УДК 004.428 DOI: 10.30533/0536-101X-2018-62-2-233-236

О компонентах интерфейсов программирования приложений, использующих двух- и трехмерную графику

© 2018 г. Г.М. Маркелов

Московский государственный университет геодезии и картографии, Москва, Россия
markyandex@gmail.com

On components of application programming interface for rendering 2D- and 3D-graphics

G.M. Markelov

Moscow State University of Geodesy and Cartography, Moscow, Russia
markyandex@gmail.com

Received December 11, 2017 *Accepted March 30, 2018*

Keywords: API, computer graphics, Direct3D, OpenGL, rendering.

Summary. The most popular components of application programming interface for rendering 2D- and 3D-graphics are examined in this article. The study is of an interdisciplinary nature, because data visualization is of great importance both for practical applications and for scientific researches. Author has conducted a retrospective analysis and considered core features of application programming interfaces examined. Methods of their implementation via drivers have been scrutinized, and functional differences obtained as a consequence of these approaches have been investigated. Component object model technology is accomplished via DirectX and application of binary interface OpenGL. OpenGL uses state machine paradigm. Description of the way a developer works with objects is given. A programmer determines variables of condition which affect on a result of the function. Attention to the individual features of application programming application interfaces for rendering 2D and 3D computer graphics is drawn. A level of development complexity according to method of application programming interface driver is assessed. A sequence of rendering a scene using OpenGL is described. A structure of the equipment drivers (kernel and user modes) of considered solutions and described differences between them is given. The most common spheres of implementation of application programming interfaces studied are given. As a result it was concluded that currently due to natural development of competitive products any technical differences have been erased and the choice of the instrument is mostly subjective.

Citation: *Markelov G.M.* On components of application programming interface for rendering 2D- and 3D-graphics. *Izvestiya vuzov* «*Geodeziya i aerofotosyemka*». Izvestia vuzov «Geodesy and Aerophotosurveying». 2018, 62 (2): 233–236. [In Russian]. DOI: 10.30533/0536-101X-2018-62-2-233-236.

Поступила 11 декабря 2017 г. *Принята к печати 30 марта 2018 г.*

Ключевые слова: API, Direct3D, компьютерная графика, OpenGL, рендеринг.

Рассмотрены области применения двух самых популярных интерфейсов. Затронуты вопросы особенностей их реализации и производительности. Приведена схема рендеринга сцены на примере OpenGL. Описаны некоторые аспекты использования OpenGL и Direct3D. Сделан вывод о различиях между Direct3D и OpenGL на данный момент времени.

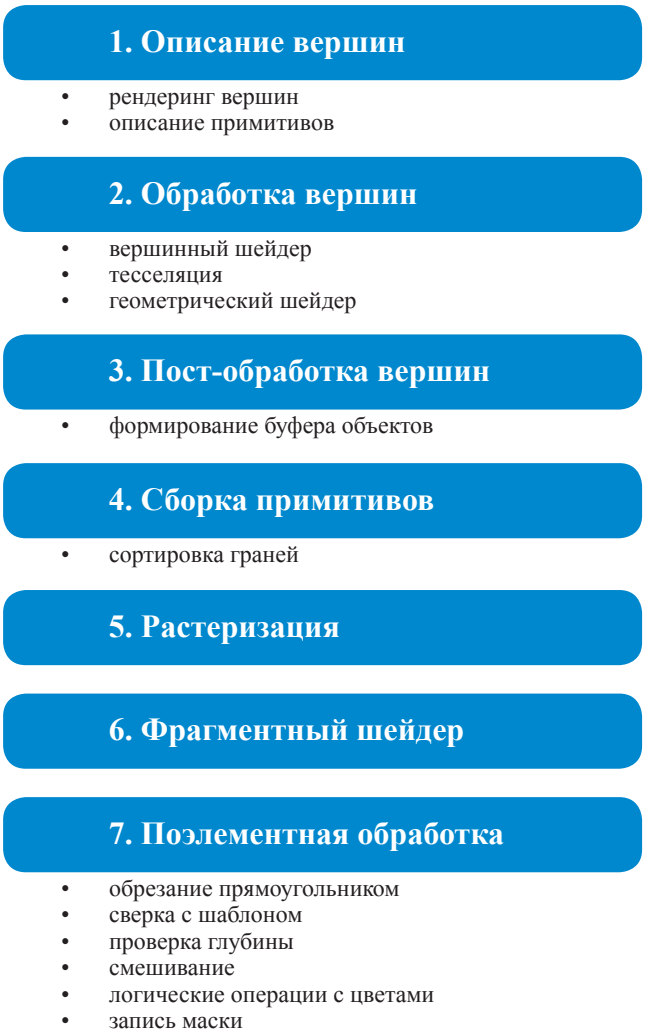
Для цитирования: *Маркелов Г.М.* О компонентах интерфейсов программирования приложений, использующих двух- и трехмерную графику // Изв. вузов «Геодезия и аэрофотосъемка». 2018. Т. 62. № 2. С. 233–236. DOI: 10.30533/0536-101X-2018-62-2-233-236.

