагментами. Для этой стадии также применяется специализированный язык (Fragment program).

В настоящее время для графики в реальном времени существует несколько специализированных языков для программирования 2х стадий конвейера (Vertex program и Fragment program), которые позволяют работать соответственно с вертексами, которые поступают в карту (позволяют оперировать как цветом, так и расположением вертексов, цветом тумана) и фрагментами изображения после растеризации (операции с текстурами). Для того, чтобы не только управлять этими стадиями, но и использовать дополнительную информацию о свойствах 3х мерных моделях, освещении и т. д., используется механизм регистров, через которые дополнительные данные могут поставляться на программируемые модули извне. Эти данные задаются программно.

Reyes



Архитектура Reyes была создана компаниями Lusacfilm и Pixar для высококачественного рендеринга сложных сцен. Конвейер Reyes имеет 4 основные стадии, которые можно видеть на рисунке.

Основным примитивом, используемым в данной системе, является микро-

полигон, однородно закрашенный четырехугольник. В оригинальной реализации он не может быть больше половины размера пикселя.

Стадии этой архитектуры:

Деление примитивов. Входными данными для этой архитектуры являются высокоуровневые примитивы, такие как поверхности высокого порядка, бикубические поверхности и т. д. Примитивы могут делиться до уровня микро-полигонов. Эта операция выполняется в пространстве камеры.

Затенение. Также выполняется в этом же пространстве используя процедурно заданные шейдера. Так как микро-полигоны имеют размер меньше пикселя, каждый из них имеет плоскую закраску.

Растеризация. Каждый микро-полигон проецируется в экранное пространство, проходит отсечение по области видимости и растеризуется. Архитектура Reyes использует стохастический метод для определения цвета пикселя по попавшим в него нескольким микро-полигонам.

Определение видимости\Фильтрация. Определение видимости происходит, используя стандартный Z-буфер. Затем происходит фильтрация sub-пикселей для получения конечного изображения.

Архитектура Reyes отличается от OpenGL в трех местах: пространстве, в котором выполняется затенение, характеристиках доступа к текстурам и методе растеризации. Затенение в OpenGL выполняется в 2х местах: при повертексной обработке (в пространстве камеры, обычно при просчете освещения) и в операциях с фрагментами (в экранных координатах). В архитектуре Reyes выполняется только одна стадия затенения при обработке микро-полигонов в системе координат камеры.

В OpenGL текстурирование является пофрагментной операцией, которая выполняется в экранной системе координат. Доступ к текстурам при этом является некогерентным и для устранения артефактов в изображениях применяется mip-mapping. В Reyes же доступ когерентен и дополнительная фильтрация не нужна. Основным примитивом для растеризации в OpenGL – треугольник. Для Reyes – микро-полигон.

DirectX Next и шейдера версии 4

Архитектура следующей версии DirectX пока еще не определена. Можно только догадываться о тех возможностях, которые она принесет.

Предполагается, что это будет та же линейная архитектура конвейера. Однако туда будут, вероятно, внесены такие возможности как:

Новые шейдера, которые смогут генерировать вершины в вершинном шейдере. Таким образом, у пользователя появится возможность программировать еще одну стадию.

Произвольный доступ к данным из шейдеров в виртуальном адресном пространстве ускорителя. Также будут введены дополнительные команды в язык шейдеров для адресации.

Так как архитектура графического процессора не будет сильно меняться, то для большей гибкости и ускорения обработки данных будет увеличиваться количество процессоров, которые занимаются обработкой данных на отдельных стадиях (вершинные и фрагментные).

