Санкт-Петербургский Государственный Университет

Программная инженерия
Кафедра системного программирования

Орачев Егор Станиславович

Разработка библиотеки для визуализации 3D графики

Курсовая работа

Научный руководитель:
ст. преп. Пименов А. А.

Санкт-Петербург
2019

Оглавление

Введение

Трехмерная графика – раздел компьютерной графики, посвященный методам визуализации объемных моделей и объектов, описанных в 3D (3-dimensional) пространстве. К основным областям применения трехмерной графики можно отнести: системы автоматизированного проектирования, кинематограф, анимацию, видеоигры, симуляторы и многое другое.

Существует несколько способов построения изображения по трехмерным данным, одними из которых являются: растеризация примитивов, трассировка лучей, трассировка пути. Применение конкретного способа с последующей его конфигурацией обуславливается различными факторами:

- Временными ресурсами. При работе в режиме реального времени максимальный интервал генерации изображения составляет приблизительно 33 миллисекунды (соответствует частоте обновления 30 Гц), что позволяет создать непрерывный для человеческого глаза видеопоток.

- Вычислительными ресурсами, предоставляемыми конкретной платформой, вместе со спецификой их использования. К примеру, мобильные устройства накладывают существенные ограничения на энергопотребление и не имеют активное охлаждение.

- Объемом отображаемых данных.

- Качеством отображения, которое может характеризоваться как фото-реалистичное, физически корректное, кинематографичное, "мультяшное" [12] и так далее.

Наличие вышеизложенных факторов в сочетании с многочисленными требованиями реальных проектов, связанных с визуализацией данных, создает ряд проблем при поиске уже готовых графических решений и при их непосредственной интеграции в систему.

В проекте CoreCVS [2] (Computer Vision Primitives Library) в рамках работы над библиотекой для симуляции физики полетов дронов разработчики столкнулись с необходимость визуализации процесса симуляции как для отладки программ, так и для лучшего понимания того, как взаимодействуют объекты между собой. Для решения этой проблемы используется существующая в рамках проекта графическая библиотека, написанная с использованием OpenGL [8] (Open Graphics Library), которая позволяет отображать относительно простую геометрию с базовой моделью освещения. Однако данная библиотека использует в своей основе уже устаревшее подмножество функциональности OpenGL, что создает необходимость значительного рефакторинга для дальнейшего ее использования.

В итоге было принято решение разработать с нуля новую графическую библиотеку в рамках CoreCVS, которая не только бы замещала функциональность своей предшественницы, но и расширяла ее в соответствии с задачами, решаемыми как в текущих, так и в будущих проектах.

Данное решение мотивированно не только стремлением минимизировать количество сторонних зависимостей всего проекта, но и желанием использовать в качестве основы новейшее Vulkan API [11] (Vulkan Application Programming Interface), которое призвано заменить OpenGL в силу его морального устаревания. Анализ существующих решений также показал невозможность интеграции сторонних библиотек ввиду ряда причин, изложенных далее в тексте работы.

1. Постановка задачи

Цель данной работы – реализация компактной библиотеки для визуализации трехмерной графики в реальном масштабе времени с возможностью использовать как низкоуровневый интерфейс для создания специфичных моделей отображения, так и высокоуровневые абстракции, предоставляющие стандартный способ визуализации объектов. Для ее достижения были выделены следующие задачи:

- Исследование предметной области. Сравнение существующих решений
- Изучение Vulkan API
- Разработка и формализация требований к проекту
- Разработка и реализация графической библиотеки с учетом заявленных требований
- Интеграция решения в существующий проект CoreCVS

2. Терминология

- Графическая подсистема (англ. Graphical backend) – модуль программы, реализующий графическую функциональность приложения.
- Шейдер (англ. Shader) – программа, предназначенная для исполнения на графическом процессоре.
- Материал – контейнер, которых хранит описание поверхности объекта в виде набора атрибутов и некоторую программу, которая контролирует процесс обработки данного материала.
- Графический интерфейс (англ. Graphical API) – интерфейс для взаимодействия с графическим ускорителем системы.
- Application Programming Interface (API) – интерфейс прикладного программирования.
- Central Processing Unit (CPU) – центральный процессор.
- Graphical Processing Unit (GPU) – графический процессор.
- Graphical User Interface (GUI) – графический пользовательский интерфейс.

