Лекция 3.3. Графический конвейер

Элементы трехмерной сцены

В общем виде трехмерную сцену можно представить как набор отдельных групп элементов: группы трехмерных объектов, группы источников освещения, группы применяемых текстурных карт, группы камер.

Трехмерный объект обладает свойствами координат вершин треугольников в пространстве сцены и локальных координат в пространстве текстурной карты. Ему соответствует алгоритм поведения: параметры масштабирования, угла поворота, смещения и прочие изменения в течение времени в соответствии с замыслом разработчиков. Производным от первых двух свойств является грань — плоскость объекта, имеющая три вершины, с наложенными на нее текстурами.

Источник освещения может обладать следующим набором свойств (полностью или частично): координатами в пространстве сцены, ориентацией (направленностью), типом излучения (фоновым, точечным и т. п.), цветом и алгоритмом изменения светового излучения.

Камера представляет собой точку, откуда наблюдатель обозревает трехмерную сцену. Почему же предпочитают использовать термин камера? Дело в том, что наблюдатель - человек обладает стереоскопическим зрением, а камера имеет только один «глаз». Поэтому глубину сцены приходится имитировать на плоском экране с помощью различных ухищрений. Плоскость, в которой расположена камера, называется плоскостью проецирования. Камера обладает свойствами координат в пространстве сцены, целевой точкой, углом зрения, углом поворота. Линия, соединяющая камеру и целевую точку, называется линией визирования. Угол поворота рассчитывается относительно оси линии визирования.

Текстурой (или текстурной картой) называют двух- или трехмерное изображение, имитирующее зрительное восприятие человеком свойств различных поверхностей. На полигон можно накладывать несколько текстур, смешивая их различными способами, с тем чтобы

моделировать зрительный образ нужного материала. Специализированные текстуры (например, карты окружающей среды) сами не отображаются, а используются для модификации других текстур, накладываемых на полигон.

Аппаратные средства

В самом общем виде современный ускоритель 3D-графики должен иметь: блок геометрических преобразований расчета освещения (геометрический процессор), блок-механизма установки примитивов, блок обработки текстур и блок обработки буфера кадра.

Геометрический процессор обрабатывает примитивы и строит проекцию трехмерной сцены. Блок расчета освещения формирует данные о параметрах освещения для вершин примитивов.

Блок установки примитивов с помощью данных, полученных из буфера глубины, определяет видимость отображаемой точки (пиксела).

Видимые пикселы обрабатываются в блоке текстур различными методами с применением одной из технологий фильтрации и наложения текстурных карт. Обработанный таким образом пиксел вновь помещается в буфер кадра, либо заменяя находящееся там значение, либо смешиваясь с ним по выбранному правилу.

Программные интерфейсы

Скорость и качество обработки трехмерной сцены во многом зависят от совершенства инструкций, передаваемых приложением графическому ускорителю. Такие инструкции объединены в специализированные прикладные программные библиотеки (Graphics API). С одной стороны, инструкции должны учитывать особенности построения аппаратной части графического адаптера. С другой стороны, конструкция графического адаптера должна соответствовать возможностям API.

Точного соответствия между аппаратурой и API не бывает: иногда разработчики видеокарты опережают время и вводят функции, которые не поддерживаются действующими API. Часто в API появляются инструкции, которые не могут быть выполнены конкретным графическим адаптером. Поддержка API реализуется через драйвер

видеокарты. Бывает, что разработчики игр пытаются использовать функции видеокарты, недоступные через стандартную версию API.

Совокупность аппаратных и программных средств обработки трехмерных сцен образует графический конвейер, конечным итогом работы которого является кадр, размещенный на экране монитора. Ниже рассмотрены основные принципы работы такого конвейера.

