Обзор предметной области

Раньше не существовало способов эффективно писать программы для графических процессоров. Если разработчик приложения хотел включить специальные эффекты для обработки изображения, то ему приходилось либо добавлять обработчики, работающие на CPU, что значительно замедляло скорость выполнения программы, либо довольствоваться сильно ограниченным набором поддерживаемых аппаратно эффектов. Для удобного написания программ разработчикам требовалось создать специализированный язык, который бы позволял создавать простые и быстрые программы для создания и применения различных эффектов. Именно для этой цели была разработана и стандартизирована концепция шейдеров. Благодаря им, теперь практически в любой современной графической симуляции используется код, написанный для видеопроцессора: от реалистичных эффектов освещения в высокотехнологичных AAA-играх до двухмерных эффектов постпроцессинга и симуляции жидкостей.

Однако, бывают случаи, когда программирование шейдеров представляется загадочной чёрной магией и его часто понимают неправильно. Существует множество примеров кода, демонстрирующих создание невероятных эффектов, но в которых практически нет объяснений. Даже опытные программисты шейдеров сталкиваются с проблемой, когда реализация шейдера отнимает уйму времени ввиду сложности его представления.

для воссоздания визуальных эффектов на видеопроцессоре исполняется специальный код, который называется шейдером. По своему типу шейдеры делятся на два основных типа:

- вершинные шейдеры оперируют с различными геометрическими данными вершин объектов в трехмерном пространстве;

• пиксельные или фрагментные шейдеры работают с фрагментами изображения, пикселями, преобразуя их некоторым образом, что может использоваться для постпроцессинга изображения – преобразования уже полученного изображения в новое (например, для применения эффекта черно-белого цвета).

Чтобы понять, как работают шейдеры и в каком порядке они применяются, можно рассмотреть упрощенную схему графического конвейера (рис. 1).

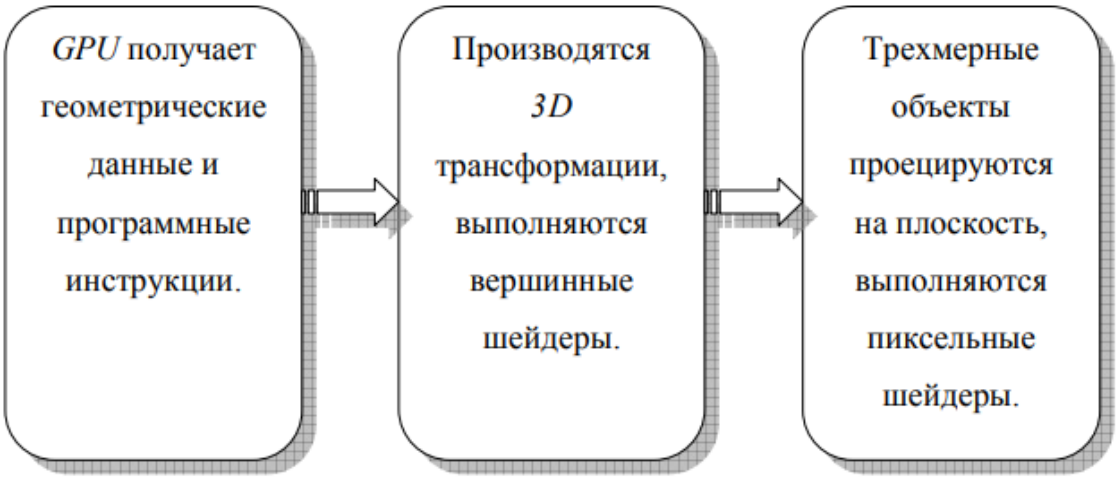

Рис. 1 Упрощенная схема графического конвейера

Обычно для решения задач на графических процессорах чаще используются пиксельные шейдеры. Как показано выше, на последнем этапе происходит проецирование трехмерных объектов на плоскость. На объекты накладывается текстура (двумерное изображение, накладываемое на грани трехмерных объектов), а для каждого пикселя вычисляется результирующее значение цвета. Далее для каждого пикселя выполняются фрагментные шейдеры. В результате работы последнего из них получается финальный цвет пикселя, который и выводится на экран. Таким образом, каждый фрагментный шейдер будет выполнен N раз, где N – число пикселей результирующего изображения. Именно за счет наличия множества конвейерных процессоров фрагментные шейдеры выполняются быстро.

Модель шейдеров стандартизирует некоторые технические параметры шейдеров. В настоящее время последней версией является Shader Model 6.

Сейчас распространены две графические библиотеки: OpenGL и Direct3D. Для каждой из них были созданы C-подобные языки для написания шейдеров:

- для OpenGL: GLSL (The OpenGL Shading Language);
- для Direct3D: HLSL (High Level Shader Language);
- для OpenGL и Direct3D: Cg (C for Graphics).

Данные языки очень схожи с C, поддерживают различные типы данных, структуры, функции. Начиная с Shader Model 3, поддерживаются операторы ветвления. Отличительной особенностью является наличие встроенных типов для векторов и матриц, а также множества функций, оперирующих с ними. Стоит отметить, что также существуют низкоуровневые, ассемблероподобные языки, например, DirectX ASM.

Среда разработки Unity позволяет писать шейдеры на всех приведенных выше языках, однако, наиболее широко распространена практика написания шейдеров на языке HLSL, это обусловлено более качественной поддержкой языка самим движком Unity.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Hohn Kessenich, Dave Baldwin, and Randi Rost. 2014. The OpenGL© Shading Language (Version 4.50). https://www.opengl.org/registry/doc/GLSLangSpec.4.50.pdf. (дата обращения: 10.12.2019).

2. Khronos Group, Inc. 2009. ARB_shader_subroutine. https://www.opengl.org/registry/ specs/ARB/shader_subroutine.txt (дата обращения: 10.12.2019).