РАЗРАБОТКА ПОДСИСТЕМЫ 2D РЕНДЕРИНГА ДЛЯ КРОССПЛАТФОРМЕННОГО ИГРОВОГО ДВИЖКА

Лобанов В.В., Коровкин В. А.
НИ ТПУ, ИШИТР, гр. 8К12, студент, vvl45@tpu.ru
НИ ТПУ, ИШИТР, старший преподаватель ОИТ, alcasar@tpu.ru

Аннотация

Данная статья представляет процесс проектирования и реализации подсистемы 2D рендеринга для кроссплатформенного игрового движка с применением библиотеки SDL3 и графическим API OpenGL на языке программирования C++. В работе описывается процесс выбора подходящих технологий для разработки системы, принцип работы конвейера рендеринга, затрагиваются техники для разработки интерактивных графических кроссоперационных приложений.

Ключевые слова: кроссплатформенные приложения, игровой движок, рендеринг, API рендеринга, SDL, OpenGL.

Введение

Игровой движок – комплексное программное обеспечение, которое масштабируемо и может применяться в качестве основы для множества различных игр без значительных модификаций [1]. Игровые движки включают в себя работу с низкоуровневыми концепциями операционной системы, с обработкой графики, звука, пользовательского ввода и в некоторых случаях работу с сетевыми технологиями. Подсистема рендеринга, в свою очередь, является одним из самых вариативных компонентов для реализации, что является привлекательной чертой для проведения исследования на уровне архитектуры этой системы. В данной статье представлена декомпозиция процессов, происходящих при рендеринге, а также реализация подсистемы 2D рендеринга как одного из компонентов кроссплатформенного игрового движка. Большое внимание уделялось совместимости реализации между двумя популярными операционными системами, что было достигнуто за счёт принципов абстракции и выбора кроссплатформенных средств разработки.

Таким образом, целью работы является проектирование и реализация подсистемы 2D рендеринга для кроссплатформенного игрового движка.

Для достижения поставленной цели были выдвинуты следующие задачи:

1. Определение требований для подсистемы рендеринга игрового движка.
2. Проектирование архитектуры подсистемы рендеринга игрового движка.
3. Выбор технологий для реализации требований подсистемы рендеринга игрового движка.
4. Реализация подсистемы рендеринга игрового движка.

Требования к разрабатываемой системе

Подсистема рендеринга – один из самых комплексных в реализации компонентов игрового движка. Перед тем, как приступить к разработке этой подсистемы, необходимо задать требования. В первую очередь были выявлены функциональные требования:

1. Система должна иметь функционал для отображения двумерных графических примитивов без сглаживания:
    a. квадрат.
    b. треугольник.
    c. круг.
    d. капсула.
    e. линия.

2. Система должна поддерживать рендеринг изображений в виде текстур для отображаемых 2D объектов.

3. Система должна позволять менять оттенок текстур для отображаемых 2D объектов.

4. Система должна поддерживать отображение текстур с компонентом прозрачности alpha.

5. Система рендеринга должна поддерживать работу в нескольких окнах.

Также есть конкретизирующие нефункциональные требования:

1. Система рендеринга должна работать в программном потоке, отличного от основного.

2. Система должна поддерживать формат изображений .png для создания текстур.

3. Система рендеринга не должна использовать сторонние библиотеки для непрямого взаимодействия с API рендеринга.

4. Игровой движок должен реализовывать свой полный функционал на 64-разрядной версии операционной системы операционной системы № 1.

5. Игровой движок должен реализовывать свой полный функционал на 64-разрядной версии операционной системы операционной системы № 2.

Дополнительно стоит прокомментировать нефункциональное требование под номером 3. Это требование было выдвинуто с целью ознакомления работы с самим API рендеринга. В случае, если сторонняя библиотека уже будет реализовывать взаимодействие свою реализацию рендеринга, то для создания подсистемы рендеринга нужно будет лишь взаимодействовать API самой библиотеки. При соответствии приведенным выше требованиям подсистема рендеринга включает в себя необходимый функционал, с помощью которого можно будет развивать другие системы игрового движка.