Шесть этапов работы 3D-конвейера

Большинство приложений трехмерной графики, в том числе игр, при построении объемных сцен придерживаются определенной последовательности действий, в совокупности составляющей 3D-конвейер. Итогом работы 3D-конвейера является отрисовка (рендеринг) результирующего изображения на дисплее компьютера. Группу операции, выполняющих обособленные промежуточные действия, принято называть этапом, или стадией 3D-конвейера. Описываемая ниже последовательность операций отнюдь не является жестко заданной, а скорее общепринятой в современных графических подсистемах. При конкретной реализации на программном и аппаратном уровнях могут появляться существенные отличия, однако смысловое содержание блоков практически не меняется. Итак, на сегодняшний день процесс визуализации трехмерной сцены на экране компьютера выглядит следующим образом.

Первый этап

Здесь определяется состояние объектов, принимающих участие в сцене, которую необходимо отобразить. На первый взгляд к самой графике этот этап отношения не имеет. На самом деле он является определяющим, ибо состояние объектов и их взаимное положение формируют логику последующих действий программы. Например, если вашего персонажа в игре уже «убили», какой смысл вообще выводить трехмерную сцену? С каждым объектом в сцене связана соответствующая текущему моменту геометрическая модель. То есть объект один (например, некий монстр), но с ним сопоставляются разные геометрические модели в зависимости от состояния — жив,

ранен убит, трансформировался в другой объект и т.д. Практически все операции на первом этапе выполняет центральный процессор. Результаты его работы пересылаются в графический чипсет посредством драйвера.

Второй этап

Происходит декомпозиция (разделение на примитивы) геометрических моделей. Внешний вид объекта формируется с помощью набора определенных примитивов. Чаще всего в роли примитива выступает треугольник как простейшая плоская фигура, однозначно располагаемая в трехмерном пространстве. Все прочие элементы состоят из таких треугольников. Таким образом, можно утверждать, что по большей части термины «полигон» и «треугольник» применительно к игровой 3D- графике суть синонимы.

Современные графические процессоры умеют выполнять дополнительные операции, например тесселяцию (Tessellation), то есть разделение исходных треугольников на более мелкие. Некоторые графические чипсеты могут аппаратно обрабатывать геометрические модели, построенные на основе параметрических поверхностей (механизм RT-Patches). Часть графических процессоров умеет превращать плоские треугольники в трехмерные поверхности путем «выдавливания» в третье измерение (механизм N-Patches).

Итоговый результат операций второго этапа пересылается в блок трансформаций и освещения (Transform&Lighting, T&L) геометрического процессора.

Третий этап

В блоке T&L на аппаратном уровне к вершинам треугольников применяют различные эффекты преобразований и освещенности. Содержание операций блоков T&L новейших графических чипсетов можно динамически изменять посредством вершинных шейдеров (Vertex Shaders) — специальных микропрограмм, включаемых в код игры. То есть сегодня персональный компьютер в дополнение к центральному процессору получил полноценный программируемый

графический процессор. По завершении операций трансформации и расчета освещенности параметры вершин нормализуются и приводятся к целочисленному виду.

Четвертый этап

На данном этапе происходит так называемая установка примитивов (Triangle Setup). Графический процессор пока ничего не знает о свойствах треугольников, поскольку обрабатывал вершины по отдельности. Теперь необходимо «собрать» вершины в треугольники и преобразовать результаты в координаты и цвет каждого пиксела, а также отсечь невидимые области.

В ходе «сборки» определяется видимость объектов с позиции камеры. Полигоны, находящиеся ближе к камере, могут загородить более удаленные полигоны. Для хранения информации о степени удаленности объекта от плоскости проецирования используют специальный буфер глубины (Z-буфер). Современные графические процессоры применяют различные механизмы отсечения невидимых полигонов на ранних этапах 3D-конвейера с тем, чтобы избежать излишних операций. Данные буфера глубины обрабатываются специализированными блоками графического процессора. В конечном счете, на выходе блока геометрических преобразований получают проекцию трехмерной сцены на плоскость визуализации. Координаты и исходный цвет видимых пикселов передаются в текстурный конвейер (Texture Pipeline).