3. Требования к проекту

Концептуально требования к разрабатываемой графической библиотеке можно сформулировать в виде следующего набора тезисов:

- Минимальное количество внешних зависимостей
- Графическая функциональность и ничего более
- Возможность расширения за счет добавления новых моделей освещения, эффектов, иерархий объектов и геометрических моделей

Несмотря на некоторую абстрактность вышеизложенных тезисов, они наиболее емко выражают идею разработки данной библиотеки. Основные функциональные требования, которые вытекают прежде всего из задач в рамках проекта CoreCVS, изложены далее:

- Кроссплатформенность: библиотека компилируется и работает на линейке основных операционных систем (Window, Linux, MacOS).
- Независимость от графического интерфейса: библиотека взаимодействует с видеоускорителем системы через унифицированный интерфейс, который сам по себе является частью проекта. Это сохраняет возможность для дальнейшего расширения и снижает количество вертикальных зависимостей в проекте.
- Поддержка Vulkan: доступ к данному API осуществляется через интерфейс, представленный в предыдущем пункте.
- Независимость от структуры сцены: библиотека поддерживает список видимых объектов и осуществляет их последовательное отображение. Иерархию сцены, зависимости между объектами определят пользователь.
- Система материалов: менеджер материалов осуществляет создание, удаление, изменение материалов, их идентификацию, хранение. Конфигурация отображения осуществляется с помощью материалов.

- Система хранения геометрии: возможность создавать и использовать геометрические объекты с произвольным набором атрибутов.
- Независимость от оконной системы: финальное изображение выводиться на экран, в виджет GUI системы, либо сохраняется во внутреннем буфере.

4. Существующие решения

В данном разделе проводится обзор и сравнение существующих графических решений с открытым исходным кодом, доступным для свободного использования. Критерии обзора вытекают прежде всего из требований, изложенных ранее. Основные факторы для включения в рассмотрение: наличие документации и обучающих материалов, активная поддержка или разработка, интеграция с Vulkan и поддержка широкого класса графических интерфейсов, индексация[1] на хостинге GitHub.

4.1. BSF

BSF (bs framework) – кросс-платформенная C++14 библиотека, разрабатываемая Game Foundry [1]. Предоставляет унифицированную базу для разработки графических приложений реального времени, видеоигр и инструментов программирования. Поддерживаемые графические интерфейсы: Vulkan, OpenGL, DirectX. Функциональность: конвейер экранных пост-эффектов, отложенное, физически корректное затенение, каскадная генерация теней, система материалов.

4.2. Ogre3D

Ogre3D (Object-Oriented Graphics Rendering Engine) – кроссплатформенная C++ графическая библиотека, разрабатываемая открытым сообществом [7]. В качестве графических интерфейсов поддерживает OpenGL, DirectX, WebGL. Функциональность: система материалов и пост-эффектов, менеджмент ресурсов, система анимации, граф сцены.

4.3. Filament

Filament – кросс-платформенная C++ библиотека графики, разрабатываемая Google [4]. Спроектирована таким образом, чтобы быть

[1]Дата обращения: 11.12.2019

максимально компактной и эффективной для Android платформы. Поддерживает OpenGL, Vulkan API, WebGL. Предоставляет C++, Java, JavaScript интерфейсы. Основные графические возможности: физически корректное затенение, освещение на основе изображения, система материалов, камера с физически-основанными параметрами, базовые пост-эффекты.

4.4. Концептуальные недостатки

Несмотря на многочисленную функциональность, которой обладают рассмотренные решения, они не в полной мере удовлетворяют требованиям проекта CoreCVS. В частности, BFS и Ogre3D прежде всего ориентированны на разработку видеоигр, поэтому данные библиотеки сильно завязаны как на иерархической структуре сцены, так и на обработке данных. Они предоставляют собственные форматы геометрии, языки описания шейдеров и методы загрузки ресурсов, таких как модели, текстуры. Данная функциональность избыточна, так как в проекте CoreCVS уже существуют модули по обработке 3D геометрии и изображений.

Проект Filament имеет больший фокус на графическую составляющую, однако он в первую очередь ориентирован на создание физически корректного освещения на Android плафторме, что выражается с существенно меньшей гибкости в настройке как самого освещения, так и экранных эффектов, создание которых практически ограничено на мобильных устройствах ввиду их высокой вычислительной сложности.

Поэтому было принято решение отказаться от интеграции одного из существующих решений и сосредоточиться на разработке новой библиотеки, с учетом всей функциональности, которая в той или иной форме уже представлена в проекте CoreCVS.

5. Vulkan API

Vulkan – это кроссплатформенное API для работы с трехмерной графикой и параллельными вычислениями. Предоставляет C99 совместимый интерфейс для взаимодействия с GPU и спецификацию, которая описывает семантику исполнения программ. Впервые был представлен в феврале 2016 года. В настоящее время поддерживается консорциумом Khronos Group.

5.1. Обзор основных преимуществ

Vulkan является представителем графических интерфейсов нового поколения, таких как Metal [6] и Direct3D [3]. В сравнении со стандартами прошлого поколения, Vulkan имеет ряд новых функциональных особенностей, которые призваны повысить эффективность работы конечных приложений:

- Возможность работать на множестве операционных систем, включаю мобильную платформу.
- Меньшее количество ресурсов, используемых во время работы. Это достигается ручным управления памятью, используемой драйвером как на CPU, так и на GPU, явным созданием и удалением графических конвейеров обработки примитивов.
- Лучшая расширяемость на платформах с несколькими ядрами. Драйвер позволяет формировать команды для GPU из нескольких потоков.
- Универсальное представление GPU программ в бинарном формате SPIR-V [10]. Это позволяет писать шейдерные программы на любом из доступных языков, включая GLSL [9] и HLSL [5], и компилировать их в объектные модули независимо от конечной платформы.