Выбор технологий для реализации

Существование подсистемы рендеринга невозможно без способа отобразить результаты работы системы. В связи с этим в первую очередь необходимо выбрать библиотеку, которая будет покрывать требования, связанные с управлением окнами.

RayLib, Monogame, SFML и другие высокоуровневые фреймворки, которые уже реализуют собственную систему рендеринга не подходят, так как они противоречат нефункциональному требованию №3. Итак, в качестве основных вариантов рассматривались следующие библиотеки:

1. GLFW – библиотека, содержащая в себе функционал для работы с окнами, вводом пользователя. Изначально создавалась для взаимодействия с API рендеринга OpenGL.

2. SDL – библиотека, которая предоставляет более широкий функционал, чем GLFW. В базовую версию библиотеки входят модули для работы с окнами, вводом пользователя, звуком, файловой системой, а также базовый 2D рендерер.

Несмотря на то, что SDL уже имеет свой собственный 2D рендерер, он чаще всего подходит только для простых проектов и его будет крайне тяжело масштабировать. С другой стороны, SDL также предоставляет возможность пользователю создать свой собственный движок рендерера и использовать его для создаваемых окон.

GLFW лучше подойдет для задачи создания окон и манипуляции ими, так как ее функционал сосредоточен лишь на этом. Однако игровой движок включает в себя другие компоненты, которые также должны быть реализованы кроссплатформенно: управление файловой системой, сетевое взаимодействие и многие другие подсистемы. SDL имеет дополнительные библиотеки-модули для покрытия этих потребностей (например, сетевого взаимодействия при помощи SDL_net [2]). Исходя из этого SDL более безопасен при разработке игрового движка, так как все интересующие разработчика части можно реализовать самостоятельно, а для других использовать встроенное решение этой библиотеки.

К тому же при более детальном сравнении SDL и GLFW в области управления окнами было замечено, что в GLFW нет встроенной возможности сохранять контекст движка рендеринга при работе с несколькими окнами [3]. Необходимо копировать существующий

контекст и использовать именно его копию при создании нового окна, это добавляет дополнительные сложности в управление контекстом в том случае, если его необходимо будет изменить. В SDL для контекста есть отдельный класс, который можно передать в конструктор окна, чтобы между окнами контекст сохранялся. Таким образом, было принято решение выбрать SDL в качестве средства для создания окон и манипуляции ими.

Перейдем к выбору API рендеринга. Среди кроссплатформенных графических API наиболее распространенными считаются OpenGL и Vulkan. OpenGL предоставляет не самые лучшие показатели по производительности [4], однако именно за счет кроссплатформенности и относительной простоты использования имеется склонность использовать его в качестве начального инструмента для создания движка рендеринга.

Vulkan представляет собой более современный графический API, который дает больше свободы при работе с графикой [4], что делает его более комплексным инструментом по сравнению с OpenGL. На данный момент Vulkan находится в активном развитии и является перспективным API для изучения в будущем. Для создания базового движка рендеринга нет необходимости использовать продвинутые возможности Vulkan, поэтому в качестве API рендеринга будет использоваться OpenGL. Однако с применением принципа абстракции Vulkan планируется внедрить в игровой движок в качестве второго варианта API, который может использовать движок рендеринга.

Таким образом, перечень используемых технологий включает в себя SDL для создания манипуляциями окнами и API рендеринга OpenGL.

Описание подсистемы рендеринга

Для реализации системы рендеринга, необходимо понимать ее внутренние процессы. В качестве примера рассмотрены те действия, которые должны произойти, чтобы отобразить простую текстуру на окне. Этапы существенно упрощены, так как система создается только для двухмерного пространства.