Пятый этап

Графический чипсет может иметь несколько параллельных текстурных конвейеров. В каждом из них происходит наложение текстур различного типа, в том числе и тех, которые сами не отображаются (например, карты высот), а служат для модификации других текстур. На этом этапе в современных чипсетах возможно исполнение пиксельных шейдеров (Pixel Shaders) — специальных микропрограмм, определяющих порядок смешивания текстур, полученных на выходе из каждого конвейера. Здесь также учитывается

информация (полученная на этапе установки примитивов) о принадлежности пиксела к определенному треугольнику. В целом указанные операции составляют суть процесса визуализации (рендеринга).

Шестой этап

На заключительном этапе работы 3D-конвейера к полученному в текстурном блоке плоскому изображению применяют операции устранения дефектов (например, часто используют билинейные, анизотропные и другие фильтры). Операции конечной обработки применяют какой-либо эффект (например, туман) к целиком сформированному изображению. Перед выводом в буфер кадра в последний раз проверяется видимость пикселов, и наконец изображение появляется на экране.

Перечисленные этапы в конкретных графических чипсетах могут быть переставлены, разделены, объединены, выполняться неоднократно (в несколько проходов), однако их физический смысл остается неизменным. Технологически некоторые элементы этапов или этапы целиком могут быть выполнены различными способами. Вариант реализации зависит от особенностей приложения и видеокарты.

В зависимости от типа видеоускорителя часть этапов просчитывается программно, а часть — аппаратно. Самые современные ускорители имеют на борту графический процессор, способный аппаратно просчитывать этапы трансформации (преобразований) и расчета освещения, наложения текстур.

Особенности первого этапа работы графического конвейера

Одним из ключевых критериев реалистичности сцены является качество представленных в ней моделей. Качество отображения, близкое к фотореализму, возможно при наличии 500 000 и более треугольников в сцене. Объем данных для обработки каждой вершины треугольника составляет, как минимум, 12 байт: 3 для координат вершины + 3 для параметров нормалей в вершинах + 3 для координат

текстуры + 3 для параметров цвета. Для треугольника в целом минимальный объем данных составит 36 байт. Если же применить различные эффекты, предусмотренные, например, в библиотеке DirectX (карты окружения, анизотропную фильтрацию), то максимальный объем данных для одного треугольника достигает 768 байт. Принято считать, что средний объем данных на каждый треугольник составляет 180 байт. При наличии полумиллиона треугольников в сцене общий объем данных составит около 90 Мбайт. Приемлемая скорость смены кадров в играх — примерно 30 раз в секунду, то есть общий поток данных превысит 2,5 Гбайт/с.

Путем оптимизации моделей часто удается добиться совмещения вершин треугольников. Выдающимся считается коэффициент оптимизации 0,5 — это снижает поток данных вдвое, в нашем случае примерно до 1,3 Гбайт в секунду. Напомним, что мы ведем расчет только для геометрических данных. При максимальных требованиях к качеству и скорости поток данных возрастает в пять-восемь раз! А ведь по шине помимо «геометрии» необходимо передать и текстуры, и служебные команды. Очевидно, что на современном этапе развития аппаратной составляющей графической системы ПК решение проблемы «в лоб» невозможно.

Поэтому разработчики видеокарт и программисты начали искать другие способы.

Одним из методов улучшения формы трехмерных полигональных моделей и снижения потока данных служит применение трехмерных поверхностей высокого порядка. В этом случае исходный объект можно представить меньшим числом элементов, а детализацию выполнить средствами графического адаптера путем разбиения граней на мелкие треугольники (тесселяция). В библиотеке DirectX версии 8 появились средства описания поверхностей высокого порядка с помощью набора контрольных точек (Patshes).

Второй этап: борьба за качество

В общем случае после декомпозиции модели блок геометрического конвейера в качестве исходных данных получает данные о координатах

вершин грани объекта и нормалях (векторах к освещаемой поверхности) в вершинах. В зависимости от заданной степени тесселяции исходный треугольник может различными способами разбиваться на более мелкие (от 4 до 64). Этот процесс эффективно обсчитывается в графическом процессоре. Тесселяция позволяет несколько повысить качество отображения объектов, однако не избавляет от коренного недостатка: сравнительно грубых исходных граней.

Параметрические поверхности.