Подробнее о DXNext можно узнать в [3]

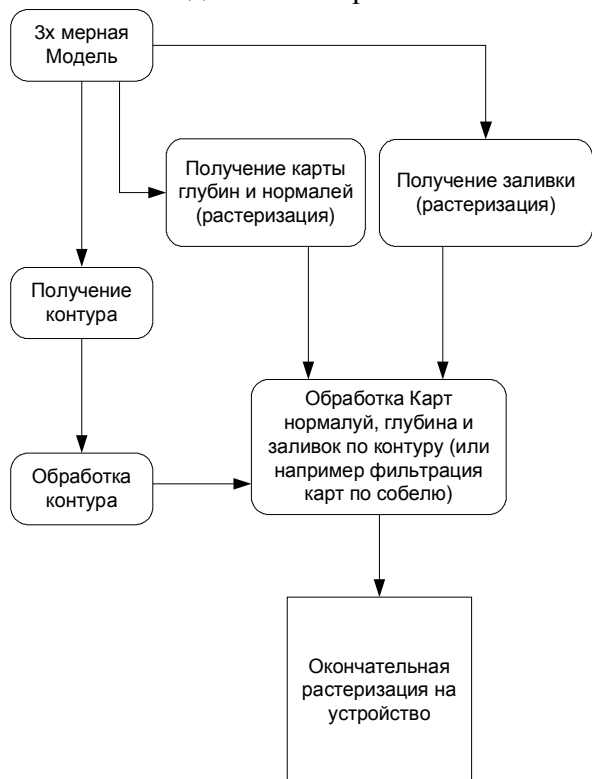
Особенности и выводы

Как можно видеть, основной отличительной чертой существующих конвейеров является линейность и независимость стадий. При такой организации данные с одной стадии конвейера поступает на следующую в линейке и теряется информация как о преобразовании, прошедшем на стадии, так и информация об исходных данных.

Настраиваемый конвейер

Пример из жизни

В процессе создания нефотореалистичного рендера возникла основная идея, которая сформировалась в результате исследований – нужна возможность пользоваться на каждой стадии отрисовки данными не только с предыдущего уровня, но и с других, а также нужно более гибко связывать стадии конвейера.



Пример такого конвейера показан на рисунке. Основным отличием его от рассмотренных выше является то, что необходимо обрабатывать в одной из стадий данные, полученные с нескольких предыдущих. Линейный конвейер не подходит к этой схеме, так как в нем информация о результатах предыдущих стадий теряется (доступны только результаты последней), а в данном случае данные с разных стадий считаются равноправными. Также невозможно использовать результаты из нескольких стадий.

Скорее всего для реализации понадобится некий буфер, в котором можно хранить как данные начальной 3x мерной сцены, так и данные, полученные на промежуточных шагах. При этом графический конвейер должен быть полностью программируемым для того, чтобы можно было воспользоваться возможностью строить графы из стадий.

Для данного класса задач (нефотореалистичный рендеринг) такой конвейер качественно улучшит возможности программного обеспечения. Почему?

Используя такой конвейер, мы получаем возможность одновременно использовать векторные и растровые данные. Например, мы можем посчитать векторно контур, а потом наложить его на растровую картинку. Качественное улучшение состоит в том, что можно получить разное качество картинки со стадий. В некоторых случаях рендер идет в четырехкратном разрешении для того, чтобы получить при уменьшении сглаженные линии. Контур при этом останется без потери качества. Также, если

же у нас будет возможность сбрасывать отдельно результаты стадии, то можно будет использовать послойное наложение в любой из программ для постпроцесса. В любом случае просчитывать контур по Z-буферу удобнее на стадии отрисовки, чем на постпроцессе.

В том числе появляется возможность распараллеливать графические вычисления, используя эту архитектуру. Те мы можем перенести выполнение стадий на отдельные, возможно, специализированные компьютеры для ускорения обработки.

Обзор Архитектуры

Изменяемая (нелинейная) конфигурация графического конвейера

Возьмем как пример схему, показанную на рисунке выше. Что же необходимо для реализации такого нелинейного конвейера?

Процесс рендеринга имеет очень сильные связи стадий. Мы не можем начать растеризацию до тех пор, пока не получим конечные значения вертексов. Также мы не можем отфильтровать изображение до того, как оно будет получено. В системе с линейной архитектурой все просто – мы последовательно выполняем стадии до момента вывода изображения на экран (в файл и т. д.). В нашей же системе должен присутствовать планировщик, который не позволит выполнить стадию, если все предыдущие не выполнены.