Введение

Direct3D и OpenGL — конкурирующие интерфейсы прикладного программирования (API — application programming interface), которые используются в приложениях для рендеринга 2D- и 3D-компьютерной графики. Разработка приложений Direct3D ориентирована на платформу Microsoft Windows. OpenGL

API — открытый стандарт. Это означает, что различные производители аппаратного обеспечения и разработчики операционной системы могут свободно реализовывать OpenGL как часть своей системы. Реализации OpenGL существуют для самых разных платформ. В частности, OpenGL представляет собой доминирующий графический API-интерфейс UNIX-подобных компьютерных систем (образованных под влиянием UNIX).

Direct3D основан на технологии COM (Component Object Model — объектная модель программных компонентов) [1]. COM — это технологический стандарт представления компонентов объекта, которые могут использоваться во многих программах одновременно.



Порядок рендеринга сцены на примере OpenGL [4]
An order of rendering of a scene using OpenGL[4]

С применением других языков программирования невозможно оперировать классами на чистом C++, так как они не имеют стандартизованного бинарного представления из-за использования компиляторами уникального метода декорирования имен. COM же позволяет работать с объектно-ориентированной концепцией из любого языка, его поддерживающего.

OpenGL использует шаблонные функции языка C. Для них существует стандартизированный ABI (application binary interface — двоичный интерфейс приложений), а это значит, что OpenGL может быть использован из любого языка, который поддерживает вызов функций встроенных библиотек (т.е., практически говоря, из любого вообще) [2]. В OpenGL реализована так называемая машина состояний (конечный автомат) [3]. Разработчик задает переменные состояния, и они остаются действительными до следующего изменения. Результат вызовов функций OpenGL зависит от внутреннего состояния и может изменять его. В OpenGL, чтобы получить доступ к конкретному объекту, сначала необходимо установить его в качестве текущего, а затем уже приступать к манипулированию им.

В Direct3D взаимодействие с объектами реализовано иначе: они имеют свой интерфейс, и разработчик обращается к указателям на них (переменным, содержащим адреса ячеек памяти или нулевой адрес). Работа с объектом осуществляется путем вызова методов его интерфейса. Создание объектов происходит как вызовы методов интерфейса. В целом Direct3D предназначен для виртуализации 3D-аппаратных интерфейсов. Он освобождает программиста от адаптирования графического оборудования. OpenGL предназначен для 3D-аппаратной системы рендеринга, которая может быть эмулирована в программном обеспечении. Порядок рендеринга сцены на примере OpenGL представлен на рисунке.

Реализация

OpenGL и Direct3D реализуются в драйвере устройства отображения. Однако существен-

ное различие заключается в том, что Direct3D реализует API в общей среде выполнения (поставляемой Microsoft), которая, в свою очередь, взаимодействует с интерфейсом драйвера устройства низкого уровня. С OpenGL каждый поставщик реализует полный API в драйвере. Это означает, что некоторые функции API могут иметь несколько другое поведение от одного поставщика к другому. Компиляторы шейдеров GLSL (OpenGL Shading Language – язык высокого уровня для программирования шейдеров) разных производителей также могут вести себя по-разному. Тем не менее такой подход имеет своим преимущества: OpenGL развивается с реализацией новых функций производителем в драйвере, который фактически выступает как расширение официальной спецификации — приложения могут использовать новые функции прямо сейчас, не дожидаясь официального релиза. Со временем популярные расширения интегрируются в основную спецификацию. Таким образом, каждая новая версия OpenGL — это старая версия плюс несколько новых интегрированных расширений.