В библиотеке DirectX 8.1 появились инструкции для работы с поверхностями Безье — набор RT-Patches.

Параметрические поверхности, описываемые типом RT-Patches (Rectangular and Triangular Patches) имеют прямоугольную форму. В библиотеке DirectX использованы поверхности Безье первого, второго и пятого порядка. Поверхность первого порядка представляет собой плоскость, а более высокого порядка — трехмерную структуру. Поверхность пятого порядка имеет большое число контрольных точек, что позволяет более точно моделировать сложные кривые, задающие поверхность.

Графический процессор, поддерживающий RT-Patches, разбивает представленную приложением поверхность высокого порядка на сегменты (треугольники и вершины), исходя из заданных разработчиком программы параметров. Каждый этап тесселяции увеличивает число сегментов вдвое. Тем самым обеспечивается адаптивная детализация поверхностей высокого порядка, связанная с расстоянием до плоскости проецирования. Вблизи камеры можно применить подробную детализацию, а на достаточном расстоянии задействовать минимум ресурсов.

Применение параметрических поверхностей позволяет моделировать живые объекты и динамически модифицируемые формы (например, колебания ткани платья в движении) на фотореалистичном уровне. Такая технология широко применяется в мощных программах трехмерного моделирования. Однако в 3D-приложениях рендеринг

сцены идет продолжительное время, что совершенно неприемлемо для игр. Поддержка RT-Patches на аппаратном уровне открывает возможность использования в играх моделей, качество которых не хуже кинематографических персонажей, поскольку вместо огромного потока сведений о треугольниках достаточно передать формулы, задающие поверхности Безье (объем передаваемых данных снижается на два порядка!).

Главными препятствиями на пути внедрения технологии RT-Patches в играх выступает трудоемкость их разработки и несовместимость с видеоадаптерами, не поддерживающими данную технологию. То есть разработчик игры должен либо создавать все модели на основе параметрических поверхностей (и тем самым исключить из круга потенциальных покупателей владельцев «устаревшего» железа), либо выполнять двойную работу, создавая дополнительный комплект объектов из традиционных моделей, построенных на основе треугольников.

Обработка RT-Patches на аппаратном уровне была впервые обеспечена в блоке Surface Engine геометрического процессора чипсета GeForce3. Однако отсутствие реальных приложений с моделями на основе параметрических поверхностей, привели фирму nVIDIA к решению отказаться (быть может, временно) от поддержки RT-Patches в чипсетах GeForce3/4 (путем отключения этой функции в драйверах, начиная с версии 21.81).

Имитация параметрических поверхностей

Каждая вершина полигона полностью описывается параметрами трехмерных координат, нормали, цвета, текстурной координаты. При обработке вершины блоком T&L или программой (вершинным шейдером) графический процессор не имеет какой-либо информации о соседних вершинах треугольника. Нормаль (вектор, перпендикулярный к моделируемой поверхности) также является атрибутом вершины, но не треугольника.

Трехмерные поверхности на основе N-Patches строят на базе контрольных точек, расставляемых графическим процессором на

каждой стороне исходного треугольника. Суть N-патчей проста — из нормалей можно получить информацию о кривизне поверхности. Для этого требуется выполнить два условия: получить данные о всех трех вершинах полигона и сгруппировать их.

Получив координаты вершин и значения нормалей в вершинах к создаваемой трехмерной поверхности, графический процессор, поддерживающий технологию N-Patches, рассчитывает положение контрольных точек, разделяя стороны треугольника на равные отрезки. Тем самым исходный треугольник разбивается на более мелкие треугольники (до 64). Значения нормалей в генерируемых контрольных точках получают методом линейной или квадратичной интерполяции.

Координаты контрольных точек рассчитываются методом их проецирования на плоскость, перпендикулярную к нормали в вершине исходного треугольника. Если все три нормали в вершинах будут перпендикулярны к плоскости исходного треугольника, он так и останется плоским, разбей его хоть на 1000 частей.