- Унифицированный менеджмент вычислительных ядер на GPU. Это позволяет использовать вычислительный конвейер посредством Vulkan API, не прибегая к сторонним вычислительным библиотекам, таким как Nvidia CUDA и OpenCL.

5.2. Сравнение с другими графическими API

В области разработки графических приложений существует ряд промышленных API, часть из которых уже была упомянута ранее в тексте работы. Direct3D предоставляет схожую графическую функциональность, что и Vulkan, однако он доступен только на платформе от компании Microsoft. Графический API Metal, разрабатываемый корпорацией Apple, также доступен только на устройствах и системах этой компании. OpenGL является открытым стандартом, доступным на многих платформах, исключая мобильные устройства, где используется OpenGLES. Однако развитие данного API затруднено требования обратной совместимости, а поддержка его новых версий уже не осуществляется на некоторых устройствах[2].

Таким образом, было принято решение использовать Vulkan API в качестве основы для реализации данного проекта, так как он удовлетворяет функциональным требованиям разрабатываемой библиотеки и обладает активной поддержкой на основных платформах.

[2]Компания Apple прекратила поддержку новых версий OpenGL в 2018, объявив официально данный API устаревшим на своей платформе

6. Текущий прогресс

Прогресс по проекту на данный момент:

- Проведен обзор и сравнение существующих решений в соответствии с требованиям, предъявляемыми к разрабатываемой графической библиотеке.
- Проведен анализ существующих графических интерфейсов. Выбран Vulkan API как основа для реализации библиотеки по ряду причин, изложенных ранее в тексте работы.
- Выбран стек технологий для реализации проекта (C++11, CMake, Vulkan SDK) с учетом функциональных требований и условий интеграции с библиотекой CoreCVS, в которой в качестве основного языка разработки используется C++11, а в качестве инструмента сборки и управления зависимостями используется CMake.
- Изучены обучающие материалы для работы с Vulkan SDK и официальная Vulkan API документация на уровне, необходимом для реализации проекта.
- Настроен репозиторий проекта, система автоматической сборки библиотеки и запуска тестов, чтобы обеспечить работоспособность конечного решения на нескольких платформах.
- На данный момент ведется активная работа над разработкой архитектуры системы и написанием необходимых модулей библиотеки.

Список литературы

[1] BSF Framework // github. — Access mode: https://github.com/GameFoundry/bsf (online; accessed: 14.11.2019).

[2] Computer Vision Primitives Library // github. — Access mode: https://github.com/PimenovAlexander/corecvs (online; accessed: 11.11.2019).

[3] Direct3D 12 Graphics // Microsoft Online Documents. — 2018. — Access mode: https://docs.microsoft.com/ru-ru/windows/win32/direct3d12/direct3d-12-graphics?redirectedfrom=MSDN (online; accessed: 11.12.2019).

[4] Filament Graphics Library // github. — Access mode: https://github.com/google/filament (online; accessed: 16.11.2019).

[5] High Level Shading Language for DirectX // Microsoft Online Documents. — 2018. — Access mode: https://docs.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl (online; accessed: 20.11.2019).

[6] Metal Programming Guide // Apple Documentation Archive. — 2018. — Access mode: https://developer.apple.com/library/archive/documentation/Miscellaneous/Conceptual/MetalProgrammingGuide/Introduction/Introduction.html (online; accessed: 12.12.2019).

[7] Ogre3D Engine // github. — Access mode: https://github.com/OGRECave/ogre (online; accessed: 15.11.2019).

[8] The Khronos Working Group. OpenGL 4.4 Specification // Khronos Registry. — 2014. — Access mode: https://www.khronos.org/registry/OpenGL/specs/gl/glspec44.core.pdf (online; accessed: 9.12.2019).

[9] The Khronos Working Group. OpenGL Shading Language 4.4 Specification // Khronos Registry. — 2016. — Access mode: https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.4.40.pdf (online; accessed: 20.11.2019).

[10] The Khronos Working Group. SPIR-V 1.5 Specification // Khronos Registry. — 2018. — Access mode: https://www.khronos.org/registry/spir-v/specs/unified1/SPIRV.pdf (online; accessed: 20.11.2019).

[11] The Khronos Working Group. Vulkan 1.1 API Specification // Khronos Registry. — 2019. — Access mode: https://www.khronos.org/registry/vulkan/specs/1.1/html/vkspec.html (online; accessed: 10.11.2019).

[12] Wikipedia. Cel Shading // Wikipedia, the free encyclopedia. — Access mode: https://en.wikipedia.org/wiki/Cel_shading (online; accessed: 9.12.2019).