1. Подготовка необходимых данных, текстур и шейдеров. На этом этапе необходимо подготовить всю геометрию для API рендеринга. В нашем случае будет достаточно простого квадрата, поверх которого будет отображена соответствующая текстура. Текстуры и шейдеры будут загружаться из файлов в режиме реального времени.

2. Загрузка в API рендеринга информации и создание контекста. Здесь происходит перенос всей информации из первого этапа в API рендеринга. Эту часть необходимо будет занести в слой абстракции от API рендеринга.

3. Подключение шейдеров. Шейдеры представляют собой программы, которые исполняются на графическом процессоре. Для отображения текстуры необходимо будет использовать вершинный и фрагментный шейдеры. Вершинные шейдеры выполняются для каждой вершины отображаемого объекта, а фрагментные шейдеры – для каждого пикселя выходного изображения. В связи с этим, большую часть вычислений более эффективно проводить в вершинном шейдере и затем передавать информацию в фрагментный шейдер, чтобы на этом этапе происходили только необходимые действия, такие как покраска пикселей в нужный цвет и отображение текстур.

4. Камера. Камера представляет собой объект в пространстве, с позиции которого в окно выводится проекция объектов. В целях 2D рендеринга используют ортогональную проекцию, которая не учитывает расстояние объектов до точки обзора. В контексте системы рендеринга камера будет представляться матрицей проекции, которая будет загружаться в шейдер для модификации отображаемого результата.

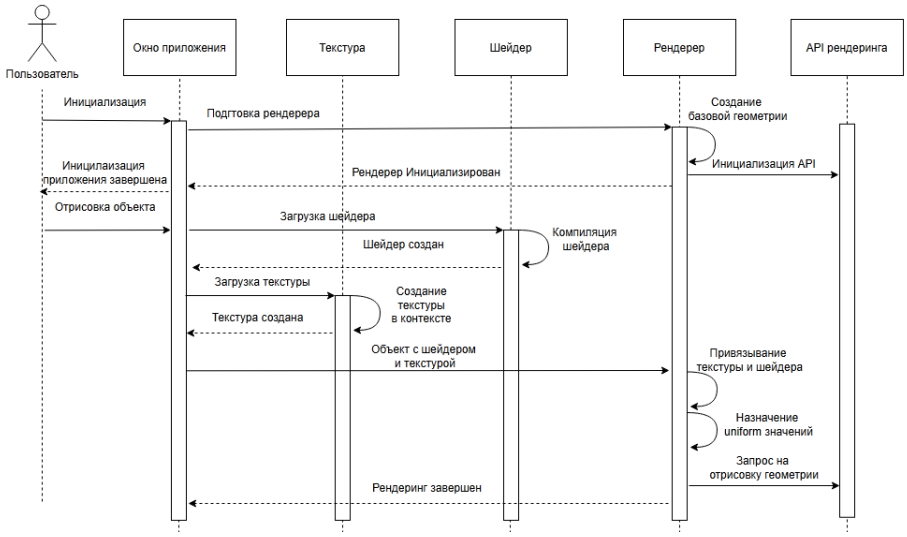

Рис. 1. Диаграмма последовательностей для описания работы подсистемы рендеринга

Так как реализация отображения элементов в окне зависит от API рендеринга, необходимо использовать принцип абстракции. Классы, с обозначением «(virtual)» являются абстракцией и нуждаются дополнительном классе-реализации, использующем конкретный API рендеринга. В результате анализа необходимых компонентов были выделены следующие компоненты движка рендеринга:

1. VertexBuffer (virtual) – массив, который содержит информацию о вершинах. Чаще всего это координаты, но также сюда могут входить цвета, нормали и другие свойства вершин.

2. BufferLayout (virtual) – принцип расположения элементов в массиве вершин. В нем определяется, как движок рендеринга будет считывать данные, предоставляемые ему для рендеринга.

3. IndexBuffer (virtual) –массив с индексами вершин, которые необходимо создать. Использование этого элемента позволяет сэкономить вычислительные мощности при отображение любых элементов, состоящих из 2 полигонов и более.