Третий этап: борьба за скорость

Современные графические ускорители (серий GeForce и Radeon) имеют геометрический конвейер сходной архитектуры, с аппаратной поддержкой T&L и вершинных шейдеров. Источником данных обычно является буфер вершин, который хранится в локальной видеопамяти. В буфер вершин параметры вершин треугольников поступают массивами (Triangle Strip, Triangle Fan), в последовательности, заданной приложением. Из буфера данные считываются в кэш (условно именуемый кэшем подготовки — Pre-T&L Cashe), обслуживающий блоки T&L и обработки вершинных шейдеров (Vertex Shaders ALU). Буфер вершин бывает двух типов: обычный и индексируемый.

Из обычного буфера данные считываются последовательно, образуя поток вершин (Vertex Stream). Такой подход кардинально снижает количество промахов кэша, однако никак не влияет на общий объем потока данных.

В индексируемом буфере параметры вершин описываются индексами. Поэтому вершина с подходящими параметрами может использоваться в модели многократно, что существенно сужает поток данных. Индексируемый буфер вершин применен во многих современных графических чипсетах.

При считывании параметров вершин из оперативной памяти производительность чипсета определяется полосой пропускания шины.

Если все вершины поместились в кэш подготовки, то эффективность графического процессора ограничивается только производительностью блоков T&L и обработки вершинных шейдеров.

Вершинные шейдеры

Большинство графических чипсетов располагают блоком расчета трансформаций и освещения (T&L), выполняющим фиксированные (заданные аппаратно) операции: установка параметров рендеринга освещения, текстур, матричные преобразования.

Шейдерами (Shaders) называют микропрограммы, задаваемые инструкциями API, которые динамически изменяют алгоритм работы графического процессора видеокарты. Вершинные шейдеры (Vertex Shaders) определяют порядок трансформации вершин треугольников. Как уже говорилось выше, минимально необходимый объем данных для каждой вершины полигона составляет 12 байт. Следовательно изменение свойств вершины вызывает необходимость пересчета большого объема данных, что сильно загружает центральный процессор и шину. С появлением чипсетов, поддерживающих технологию вершинных шейдеров (GeForce3/4, Radeon 8500), задачи расчета трансформации вершин решаются графическим процессором, то есть объем геометрических данных, передаваемых по шине, резко снижается. Например, вместо громадного массива новых параметров освещения вершин теперь достаточно передать несколько коротких инструкций API для графического процессора.

В спецификации DirectX 8.1 определено 17 инструкций для обработки свойств вершин, а длина шейдера ограничена 128 инструкциями. Укажем важнейшие операции, которые можно выполнять аппаратными средствами графического процессора:

- межкадровая интерполяция вершин (Key Frame Interpolation), что существенно ускоряет анимацию;
- наложение вершин (Vertex Blending) с использованием более чем четырех матриц преобразования, что облегчает «скелетную» анимацию сложных моделей без их разбиения на несколько частей;
- процедурная геометрия, то есть искажение свойств вершин каким-либо параметрическим объектом (например, волны на водной поверхности или перекатывающиеся под кожей мускулы персонажа);

сложные модели освещения, учитывающие свойства материала объектов.

Кэширование шейдеров позволяет хранить в графическом чипсете несколько программ, что уменьшает время их пересылки из памяти. При использовании API DirectX кэшированием шейдеров управляет драйвер видеокарты, а в API OpenGL можно прямо указывать, какие шейдеры будут резидентными. Технология вершинных шейдеров является значительным шагом вперед на пути к фотореалистичной графике. Вершинные шейдеры редко используются обособленно, в большинстве операций они тесно связаны с пиксельными шейдерами.

Адаптивная детализация

Современные приложения трехмерной графики позволяют создавать для игровых программ реалистичные объекты с высокой степенью детализации даже при использовании исключительно полигонов. Однако с практической точки зрения нерационально все объекты трехмерной сцены отображать с максимальной детализацией, так как человеческое зрение отказывается воспринимать мелкие детали удаленных объектов.

Очевидно, что и в играх нет смысла воспроизводить каждый трак на гусеничной ленте танка, если человек все равно не способен его различить. К тому же непрерывная обработка нескольких сотен или

тысяч полигонов, составляющих объект, независимо от его удаления, приводит к загрузке компьютера бессмысленной работой.