Соответственно, необходимо иметь описание стадии, которое будет включать информацию о том, в каком порядке она будет выполняться, те описание самого графа.

Действия планировщика довольно просты. Нужно начать стадию, если все предыдущие выполнены (возможно параллельное выполнение стадий, которые не зависят друг от друга). Также необходимо предоставить стадии все необходимые данные для выполнения. Для этого самым простым выходом будет предоставление доступа ко всем данным, полученным на каждой стадии.

Произвольный доступ к данным стадиями

В современных графических архитектурах можно увидеть, что данные, поступающие на каждую стадию, имеют совершенно определенный формат. Исключения составляют дополнительные данные для программируемых стадий в графических процессорах. В остальном, формат данных и способ хранения зависит от конкретной стадии. Для реализации нашего конвейера есть два варианта решения: нужно иметь или более-менее универсальный формат для хранения данных или возможность возложить эту обязанность на конкретную стадию. Второе решение кажется наиболее приемлемым. Посмотрим: каждая стадия работает в общем пространстве памяти. В этом случае данные, которые ей необходимы, будут доступны. Предполагается, что стадия не будет иметь право изменять уже полученные данные.

В полностью программном рендере это реально и вполне выполнимо. Трудности могут поджидать при переносе архитектуры на графические ускорители. В этом случае придется делать менеджер ресурсов или организовывать страничный доступ. В любом случае необходимо также расширять набор команд для адресации в таком пространстве.

Параллельное выполнение стадий

Процесс рендеринга может быть распараллелен на нескольких уровнях. Такая обработка возможна как на уровне кадров, если выполняется построение видеопотока, так и на уровне архитектуры рендера и в алгоритмах. На уровне архитектуры предполагается одновременная работа алгоритмов без знаний их специфики, например закраска треугольников. На уровне алгоритмов параллельная работа возможна в случае поддержки вычислений со стороны системы (например, есть множество библиотек Fortran для матричных вычислений в многопроцессорной среде). Мы же рассмотрим только уровень архитектуры.

Как было сказано выше, мы можем распараллелить вычисления, которые выполняются нелинейным конвейером. Есть несколько путей:

Использовать небольшие области на изображении (тайлы), которые будут просчитываться на отдельных машинах, а потом склеивать их на финальной стадии в конечную картинку.

Каждую стадию (или их группу) выполнять на одной машине

Рендерить каждый кадр на одной машине

Третий вариант - самый простой. Он используется везде, где невозможна или ограничена сетевая обработка. Его мы рассматривать не будем.

Первый вариант используется в таких системах как Pixar Renderman и является достаточно распространенным. При такой организации все изображение делится на тайлы – небольшие участки. Далее специальный планировщик раскидывает задания по сети и отдельные машины рендерят те тайлы, которые к ним пришли.

После чего результаты отправляются на основную машину и склеиваются там в целое изображение. Затраты на сетевой трафик определяются объемом сцены, которая пересылается на каждую машину. Этот трафик в случае сложных сцен может достигать огромных объемов.

В нашей архитектуре возможна такая обработка. Минусом может являться то, что возможно на какой-то из стадий понадобятся данные, которые можно получить один раз, например, контур. При этом получается, что мы будем считать контур на каждой из машин – клиентов. Однако при применении таких технологий как Global Illumination, этот недостаток заметен и на рендерах с обычной архитектурой. Поэтому разницы в скорости между линейной и нелинейной архитектурой не будет.

Второй вариант можно использовать только на полностью программируемом конвейере. В этом случае планировщик должен отправлять нужные данные на отдельные машины и забирать при завершении стадии результаты. При этом если конвейер допускает распараллеливание, то граф построен так, что могут выполняться больше 2х стадий одновременно, то они будут выполняться одновременно на разных машинах.

