Шейдеры

Нынешние видеокарты - это программируемые устройства. Некоторый код можно выполнять прямо на графическом процессоре (**GPU**). Эти программы называют ***шейдерами***.

Шейдер – программа, которая загружается в ускоритель, и конфигурирует его узлы для обработки соответствующих элементов. Шейдер - короткая программа для одной из ступеней графического конвейера, позволяющая программировать графический ускоритель. На практике шейдер - некая последовательность машинных кодов, которую разработчик, как правило, описывает на специальной разновидности ассемблера.

Шейдеры делятся по своим функциям на вершинные, пиксельные, геометрические.

Вершинные шейдеры (Vertex Shader) оперируют данными, сопоставленными с вершинами многогранников. К таким данным, в частности, относятся координаты вершины в пространстве, текстурные координаты, тангенс-вектор, вектор бинормали, вектор нормали. Вершинный шейдер может быть использован для видового и перспективного преобразования вершин, генерации текстурных координат, расчета освещения и т.д.

Геометрические шейдеры (Geometry Shader) в отличие от вершинного, способны обработать не только одну вершину, но и целый примитив. Это может быть отрезок (две вершины) и треугольник (три вершины), а при наличии информации о смежных вершинах (adjacency) может быть обработано до шести вершин для треугольного примитива. Кроме того геометрический шейдер способен генерировать примитивы «на лету», не задействуя при этом центральный процессор.

Пиксельные шейдеры (Pixel Shader) работают с фрагментами изображения. Под фрагментом изображения в данном случае понимается пиксель, которому поставлен в соответствие некоторый набор атрибутов, таких как цвет, глубина, текстурные координаты. Фрагментный шейдер используется на последней стадии графического конвейера для формирования фрагмента изображения.

Шейдеры позволяют создавать сложнейшие эффекты в реальном времени. Изначально эти программы можно было писать только на "фирменных" языках производителей видеокарт - ***ATI*** и ***nVidia***. Оба языка очень похожи на ассемблер, но, увы, несовместимы.

В **DirectX 8.0** появляется универсальный язык программирования шейдеров. Он тоже похож на ассемблер, но позволяет обойтись одним шейдером **для обоих типов видеокарт**.

Позднее появились шейдерные языки программирования более высокого уровня.

В **DirectX 9.0** появился С-подобный язык **HLSL** (HLSL — High Level Shader Language)

Наблюдательный совет за архитектурой **OpenGL** (OpenGL **A**rchitecture **R**eview **B**oard – ARB) разработал аналогичную технологию **GLSL** (The OpenGL Shading Language). Большинство возможностей языка ANSI C сохранено, к ним добавлены векторные и матричные типы данных, часто применяющиеся при работе с трехмерной графикой. В контексте **GLSL** шейдером называется независимо компилируемая единица, написанная на этом языке. Программой называется набор откомпилированных шейдеров, связанных вместе.

nVidia совместно с **Microsoft** разработала язык **Cg** (C for Graphics) (такой же по сути язык от Microsoft как и **HLSL**). Язык очень похож на C, он использует схожие типы. Поддерживаются функции и структуры. Несмотря на то, что язык разработан **nVidia**, он без проблем работает и с видеокартами ATI. Однако следует учесть, что все шейдерные программы обладают своими особенностями, которые следует получить из специализированных источников описания архитектуры ускорителей.

Эти технологии далеки от совершенства, прежде всего из-за слабой оптимизации компиляторов.

Программные интерфейсы

Поскольку сердцем современного видеоадаптера является *графический процессор*, имеющий собственную систему команд, эффективное использование возможностей 3D-ускорителя подразумевает понимание такой системы команд прикладными программами.

Однако ***при широкой номенклатуре графических процессоров*** нельзя написать программу, которая бы одинаково эффективно работала с любой системой команд любого графического процессора.

Поэтому и разработчики программ, и создатели графических процессоров нуждаются в универсальной системе, обеспечивающей преобразование запросов программы в последовательность команд 3D-ускорителя и программную реализацию отсутствующих в графическом процессоре аппаратных блоков.

Роль такой системы играют специализированные прикладные программные библиотеки или интерфейсы прикладного программирования (API – Application Program Interface).

Использование API позволяет разработчикам программ делать их универсальными, абстрагируясь от низкоуровневых команд конкретного *графического процессора*.

В настоящее время подавляющее большинство прикладных программ, работающих с трехмерными объектами, опираются на одну из двух типовых библиотек — **OpenGL** или **DirectX**.

DirectX построен по объектно-ориентированной схеме, а **OpenGL** по процедурной.

Что лучше?

Программа на **OpenGL** одинаково "хорошо" выглядит и на C++, и на чистом C, чего не скажешь о **DirectX**. Простота архитектуры **OpenGL** - неоспоримый плюс: **OpenGL** работает исключительно с примитивами (треугольники, отрезки и точки) и управляется набором булевых переменных, которые позволяют включать или отключать некоторые функции - например, накладывать текстуру или нет, использовать ли освещение и т. д. Код для отображения "вашего первого треугольника" занимает примерно пятьдесят строк.

В **DirectX** эта цифра куда больше. С одним-то треугольником у **DirectX** все хорошо, но как только захочется использовать что-нибудь из современных 3D-эффектов - появляются расширения **DirectX**, и еще недавно простой и понятный код тонет в непонятных и ничего не значащих для человека, не посвященного в тайны 3D-графики, строках.

Производительность примерно одинакова для обеих библиотек, поскольку сейчас практически все функции реализуются напрямую через аппаратные ускорители.

Расхождение результатов может быть только из-за погрешности измерений.

В настоящее время большая часть создателей игр выбирает **DirectX** за удобную поддержку современных возможностей, остальные предпочитают **OpenGL** - за переносимость и простоту написания программ.

Таким образом, у нас есть две библиотеки с практически одинаковыми возможностями и быстродействием.

Отличия лишь в сложности написания кода.

Так что же выбрать?

Чтобы ваша программа работала не только на **Windows**, но и на других операционных системах, лучше выбрать **OpenGL**.