Таким образом, на каждый момент времени необходимо просчитать для всех объектов в трехмерной сцене так называемый LOD (Level of Detail — уровень детализации). Существуют два стратегических подхода к управлению детализацией: статический и динамический LOD. В первом случае заранее создают упрощенные варианты максимально детализированного объекта. Предположим, что для модели танка, состоящей из 1200 полигонов, достаточно сделать упрощенные варианты из 600 и 300 полигонов. В ходе построения сцены просчитывается удаление модели танка от плоскости проецирования и выбирается вариант, соответствующий дальности.

В технологии динамического, или непрерывного, LOD используют различные алгоритмы, позволяющие плавно регулировать число полигонов в объекте в зависимости от расстояния до картинной плоскости, обычно за счет тесселяции треугольников.

У каждого метода есть свои преимущества и недостатки. Статическое управление вызывает эффект «дерганья» изображения при смене детализации объекта. Если уровень LOD близок к граничному значению, иногда возникает циклическая смена моделей с разным уровнем детализации. К тому же приходится обсчитывать несколько разных моделей для одного объекта.

Динамическое управление детализацией потребляет значительные вычислительные ресурсы, требует непрерывного пересчета не только координат вершин треугольников, но и параметров освещенности. При частом переключении между уровнями иногда наблюдается эффект «волнистости» поверхности — форма объекта непрерывно «плывет» что в неживой природе выглядит особенно нереально.

Четвертый этап: ухищрения сборки

После трансформации, освещения и затенения вершины объединяются по три (этот процесс называется сборкой треугольников) для передачи блоку установки примитивов и последующей растеризации. Так как треугольники часто имеют общие вершины,

обычно требуется обработка лишь одной вершины для каждого нового треугольника. Недостающие вершины считываются из второго кэша, который можно условно назвать кэшем постобработки (Post-T&L Cache). Если все вершины помещаются в кэш постобработки, производительность графического чипсета определяется эффективностью блока установки примитивов. Для чипсета GeForce3 Ti500 мощность блока установки примитивов достигает 47 миллионов вершин в секунду, для чипсета Radeon 8500 — 67 миллионов.

Буфер глубины

Как уже было рассмотрено, в трехмерной графике одной из координат является значение Z, которое обычно характеризует глубину сцены, то есть степень удаления объектов от камеры (плоскости проецирования). Очевидно, что проекция непрозрачных объектов, расположенных ближе к камере, будет частично или полностью перекрывать проекцию удаленных объектов, исключая их видимость для наблюдателя. Кроме того, грани объекта, обращенные к плоскости проецирования, препятствуют видимости расположенных с тыла элементов. Графический адаптер должен определить, какие объекты следует отображать, а какие скрыть. Для этого координаты глубины помещают в Z-буфер.

Для оценки сложности сцены применительно к видимости ее элементов по глубине применяют показатель перекрытия полигонов (Overdraw). Например, если значение Overdraw составляет три единицы (что характерно для современных программ), это означает, что в среднем один видимый полигон перекрывает три полигона, расположенных глубже.

Для раннего определения видимости по глубине применяют различные технологии. В частности, фирма nVIDIA в своих графических чипсетах применяет механизм раннего отсечения невидимых поверхностей HSR (Hidden Surface Removal), а фирма ATI использует иерархическое представление данных в Z-буфере (механизм Hyper Z). В современных графических адаптерах также

широко применяют кэширование Z-буфера, сжатие Z-координат без потерь и механизм быстрой очистки буфера глубины.

Пятый этап: проблемы закраски

Для ускорения обработки текстур используют параллельную работу нескольких (обычно 2—3) конвейеров. В этом случае операции комбинирования двух-трех текстур выполняются за один цикл (проход).

Пиксельные шейдеры

Пиксельные шейдеры (Pixel Shaders) служат для динамического изменения свойств отдельных пикселов. Единственная задача пиксельного шейдера — вычислить цвет одиночного пиксела на основе исходных значений, полученных от вершинного шейдера (координаты текстур) и заданных параметров освещения. Выполнение программируемых пиксельных операций является завершающей стадией 3D-конвейера, после которой данные поступают в буфер кадра и на экран.