Трафик при таких операциях можно будет свести к минимуму, если использовать дополнительную информацию о том, какие данные нужны на каждой стадии. И в случае с заранее просчитанным контуром можно сильно сократить как время вычисления так и сетевой трафик.

Также можно совместить третий и второй варианты. Тогда очень сложные сцены

можно разбивать не только по стадиям, но и по тайлам. Этот режим работы наиболее сложен, но и может принести наибольший выигрыш.

Использование в системе нефотореалистичного рендеринга

В нефотореалистичном рендере было необходимо использовать нелинейный конвейер для просчета контура по Z-буферу. Описанная выше архитектура произошла именно из этой проблемы, которая была успешно решена.

На данный момент рендер не использует графические ускорители для своей работы и является полностью программным. Соответственно, доступ к данным стадиями не вызвал проблем. Весь конвейер был разделен на логические модули, которые общаются между собой через общую память. Порядок выполнения на данный момент устанавливается жестко, но в будущем будет задаваться извне рендера.

Также в данный момент проводятся работы по распараллеливанию процесса рендеринга на несколько машин, те готовится сетевой вариант системы. Предполагается, что те операции, которые нужно выполнять только один раз будут считаться отдельно от остальных, а параллельные операции будут раскидываться для параллельной обработки.

Использование в системах реального времени

В системах реального времени (имеются в виду графические процессоры) не нужно, а иногда даже невозможно использовать полностью программируемые возможности такого конвейера. Это происходит из того факта, что ускорители на данный момент

являются потоковыми устройствами. Также ясно, что операции, выполняемые аппаратно будут иметь более высокую скорость, чем программные.

Поэтому стоит установить несколько ограничений:

Использовать общую память для данных, таких как начальные вектора, текстуры, параметры для шейдеров, которая будет доступна из каждой стадии.

Ограничить использование графа только параллельным выполнением одной стадии. При этом можно оставить возможность программировать процессоры отдельно.

Все аппаратные операции, которые являются неизменяемыми, например отсечение и растеризацию треугольников нужно оставить.

Добавить в набор команд для программирования стадий операции с общим пулом.

Как можно видеть, эти ограничения почти приближаются к тому, что было описано выше при описании будущей архитектуры DxNext. Однако есть существенное отличие – для использования нелинейности хотелось бы иметь отдельное программирование процессоров, выполняющих работу на одной стадии. Насколько это будет реально – покажет будущее.

Но уже сейчас можно увидеть, что применение такой архитектуры повлечет более полное использование графических ускорителей, которые, например, смогут выполнять более сложные вычисления, например трассировку лучей не используя какие-то дополнительные средства.

Заключение

Результаты данной работы позволят улучшить как качество изображения, так и

помогут в создании новых алгоритмов в смежных областях, таких как различные системы визуализации, CAD.

Насколько скоро такой конвейер будет реализован в графических ускорителях – большой вопрос. Однако сдвиги в этом направлении уже видны (общее виртуальное пространство в DirectX Next). Можно только надеяться на то, что в конце концов архитектура специализированных процессоров будет полностью управляемой.

Литература

[1] John D. Owens, Bruce Khailany, Brian Towles, and William J. Dally. Comparing Reyes and OpenGL on a Stream Architecture, Stanford University Computer Systems Library, published in Graphics Hardware, 2002.

[2] Thomas W. Crockett. Beyond The Renderer: Software Architecture for Parallel Graphics and Visualization. NASA Langley Research Center. ICASE Report № 96-75.

[3] Александр Медведев. DX.Next: Ближайшее и ближнее будущее аппаратного ускорения графики. <http://ixbt.com/video2/dx-next.shtml>

[4] Adam Lake, Carl Marshall, Mark Harris, Marc Blackstein. Stylized Rendering Techniques For Scalable Realtime 3D Animation. G3D, Intel Architecture Labs, University of North Carolina at Chapel Hill.

[5] Isenberg, Helper, and Strothotte. Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Silhouette Strokes. The Eurographics Association and Blackwell Publishers. 2002.