4. VertexArray (virtual) – массив, который содержит в себе несколько VertexBuffer с одинаковым оффсетом. Этот компонент является уникальным для API рендеринга и присущ только OpenGL.

5. Shader (virtual) – код, который будет исполняться на графическом процессоре. Поддерживает механизм uniform значений, которые передаются напрямую из процессора, чтобы оптимизировать нагрузку между процессором и видеокартой.

6. Texture (virtual) – структура, которая будет содержать информацию о файле-текстуре, который будет отображаться с помощью движка рендеринга.

7. Renderer – основной класс, предоставляющий доступ к отображению элементов в окне.

8. RendererAPI (virtual) – содержит в себе специфические элементы и реализацию методов для разных графических API.

9. RenderCommand – команды для рендеринга. Команды рендеринга напрямую вызывают реализацию методов из RenderingAPI. На данный момент включают в себя следующие:

a. EnableBlending – включение режима для отображения текстур с прозрачностью.

b. SetActiveWindow – установка окна, которое на данный момент обновляется.

c. SetClearColor – установка цвета фона окна.

d. Clear – очистка кадра.

e. DrawIndexed – отрисовка элементов в соответствии с заданным массивом вершин.

Представленные выше компоненты также отображены на диаграмме компонентов ниже.

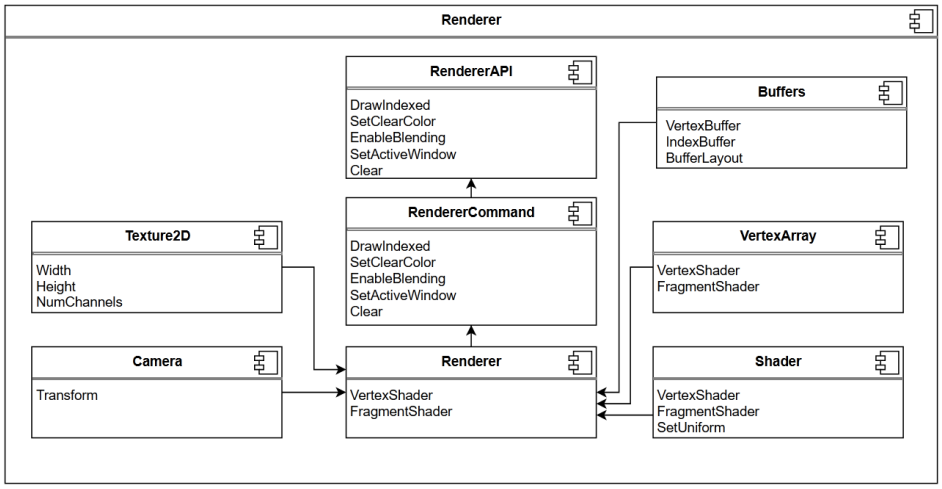

Рис. 2. Диаграмма компонентов для подсистемы рендеринга

Подобная абстракция не является совершенной: на данный момент для работы с движком рендеринга используется VertexArray, который используется только в OpenGL [5]. В дальнейшем планируется изменить принцип работы движка рендеринга, чтобы он действительно не зависел от используемого API. Эта задача будет решаться только при внедрении API рендеринга Vulkan, массивы вершин обязательны для работы OpenGL.

Результаты

Полученная система позволяет работать отображать графические примитивы с текстурами с помощью API рендеринга OpenGL и окон SDL. Благодаря принципам абстракции для взаимодействия со сторонними библиотеками используется только API разработанной системы. Тем не менее, для внедрения других графических API потребуется изменить класс VertexBuffer так, чтобы он удовлетворял массивам данных этих API.

Движок рендеринга работает на основе отображения объектов с соответствующими массивами вершин, матрицами трансформации, шейдерами и текстурами (рис. 4). Ортогональная камера позволяет наблюдать объекты на сцене в ортогональной проекции, передвигаться и вращаться относительно оси z. Данная система рендеринга является частью проекта игрового движка, выложен в свободный доступ и доступен по ссылке на ресурсе [6].