До появления пиксельных шейдеров цвет пиксела мог вычисляться на основе только двух текстур, причем оба текстурных модуля не были связаны между собой.

В чипсете GeForce3 впервые реализован механизм программируемого текстурного блока, способный оперировать с четырьмя текстурами по командам пиксельного шейдера длиной до 12 инструкций. В чипсете Radeon 8500 можно использовать шейдеры длиной до 22 инструкций, а оперировать шестью текстурами.

Технология пиксельных шейдеров позволяет успешно решать проблему смешивания нескольких текстур, в том числе карт отражения, карт теней (освещения), объемных текстур и прочих. Только с появлением пиксельных шейдеров в играх появились текстуры, реалистично имитирующие воду и облака.

Адаптивное текстурирование

Самым простым способом текстурирования является создание единственной текстуры среднего размера и наложение ее на полигоны объекта во всех случаях, когда требуется визуализация. Однако здесь возникают некоторые проблемы. Во-первых, если текстура создана с высоким значением разрешения в полной цветовой палитре (например, 4096x4096 пикселов при глубине цветового охвата 32 бит), то расход вычислительных ресурсов и памяти при наложении текстур на все объекты будет просто фантастическим — никакой ускоритель не справится с таким потоком данных.

Во-вторых, если уменьшить размер (и тем самым детализацию) текстур, то объекты по мере приближения к плоскости проекции будут выглядеть все более грубо. В некоторых играх нередки эффекты, когда при максимальном приближении поверхность объекта предстает как набор огромных пятен — это означает, что на картинную плоскость попало всего несколько текселов (ТЕКстурных пикСЕЛОВ). То же самое касается и эффекта перспективы — одна и та же текстура, будучи наложена на близкий и удаленный участки объекта, вызывает искажение перспективы.

Чтобы обойти такие проблемы, была разработана технология, получившая название MIP- mapping. Ее суть заключается в предварительном или динамическом создании набора текстур с различным разрешением и уровнем детализации на основе базовой текстуры максимального разрешения. При построении трехмерной сцены определяется удаленность полигона от картинной плоскости и в соответствии с этим значением накладывается текстура с заданным разрешением.

Фильтрация текстур

Очевидно, что цвет пиксела проекции трехмерного объекта на экране должен однозначно определяться цветом тексела соответствующей текстуры. Однако это правило действует только в простых случаях, при углах проецирования, близких к нормали, и соответствующем удалении от картинной плоскости. Ведь как пиксел

так и тексел только в математическом смысле именуются точками. Физически они выглядят как окружности, диаметр которых зависит от разрешения экрана монитора или размера текстуры. Как только угол проецирования выходит за определенные рамки, случается, что на один пиксел проецируется более одного тексела, так как проекция становится овальной. Если же объект близок к плоскости проецирования, может случиться обратная коллизия — один тексел будет проецироваться на несколько пикселов. Для вычисления правильного цвета пикселов используют различные методы фильтрации текстур.

Трилинейная фильтрация

Для повышения качества отображения объектов, имеющих перспективу по глубине, разработана технология трилинейной фильтрации. По сути, она представляет собой комбинацию технологий MIP-mapping и билинейной фильтрации. Как известно, в MIP-mapping применяют текстуры с разной степенью детализации (и разным разрешением) в зависимости от удаленности полигона от плоскости проецирования. При трилинейной фильтрации берутся две соседние текстуры, одна из которых содержит текселы, попадающие в проекцию, а другая является ближайшей к ней по удаленности, и к каждой применяют билинейную фильтрацию.

В итоге аппроксимация цвета пиксела проводится уже по восьми текселам и результат выглядит более близким к реальности, так как текстуры для определенных расстояний обсчитаны заранее. Однако и к пропускной способности памяти требования в восемь раз выше, чем, при поточечной фильтрации.