Существуют функциональные различия в том, как работают два API. Значимое различие между API заключается в том, как они управляют аппаратными ресурсами. Direct3D ожидает, что приложение сделает это, а OpenGL заставляет это сделать. Такой подход в OpenGL уменьшает сложность разработки для API, но в то же время увеличивает сложность создания реализации (или драйвера), которая хорошо работает. С Direct3D разработчик должен самостоятельно управлять аппаратными ресурсами, однако реализация проще и у разработчиков есть гибкость в распределении ресурсов наиболее эффективным способом для их приложения.

Производительность

Существенная разница в производительности возникает из-за структуры драйверов оборудования, предоставляемых разработчиками оборудования. В Direct3D драйверы независимых производителей оборудования — это драйверы режима ядра (привилегированный режим), установленные в операционной системе. Часть API, ответственная за режим пользователя (прикладной режим), обрабатывается средой DirectX, предоставляемой Microsoft. В OpenGL драйвер оборудования разделен на две части: пользовательский режим, который реализует API OpenGL, и драйвер режима ядра, который вызывается процессором, работающим в пользовательском режиме. Такой подход не оптимален, потому что вызов операций режима ядра из пользовательского режима требует выполнения системного вызова (т.е. переключения центрального процессора в режим ядра). Это — медленная операция, требующая микросекунду времени на выполнение. За это время центральный процессор не может выполнить никаких операций. Таким образом, сведение к минимуму количества раз, когда эта операция переключения будет вызвана, приведет к повышению производительности.

Ввиду того, что драйвера производителей оборудования для Direct3D выполняются в режиме ядра и пользовательский режим для них недоступен, провести какую-либо оптимизацию невозможно. Так как Direct3D в процессе работы в пользовательском режиме не обладает явным знанием внутренней работы драйвера, он не может эффектно реализовывать маршалинг (marshalling — процесс преобразования информации, хранящейся в оперативной памяти, в формат, пригодный для хранения или передачи). Это означает, что каждый вызов Direct3D, посылающий команды оборудованию, должен также инициировать переключение режима ядра. Поскольку драйвера OpenGL имеют компонент для пользовательского режима, реализация маршалинга становится возможной и повышается производительность. В последних реализациях Direct3D была включена как часть режима ядра, так и пользовательская часть, но из-за этого была потеряна обратная совместимость с предыдущими реализациями.

Применение

Традиционно область применения OpenGL — рынок профессиональной графики. Он является более общим 3D-API, предназначенным для всего спектра графического оборудования: от недорогих графических карт до профессиональной и научной визуализации графики, недоступной для потребителя среднего класса, и предоставляет функции, которые не обязательно эксклюзивные для определенного вида пользователя. Direct3D находит большее применение в продуктах, нацеленных на массового потребителя ввиду целей, заложенных в основу его разработки: поддержка низкоуровневого высокопроизводительного доступа к широко доступному рынку потребительского низкопроизводительного оборудования для рынка компьютерных видеоигр.

Заключение

Если в прошлом еще имелись существенные различия в реализациях представленных API, то с развитием они стерлись. В настоящий момент функционал OpenGL и Direct3D во многом идентичен, поэтому оба API в равной степени могут применяться для решения большинства задач, а основными критериями становятся личные предпочтения разработчика и операционная система: для продукции семейства Microsoft Windows чаще используют Direct3D, а для почти всех других операционных систем — OpenGL. В статье не рассмотрено API Vulkan, которое представляет собой дальнейшее развитие OpenGL, так как появилось оно недавно и не успело зарекомендовать себя как полноценная замена своему предшественнику, а также некоторые другие (Mantle, SDL) по причине их узконаправленности и малой популярности.

ЛИТЕРАТУРА

1. Электронный ресурс: https://msdn.microsoft.com/ru-RU/library/windows/desktop/ff471470(v=vs.85).aspx (дата обращения: 06.08.2017).

2. Электронный ресурс: https://www.opengl.org/about/ (дата обращения: 20.07.2017).

3. Электронный ресурс: https://www.khronos.org/opengl/wiki/Portal:OpenGL_Concepts (дата обращения: 25.07.2017).

4. Электронный ресурс: https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview (дата обращения: 20.07.2017).

REFERENCES

1. URL: https://msdn.microsoft.com/ru-RU/library/windows/desktop/ff471470(v=vs.85).aspx (date of reference: 06.08.2017).

2. URL: https://www.opengl.org/about/ (date of reference: 20.07.2017).

3. URL: https://www.khronos.org/opengl/wiki/Portal:OpenGL_Concepts (date of reference: 25.07.2017).

4. URL: https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview (date of reference: 20.07.2017).