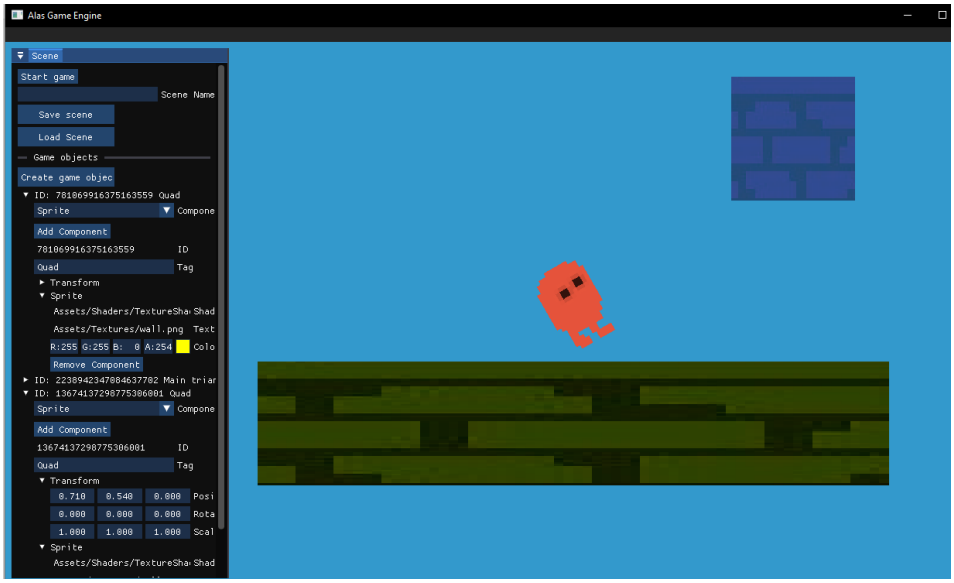

Рис. 3. Использование подсистемы 2D рендеринга для игрового движка

Заключение

В результате проведенной работы был проведен процесс проектирования и реализации подсистемы 2D рендеринга для кроссплатформенного игрового движка.

Наиболее важным принципом в разработке кроссплатформенных приложений является абстракция. Она позволяет создавать системы, реализацию которых можно определять в зависимости от используемой платформы или библиотеки. Принцип абстракции был применен в системах для управления окнами, вводом и движка рендеринга.

Не менее ответственно стоит подходить к выбору сторонних SDK при разработке кроссплатформенного программного обеспечения, так как именно они определяют список поддерживаемых платформ. Наиболее оптимальным подходом будет разработка основной системы с применением SDK с максимальной совместимостью с целевыми платформами, а затем реализовывать системы, которые оптимизированы для каждой конкретной платформы.

Список использованной литературы

1. Грегори Джейсон. Игровой движок. Программирование и внутреннее устройство. Третье издание. – СПб. : Питер, 2021. – 1136 с.

2. SDL3 Libraries // SDL Wiki. – 2024. [Электронный ресурс]. – URL: wiki.libsdl.org/SDL3/Libraries (дата обращения: 12.04.2025).

3. Context handling guide // GLFW. – 2018. [Электронный ресурс]. – URL: glfw.org/docs/3.0/context.html (дата обращения: 12.04.2025).

4. What’s the Difference Between Vulkan, OpenGL, and DirectX // PC Gazer – 2023. [Электронный ресурс]. – URL: pcgazer.com/2023/02/01/whats-the-difference-between-vulkan-opengl-and-directx/ (дата обращения: 12.04.2025).

5. Vries J. de. Learn OpenGL: Learn modern OpenGL graphics programming in a step-by-step fashion. – Netherlands: Kendall & Wells. – 2020. – 35 с.

6. Alas Game Engine // Github. – 2025. [Электронный ресурс]. – URL: github.com/Ssssssaber/Alas-game-engine (дата обращения: 12.04.2025).