Композитные текстуры

Текстуры высокого разрешения занимают в памяти много места. Учитывая широкое распространение программ с глубиной цветности 32 бит (текстура 2048x2048, 32 бит, занимает 16 Мбайт!), становится понятным, что никакой видеопамяти, при сложности сцены хотя бы в 10 000 полигонов, не хватит.

Для решения этой проблемы была придумана технология так называемых композитных текстур, или текстур с детализацией. При таком подходе требуется создать всего две текстуры: базовую и детальную. Базовая текстура содержит основные элементы и как бы создает общий фон. Детальная текстура содержит лишь мелкие элементы, необходимые при рассмотрении объекта вблизи. Обе текстуры смешиваются, причем степень их взаимовлияния определяется исходя из расстояния до плоскости проецирования. В результате на большом расстоянии видна только базовая текстура, а по мере приближения объекта к картинной плоскости начинает проявляться детальная текстура.

Шестой этап: борьба с дефектами

Дефекты изображения, возникающие при рендеринге трехмерной сцены, могут носить самый различный характер. Обычно пространственные искажения выражаются в ступенчатости ровных краев (так называемый «лестничный эффект»), потере мелких деталей изображения, появлении муара (регулярной структуры на изображении, не предусмотренной разработчиками), искажении текстур. Технологии устранения дефектов получили название anti-aliasing (сглаживание). Пространственные дефекты сглаживаются либо локальными, либо глобальными методами.

Локальные методы применяют к краям объектов, то есть текселы текстуры, являющиеся краевыми в полигоне, обрабатываются таким образом, чтобы исключить лестничный эффект. Глобальные методы применяют ко всей области картинной плоскости.

В современных 3D-ускорителях самой популярной является технология полноэкранного сглаживания дефектов сцены (FSAA — Full scene anti-aliasing), основанная на масштабировании изображения. То есть, рендеринг сцены выполняется с разрешением, превосходящим необходимое экранное разрешение, а затем изображение уменьшается, при этом каждый пиксел обрабатывается на основе маски, соответствующей коэффициенту масштабирования.

Например, при маске 2x2 и экранном разрешении 800x600 точек необходимо выполнить рендеринг сцены в разрешении 1600x1200. Затем каждому пикселу в реальном изображении будет присвоен цвет, получившийся в виртуальном разрешении в результате смешения цветов соседних пикселов маски 2x2.

Имитация движения

Для борьбы с непреднамеренными эффектами анимации используют технологию Motion Blur, которая рассмотрена подробно, а также метод аппроксимации нескольких фаз движения объекта в одном кадре итогового рендеринга сцены.

Проведите такой эксперимент — на улице резко поверните голову от одного плеча к другому, стараясь не двигать глазами. Вы наверняка заметите, что зрительная картина приобретает некоторую «смазанность», направление которой зависит от направления перемещения взгляда. Такой же эффект возникает, если смотреть из окна быстро движущегося поезда или автомобиля на близко расположенные объекты, которые иногда сливаются в одну полосу.

Разработчики игр и мультимедийных продуктов давно поняли, что эффект Motion Blur способен значительно повысить восприятие, «ощущение» движения в трехмерных сценах. Однако лишь сравнительно недавно возможности аппаратуры позволили в полной мере применять в современных продуктах эффект размывания изображения в движении.

Технология Motion Blur, применяемая в настоящее время, достаточно проста, но требует значительных вычислительных ресурсов. Суть ее заключается в создании избыточных данных и их приведении к нормальному значению. То есть, сначала создается избыточное количество кадров для анимационной последовательности, обычно кратное целому коэффициенту (например, восемь). Затем последовательность кадров делится на группы, включающие число кадров, равное принятому коэффициенту. Внутри группы все кадры смешиваются, и обсчитывается результирующий кадр. Из результирующих кадров составляется итоговая анимация.

Так как результирующие кадры, по сути, содержат несколько фаз движения, изображение получается размытым, со смещением границ в сторону, противоположную направлению движения. Чем больше принятый коэффициент Motion Blur, тем дальше отодвигается граница размытости изображения. В принципе, коэффициент должен быть прямо пропорционален скорости перемещения камеры.